Fusil Documentation

Release 1.5

Victor Stinner

May 19, 2016

Contents

Fusil the fuzzer is a Python library used to write fuzzing programs. It helps to start process with a prepared environment (limit memory, environment variables, redirect stdout, etc.), start network client or server, and create mangled files. Fusil has many probes to detect program crash: watch process exit code, watch process stdout and syslog for text patterns (eg. "segmentation fault"), watch session duration, watch cpu usage (process and system load), etc.

Fusil is based on a multi-agent system architecture. It computes a session score used to guess fuzzing parameters like number of injected errors to input files.

Available fuzzing projects: ClamAV, Firefox (contains an HTTP server), gettext, gstreamer, identify, libc_env, libc_printf, libexif, linux_syscall, mplayer, php, poppler, vim, xterm.

Fusil is an opensource project written in Python under GNU GPLv2 license.

- Website
- Fusil at bitbucket
- Fusil at Python Cheeseshop (PyPI)

CHAPTER 1

Features

Why using Fusil instead your own hand made C script?

- Fusil limits child process environment: limit memory, use timeout, make sure that process is killed on session end
- Fusil waits until system load is load before starting a fuzzing session
- Fusil creates a session directory used as the process current working directory and Fusil only creates files in this directory (and not in /tmp)
- Fusil stores all actions in fusil.log but also session.log for all actions related of a session
- Fusil has multiple available probes to compute session score: guess if a sessions is a succes or not
- Fusil redirects process output to a file and searchs bug text patterns in the stdout/stderr (Fusil contains many text patterns to detect crashes and problems)

CHAPTER 2

Use Fusil

2.1 Fusil user guide

2.1.1 Introduction

Fusil projects contains a set of fuzzers called "fusil-..." (eg. fusil-firefox). Each fuzzer has its own command line options, use different input data (text, picture, video, etc.), and test a different program or library. For each crash, Fusil stores all files used to run the session in a dedicated directory and generate a script to replay the session (eg. reproduce a crash in gdb).

The document presents how to run a fuzzer and analyze the crashs.

2.1.2 Run a fuzzer

Example with the execution of gettext fuzzer:

```
$ fusil-gettext
Fusil version 0.9.1 -- GNU GPL v2
http://fusil.hachoir.org/
(...)
[0][session 13] Start session
[0][session 13] ------------------------------------------------------------
[0][session 13] PID: 16989
[0][session 13] Signal: SIGSEGV
[0][session 13] Invalid read from 0x0c1086e0
[0][session 13] - instruction: CMP EDX, [EAX]
[0][session 13] - mapping: 0x0c1086e0 is not mapped in memory
[0][session 13] - register eax=0x0c1086e0
[0][session 13] - register edx=0x00000019
[0][session 13] ------------------------------------------------------------
[0][session 13] End of session: score=100.0%, duration=3.806 second
(...)
Success 1/1!
Project done: 13 sessions in 5.4 seconds (414.5 ms per session), total 5.9 seconds, aggresssivity: 1
Total: 1 success
Keep non-empty directory: /home/haypo/prog/SVN/fusil/trunk/run-3
```

A fuzzer does not require any human interaction: it generates data, start the target software, store files if the software crashed, and restart with new data. By default, the fuzzer stops after the first "success" (crash). You can interrupt the fuzzer with the key combinaison CTRL+c.

2.1.3 Fuzzer list

Applications

- fusil-clamav: ClamAV antivirus
- fusil-firefox: Any content embedded in an HTML page in Firefox (JPEG/PNG pictures, video, Flash, etc.)
- fusil-imagemagick: Image Magick (image manipulation), use identify or convert program
- fusil-mplayer: Mplayer (audio/video player)
- fusil-ogg123: Ogg/Vorbis music player, use the programs ogg123 (default) or ogginfo
- fusil-vlc: VLC (audio/video player)

Libraries

- fusil-gettext: Gettext library (text internationalization aka i18n), part of the GNU C library
- fusil-gstreamer: Gstreamer (audio/video codec and player), use the program gst-launch-0.10 with playbin or build a pipeline
- fusil-libc-printf: printf() function of the system C library
- fusil-poppler: popper (PDF) library used by Evince and Kpdf programs

Programming language

- fusil-php: PHP language
- fusil-python: Python language

Other

- fusil-wizzard: Generic Linux command line fuzzer
- fusil-zzuf: wrapper to the zzuf fuzzer (mutation of input files and network sockets)

2.1.4 Fuzzer options

Each fuzzer has its own options plus common Fusil options. Use –help option (eg. “fusil-firefox –help”) to get the list of all available options.

Common options:

- –success=50: Stop after 50 success (default: 1)
- –sessions=1000: Stop after 1000 sessions (default: unlimited)
- –fast/–slow: Run faster/slower (more/less CPU intensive). WARNING: –fast may introduce false positive with some fuzzers.
- -v or –verbose: Verbose mode, display more details about the fuzzer execution. Use –debug to get all messages, and –quiet for less messages.

2.1.5 Analyze a crash

After a crash, Fusil stores all files used to execute the program in a dedicated directory. Go into this directory to analyze the crash.

Directory names

The directory tree looks like:

```
gettext-4/
gettext-4/exitcode-1/
gettext-4/invalid_read-null/
gettext-4/invalid_read-null-2/
gettext-4/session-609/
```

The main directory name (gettext-4) is based on the fuzzer name (fusil-gettext in our example), and duplicate names ends with a number (gettext-2, gettext-3, ...). The name of a crash directory contains informations about the crash:

- exitcode-1: gettext exited with the code 1
- invalid_read-null: gettext was killed because of an invalid memory read from the NULL addresss
- session-609: Fusil doesn't know any useful information about the crash, the name is just the number of the session

Duplicate directories have also a number as suffix (invalid_read-null-2, invalid_read-null-3, ...).

List crash directories gives a global view of the crashs. You can also see duplicate crashs: invalid_read-null and invalid_read-null-2 should be the same bug.

If the fuzzer is still running, you may see a temporary directory which will be destroyed at the end of the session (eg. session-130, session-131, session-132, ...).

Read the session.log

A session directory always contains a file called "session.log" which contains events from the fuzzer. The file always contains useful informations about input data and usually also informations about the crash.

Read the stdout

Most fuzzers create a process. The process output, standard output and error streams (stdout/stderr), is written into a file called "stdout". It's the second most useful file to analyze a crash.

The output may be truncated before the program crash because the output buffer was not flushed before the crash. fusil-python fuzzer uses the Python command line option "-u" to get unbuffered output.

Replay the crash

To replay a crash, Fusil creates a script called replay.py to replay (reproduce) the crash. It starts the process with the same command line options, environment variables, but also the same process limitations (limit memory, start under a different user, ...) to avoid a denial of service of your computer.

Just type "./replay.py" to replay the session. To get more information, you can run the process in the gdb debugger (–gdb option) or Valgrind (–valgrind).

There are other options: use "./replay.py –help" to get the list of all options.

2.1.6 Fusil configuration

You can configure Fusil using a fusil.conf file in your configuration directory ($XDG_CONFIG_HOME environment variable or ~/.config/). Template file:

```
################################################################
# General Fusil options
################################################################
[fusil]

# Maximum number of session (0=unlimited)
session = 0

# Maximum number of success before exit (0=unlimited)
success = 1

# Minimum score for a successful session
success_score = 0.50

# Maximum score for a session error
error_score = -0.50

# Maximum memory in bytes (0=unlimited)
max_memory = 0

# (Normal) Maximum system load
normal_calm_load = 0.50

# (Normal) Seconds to sleep until system load is low
normal_calm_sleep = 0.5

# (Slow) Maximum system load
slow_calm_load = 0.30

# (Slow) Seconds to sleep until system load is low
slow_calm_sleep = 3.0

# xhost program path (change X11 permissions)
xhost_program = xhost

################################################################
# Debugger used to trace child processes
################################################################
[debugger]

# Use the debugger?
use_debugger = True

# Enable trace forks option
trace_forks = True


################################################################
# Child processes options
################################################################
[process]

# Dump core on crash
core_dump = True
```

```
# Maximum user process (RLIMIT_NPROC)
max_user_process = 10

# Default maximum memory in bytes (0=unlimited)
max_memory = 104857600

# Change the user (setuid)
user = fusil

# Change the group (setgid)
group = fusil

# Use a probe to watch CPU activity
use_cpu_probe = True
```

2.1.7 Safe environment

To avoid computer crash, Fusil limits its own process but also child processes resources.

Limitation to Fusil process are set in Application.init():

- Process priority: 19
- Limit CPU usage: project.step_sleep (in second, default: 10 millisecond)
- Limit memory: appplication.max_memory (in bytes, default: 100 MB)
- Limit time: project.session_timeout (in second, default: disabled). See also 'time.txt' documentation.

Limitation of child process:

- Process priority: 19
- Limit memory: default limit of 100 MB
- Limit time: default timeout of 10 seconds

Limitation of attached process:

- Check used memory: default limit of 100 MB

2.1.8 Be nice with CPU

Project waits until system CPU load is below 30% before creating a new session. It stops Fusil if system is not calm enough after 60 seconds.

2.2 Install Fusil

2.2.1 Install Linux packages

- Debian: Fusil Debian package
- Ubuntu: Fusil in Intrepid. WARNING: Fusil package 0.8 depends on python-ptrace but dependency is missing!
- Mandriva: Fusil Mandriva package (Cookier: 1.0.0)
- OpenEmbedded: Fusil entry in the bug tracker (see also the recipe in git)

- Arch Linux: python-fusil package
- MacPort: Fusil MacPort
- Gentoo: dev-python/fusil

2.2.2 Install from source code

Download source code

Stable version:

- Download fusil-1.4.tar.gz (MD5: 04c3844a112f788b34bde5f2ab6ee114)

See also Fusil on Python Package Index (PyPI)

Fusil 1.3 depends on Python 2.5+ and python-ptrace 0.6+.

Download the last version (developer version) with Mercurial:

```
hg clone https://bitbucket.org/haypo/fusil/
```

You can also browse Fusil source code.

Fusil dependencies

- Python 2.6+ http://python.org/
- python-ptrace 0.7+ http://python-ptrace.hachoir.org/
- GCC needed by compileC() function from fusil.c_tools: http://gcc.gnu.org/

Optional dependencies:

- rst2html program, part of docutils Python project Debian package: python-docutils http://docutils.sourceforge.net/
- Xlib Python module, required by fusil.xlib module and used by fusil-firefox: http://python-xlib.sourceforge.net/

Running on Windows:

- win32api for Python: http://starship.python.net/crew/mhammond/win32/

Installation

Fusil uses the user "fusil" and the group "fusil" to run child processes to avoid remove an arbitrary file or kill an arbitrary process.

Type as root:

```
./setup.py install
```

Or using sudo program:

```
sudo python setup.py install
```

2.2.3 Projects dependencies

Each project may require external program or special environment:

- Linux operating system: *linux_ioctl* and *linux_syscall* projects are specific to Linux
- Mplayer program: needed by *mplayer* project
- MySQL server and MySQL command line client: needed by *mysql* project
- etc.

2.2.4 Try fusil

You can use Fusil without installation by changing PYTHONPATH: Fusil module have to be part of PYTHONPATH. Go to Fusil parent directory and type:

```
export PYTHONPATH=$PYTHONPATH:$PWD
```

Then you can use any fuzzer. Example:

```
./fuzzers/fusil-gettext
```

CHAPTER 3

Develop with Fusil

3.1 HOWTO: Write a fuzzer using Fusil

3.1.1 Identify your target

Before writing a fuzzer, you have to analyze your target: identify input vectors and list methods to watch target activity.

Identify input vectors

Example of input vectors:

- the command line
- environment variables
- files
- sockets (TCP, UDP, UNIX, etc.)
- etc.

Some tools help in this job:

- strace to watch syscalls (eg. “strace -e open program ...”)
- ltrace to watch library function calls (eg. “ltrace -e getenv program ...”)
- netstat to watch network connections (sockets)

Watch target activity

Example target probes:

- standard output: stdout and stderr
- process exit status (exit code or signal used to kill the process)
- logging file (eg. /var/log/clamav/clamav.log)
- network ping
- run the process in a debugger to trace faults (signals)
- ...

3.1.2 Hello world fuzzer

First draft

Let's start with the most simple fuzzer. Create a file "hello.py" with this code:

```
#!/usr/bin/env python
from fusil.application import Application
from fusil.process.create import ProjectProcess

class Fuzzer(Application):
    def setupProject(self):
        ProjectProcess(self.project, ['echo', 'Hello World!'])

if __name__ == "__main__":
    Fuzzer().main()
```

Source code details:

- it's a Python (executable) program
- you have to inherit Application class and define the setupProject() method
- setupProject() creates the project "agents"
- the process agent is not stored in a variable: it's transparently register in Fusil agent list
- main() method starts the fuzzer, activate agents, execute the different sessions, etc.

Let's try this amazing fuzzer! Stop it using CTRL+c:

```
$ chmod +x hello.py
$ ./hello.py
Fusil version 0.9.1 -- GNU GPL v2
http://fusil.hachoir.org/
Use directory: /home/haypo/prog/SVN/fusil/trunk/run-1
[0][session 1] Start session
^C
[0][session 1] Project execution interrupted!
[0][session 1] Project done: total 5.6 seconds, aggresssivity: 1.0%
[0][session 1] Total: 0 success
[0][session 1] Process 8210 exited normally
```

Working example

This fuzzer is useless because it doesn't know if the process is still active or not. We have to add some probes: WatchProcess to watch the exit status and WatchStdout to watch the standard output:

```
from fusil.process.watch import WatchProcess
from fusil.process.stdout import WatchStdout

class Fuzzer(Application):
    def setupProject(self):
        process = ProjectProcess(self.project, ['echo', '"Hello World!"'])
        WatchProcess(process)
        WatchStdout(process)
```

Let's retry with a maximum of 1 session:

```
$ ./hello.py  --sessions=1
Fusil version 0.9.1 -- GNU GPL v2
http://fusil.hachoir.org/
Use directory: /home/haypo/prog/SVN/fusil/trunk/run-1
[0][session 1] Start session
[0][session 1] Process 8222 exited normally
Stop! Limited to 1 sessions, use --session option for more
Project done: 1 sessions in 0.1 seconds (63.8 ms per session), total 1.1 seconds, aggresssivity: 2.0%
Total: 0
```

See "Process 8222 exited normally" line: the session stopped because the process is done.

Real fuzzer

Execute the echo program with a constant argument is not so funny, we have to add some random arguments. ProjectProcess can't be used because it creates automatically the process on session start. So we will use CreateProcess which allows to do some operations before the process start. The patch:

```
from fusil.process.create import CreateProcess
from fusil.bytes_generator import BytesGenerator, ASCII0
from random import randint, choice

class EchoProcess(CreateProcess):
    OPTIONS = ("-e", "-E", "-n")

    def __init__(self, project):
        CreateProcess.__init__(self, project, ["echo"])
        self.datagen = BytesGenerator(1, 10, ASCII0)

    def createCmdline(self):
        arguments = ['echo']
        for index in xrange(randint(3, 6)):
            if randint(1, 5) == 1:
                option = choice(self.OPTIONS)
                arguments.append(option)
            else:
                data = self.datagen.createValue()
                arguments.append(data)
        return arguments

    def on_session_start(self):
        self.cmdline.arguments = self.createCmdline()
        self.createProcess()
```

Details:

- The most important method is on_session_start(): it's called when the event "session_start" is emitted. This method recreates the command line arguments and create the process.
- createCmdline() creates random arguments for the echo program: 25% of options (-e, -E or -n) and 75% of random strings (ASCII characters in range 1..255)
- __init__() is overwrite to create the datagen attribute
- fusil.bytes_generator module contains tools (classes) to generate random byte strings

Test examples

See examples/ directory. It contains hello-world (first version of Hello World) and good-bye-world (working echo fuzzer).

3.1.3 Inject data to your target

Command line and environment variable

See process.rst documentation.

Files

See mangle.rst documentation.

To write a C program, see c_tools.rst documentation

Network

See network.rst documentation.

3.1.4 Watch target activity

Standard output and logging files

Use WatchStdout and FileWatch. See also file_watch.rst documentation.

Process exit status

Use WatchProcess and read process.rst documentation.

3.2 Agent

3.2.1 Multi agent system (MAS)

Univers agent is responsible to execute all agents. Univers is stopped using univers_stop() event.

A session can be stopped using session_stop() event.

Main MAS events:

- project_start(): event received at first step but only for the first session of a project
- session_start(): event received at first step on a session
- session_done(score): event received at the last step of a session

3.2.2 Agent

An agent is an object able to send/receive messages to/from other agents. Agents have a limited vision of the whole application: agents do not see them each other. For pratical reasons, they have access to Application, Project and Session objets.

3.2.3 Active or inactive

By default, an agent is inactive. It does not receive any event and is not allowed to send messages. Calling activate() method does active then agent and then call init() method. To disable an agent, use deactive() which calls deinit() method.

Sum up of attributes and methods:

- is_active: boolean
- activate(): enable the agent, call init()
- deactivate(): disable the agent, call deinit()
- init(): create objects (eg. open file)
- deinit(): destroy objects (eg. close file)

3.2.4 live()

When an agent is active, it's live() method is called at each session step. The method have to be fastest as possible, so don't use any blocking function (eg. select() before read()).

3.2.5 Events

Method related to message handling:

- readMailbox(): read messages and call related agent event handler
- send(event, *arguments): send an event to other agents

You don't have to call readMailbox(), this job is done by Session.executeObject().

To register to a message, just add a method to your class with the prototype:

```
def on_EVENT(self, *arguments): ...
```

Example of method called on session start:

```
def on_session_start(self):
    ...
```

3.2.6 Other attributes

- name (str): Agent name (should be unique in the whole application)
- agent_id (int): Unique identifier in the whole application (integer counter starting at 1)
- logger: Logger object used by methods debug(), info(), ...
- mailbox: Mailbox used to store message until readMailbox() is called

3.2.7 Logging

To write string to logger, use methods:

- debug(message): DEBUG level
- info(message): INFO level
- warning(message): WARNING level
- error(message): ERROR level

3.2.8 Score

ProjectAgent and SessionAgent have a method getScore() which return the agent score. Default value is None (agent has no score). An agent has also 'score_weight' attribute (default value: 1.0) which is used to compute final agent score:

    minmax(-1.0, agent.getScore() * agent.score_weight, 1.0)

3.3 Fusil architecture

3.3.1 Architecture

Fusil is a multi-agent system (MAS): it uses simple objets called "agents" exchanging messages though asynchronus "message (mail) transfer agent" (MTA). This architecture allows the whole project to be very modular and very customisable.

Each agent have a live() method called at each session "step", but also event handler. An event has a name and may contains arguments. The name is used in agent method name: eg. "on_session_start()" method is called when the session starts.

Some agents do change the environment and some other watchs for errors and strange behaviour of programs.

3.3.2 Action agents

- CreateProcess: create a process
- StdoutFile: created by CreateProcess to store process output
- MangleFile: generate an invalid file using valid file
- AutoMangle: MangleFile with autoconfiguration based on aggressivity factor

Network:

- NetworkClient / NetworkServer: network client / server
- TcpClient: TCP network client
- UnixSocketClient: UNIX socket client
- HttpServer: HTTP server

3.3.3 Probes

- FileWatch: watch a text file, search specific text patterns like "segmentation fault"
- CpuProbe: watch CPU used by the process created by CreateProcess
- ProcessTimeWatch: watch process execution duration
- WatchStdout: watch process output (stdout)
- WatchProcess: watch process created by CreateProcess
- AttachProcess: watch running process
- Syslog: watch /var/log/messages and /var/log/syslog files

3.4 Network

3.4.1 Network server

- from fusil.network.server import NetworkServer
- from fusil.network.tcp_server import TcpServer
- from fusil.network.http_server import HttpServer

On new client connection, a ServerClient object is created.

3.4.2 Network client

- from fusil.network.client import NetworkClient
- from fusil.network.tcp_client import TcpClient
- from fusil.network.unix_client import UnixClient

TpcClient methods:

- __init__(project, host, port, timeout=10.0): constructor
- recvBytes(max_size=None, timeout=0.250, buffer_size=1024): Read max_size bytes by chunks of buffer_size bytes, stop after timeout seconds
- sendBytes(bytes): send bytes on socket. Return False on error, True on success

TpcClient attributes:

- host: host name/IP address
- port: port number
- timeout: socket timeout (in second)
- tx_bytes: number of bytes sent to host
- socket: socket object (set to None on error)

3.5 Fusil process agents

3.5.1 Create your process

Today, Fusil only supports UNIX process.

runCommand()

runCommand() function from fusil.process.tools is a wrapper to os.system(). It raises a RuntimeError() if the command failed (exit code is different than zero).

```
>>> from fusil.process.tools import runCommand
>>> from fusil.mockup import Logger
>>> logger = Logger()
>>> runCommand(logger, ['echo', 'Hello World!'])
```

CreateProcess and ProjectProcess

This action class does prepare the environment for a process and then use subprocess.Popen() to create the process object.

ProjectProcess creates the process with 'session_start()' event whereas CreateProcess have to be inherited to add your own session handler calling self.createProcess() method.

Attributes:

- env: Environment() object used to create environment variables
- stdin: if value is False, use a null device as stdin.
- stdout: Stdout type:
  - 'file' (default): stdout and stderr are written in a file
  - 'null': use a a null device as stdout and stderr
- cmdline: CommandLine() object used to create the command line
- timeout: maximum execution duration of the process in second (default: 10 second). If the timeout is reached, the process is directly killed with SIGKILL signal. Use None value to disable timeout.
- max_memory: Limit of memory grow in bytes (default: 100 MB). Use None value to disable memory limitation.
- popen_args: Dictionary of supplementary options to Popen() constructor (default: see source code).

Example:

```
>>> from fusil.mockup import Project
>>> project = Project()
>>> from fusil.process.create import CreateProcess
>>> null_stdout = CreateProcess(project, ['/bin/ls', '-lR'],
...     stdout='null', timeout=None)
...
```

Environment

This class is responsible to create new process environment variables. It does copy some variables from Fusil environment and allow to set/generate some others. On Linux, no variable is copied. On Windows, only SYSTSEMROOT is copied. You may copy variables like LANGUAGE, LANG, PATH, HOME or DISPLAY using:

```
env.copy('DISPLAY')
```

Methods:

- set(name, value): set a fixed value variable value
- add(var): add a new fuzzy variable
- copy(name): copy an environment variable value (only if it's set)
- copyX11(): copy X11 environment variables (don't use directly! use process.setupX11())

To set/generate a variable, use on of these classes:

- EnvVarValue: fixed value
- EnvVarLength: generate long value to find buffer overflow.
- EnvVarInteger: generate signed integer value
- EnvVarIntegerRange(min, max): generate an integer value in the range [min; max]
- EnvVarRandom: generate random bytes, use all byte values except nul byte (forbidden in environment variable value)

Attributes (EnvVarValue has only name attribute):

- name: Variable name, it can be a list
- min_length: Minimum number of bytes (default: 0)
- max_length: Maximum number of bytes (default: 2000)
- bytes: bytes set (default: set('A'))

Variable name is a string but it can be a tuple or list of strings. Examples:

```
env.set('LANGUAGE', 'fr')
env.set(('LANGUAGE', 'LANG', 'LC_ALL'), 'C', max_count=2)
env.add(EnvVarLength('PATH'))
env.add(EnvVarRandom('HOME'))
env.add(EnvVarInteger(['COLUMNS', 'SIZE']))
```

Command line

CreateProcess object has cmdline attribute of type CommandLine. This object has only one attribute: arguments which is a list of string.

Linux graphical application (X11)

To be able to use a graphical application on Linux, use:

```
>>> process = CreateProcess(project, ['ls', '-l'])
>>> process.setupX11()
```

It allows fuzzer process to access to X11 server and copy needed environment variables (HOME, DISPLAY and XAUTHORITY). See Environment.copyX11().

3.5.2 Watch process activity

WatchProcess

Watch the process created by CreateProcess: wait until the process exit or the process death (killed by a signal). It uses the exit status to compute the probe score:

- if process time has been reached the timeout, probe score is: 'timeout_score' (default: 100%)
- if exit code is nul, score is 'default_score' (default: 0%)
- if exit code is not nul, score is 'exitcode_score' (default: 50%)
- if process has been killed by a signal, score is 'signal_score' (default: 100%)

WatchProcessStdout

WatchProcessStdout inherits on FileWatch: it looks for error message patterns in process stdout (and stderr if process is configured to write stderr to stdout).

AttachProcessPID

Watch an existing process: find it using its identifier. Example:

```
>>> from fusil.mockup import Project
>>> from fusil.process.attach import AttachProcessPID
>>> project = Project()
>>> pid =  42
>>> process = AttachProcessPID(project, pid)
```

AttachProcess

Similar to AttachProcessPID but find the process using its name instead of its identifier. Example:

```
>>> from fusil.mockup import Project
>>> from fusil.process.attach import AttachProcess
>>> project = Project()
>>> AttachProcess(project, "clamav")
<AttachProcess id=13, name='attach_process:clamav' is_active=False>
```

3.5.3 CPU load probe: CpuProbe

AttachProcess, AttachProcessPID and WatchProcess have 'cpu' attribute of type CpuLoad. If CPU load is bigger than maximum load during maximum duration, set score to 'max_score' (default: 100%). Default values:

```
>>> from fusil.mockup import Project
>>> project = Project()
>>> from fusil.process.cpu_probe import CpuProbe
>>> probe = CpuProbe(project, 'cpu')
>>> probe.max_load
```

```
0.75
>>> probe.max_duration
10.0
>>> probe.max_score
1.0
```

3.6 Mangle valid file

3.6.1 MangleFile

To fuzz file parser, you can use MangleFile agent. It takes one or multiple valid files on input and then injects errors to create invalid files. It can generate multiple files for each session.

Operations

- replace: replace a byte by a random byte
- bit: invert one bit value
- special_value: replace one or more bytes to write a special value, eg. four bytes: "0xFF 0xFF 0xFF 0xFF"
- insert_bytes: insert one or more random bytes
- delete_bytes: delete one or more bytes

MangleConfig

You can configure some options to help fuzzing using 'config' attribute of MangleFile. The value is an instance of MangleConfig class. Options:

- min_op: Minimum number of mangle operations (default: 1)
- max_op: Maximum number of mangle operations (default: 10)
- operations: List of operation name (default: ["replace", "bit", "special_value"])
- max_insert_bytes: Maximum number of insered bytes (default: 8)
- max_delete_bytes: Maximum number of deleted bytes (default: 8)
- change_size: Allow operations which change data size (default: False)

Truncate

You can limit maximum file size using 'max_size' attribute of MangleFile. The value is the maximum number of bytes read from input file.

3.6.2 AutoMangle

AutoMangle is an helper to MangleFile: it tries to find the best parameters to fuzz the target using session aggressivity. Option attributes:

- hard_min_op (default: 0): Minimum number of operations
- hard_max_op (default: 10000): Maximum number of operations

- fixed_size_factor (default: 1.0): ratio used to compute the number of operations depending on the file

3.6.3 IncrMangle

IncrMangle is the incremental mangle agent. Whereas AutoMangle regenerates all errors for each session, IncrMangle keeps errors between the sessions and add some new errors. Option attributes:

- operation_per_version: Maximum number of operations applied to new session
- max_version: Maximum version number for a file, if a file is older max_version, the operations are truncated to a random number of versions
- min_offset and max_offset (default None): Minimum and maximum file offset, both are optional (use None value)

Default values:

```
>>> from fusil.mockup import Project
>>> project = Project()
>>> from fusil.incr_mangle import IncrMangle
>>> mangle = IncrMangle(project, 'filename')
>>> mangle.operation_per_version
1
>>> mangle.max_version
25
>>> mangle.min_offset, mangle.max_offset
(None, None)
```

3.7 Fusil events

An event can only be sent once in a session step (eg. you can not send session_stop event twice).

3.7.1 Application

- application_done(): Fusil is done (exit)
- application_interrupt(): Ask Fusil application to stop
- application_error(message): Fatal Fusil error

3.7.2 Project

- project_start(): Creation of the project
- project_stop(): Ask to stop active project
- project_session_destroy(): Destroy session and create a new session if we are not done

3.7.3 Session

- session_start(): Creation of a new session
- session_stop(): Ask session to stop
- session_done(score): End of the active session, score is the final session score

- session_success(): The session is a success, sent at the end of the session
- session_rename('name'): Rename the session: all names are joined using '-' separator to rename the session directory

3.7.4 Aggressivity

- aggressivity_value(value): New aggressivity value with -1.0 <= value <= 1.0

3.7.5 Process

- process_create(agent): New process created
- process_stdout(agent, filename): Filename of the process stdout
- process_exit(agent, status): Process finished (exited or killed by a signal)
- process_pid(agent, pid): Attached process identifier

3.7.6 MangleFile

- mangle_filenames(filenames): Generated filenames

3.8 FileWatch

FileWatch is a probe used by WatchProcessStdout agent to watch a plain text file. It looks for patterns in new inserted lines.

3.8.1 Constructor

Fake objects used for the documentation:

```
>>> from fusil.mockup import Project, Logger
>>> logger = Logger()
>>> project = Project(logger)
```

Constructor syntax:

```
>>> from fusil.file_watch import FileWatch
>>> log = FileWatch(project, open('README', 'rb'), 'README')
```

Or just fromFilename() static method to only write the filename once:

```
>>> log = FileWatch.fromFilename(project, 'README')
```

If you watch server log, use start="end" to skip existing logs:

```
>>> log = FileWatch.fromFilename(project, 'README', start='end')
```

3.8.2 Ignore lines

Use ignoreRegex() method to ignore lines:

```
>>> from re import IGNORECASE
>>> log.ignoreRegex('^error: meaningless error')
>>> log.ignoreRegex('^ErrOR: another error', IGNORECASE)
```

You can add your own ignore handler:

```
>>> def ignoreNumber42(text):
...     try:
...         return int(text) == 42
...     except ValueError:
...         return False
...
>>> log.ignore.append(ignoreNumber42)
>>> ignoreEmptyLine = lambda line: len(line.strip()) == 0
>>> log.ignore.append(ignoreEmptyLine)
```

3.8.3 Words patterns

'words' patterns are case insensitive and only match 'word'. Example: "abc" pattern which match line "text: abc" but not "abcd".

FileWatch includes many text patterns in 'words' attribute:

```
>>> words = log.words.keys()
>>> from pprint import pprint
>>> words.sort(); pprint(words)
[u'allocate',
 u'assert',
 u'assertion',
 u'bug',
 u"can't",
 u'could not',
 u'critical',
 u'error',
 u'exception',
 u'failed',
 u'failure',
 u'fatal',
 u'glibc detected',
 u'invalid',
 u'memory',
 u'not allowed',
 u'not valid',
 u'oops',
 u'overflow',
 u'panic',
 u'permission',
 u'pointer',
 u'segfault',
 u'segmentation fault',
 u'too large',
 u'unknown',
 u'warning']
```

Get/set pattern score:

```
>>> print log.words['overflow']
0.4
>>> log.words['overflow'] = 0.5
```

3.8.4 Regex patterns

'regexs' attribute is a list of regex, use addRegex() to add a regex:

```
>>> log.addRegex('^Crash: ', 1.0)
>>> log.addRegex('null pointer$', 1.0, flags=IGNORECASE)
```

3.8.5 Patterns compilation

All patterns are compiled by createRegex() method on agent initialisation. It uses 'patterns' and 'words' attributes. Example:

```
>>> log = FileWatch.fromFilename(project, 'README')
>>> log.words = {'error': 0.5}
>>> log.addRegex('mplayer', 1.0)
>>> for pattern, score, match in log.compilePatterns():
...     print "%r, score %.1f%%, regex=%s" % (pattern, score, match)
...
'mplayer', score 1.0%, regex=...
'error', score 0.5%, regex=...
```

3.8.6 Cleanup line

You can register a function to cleanup lines:

```
>>> log.cleanup_func = lambda text: text[7:]
```

Test of the function:

```
>>> # Prepare test
>>> log.activate()
>>> log.init()
>>> log.show_not_matching = True; logger.show = True
>>> # Example of line
>>> log.processLine('PREFIX:Real line content')
Not matching line: 'Real line content'
>>> # Empty line
>>> log.processLine('PREFIX:')
>>> # Cleanup test
>>> log.show_not_matching = False; logger.show = False
```

3.8.7 Line number

'nb_line' contains the number of lines (without ignored lines) and 'total_line' the total number of lines. 'max_nb_line' attribute is the maximum number of total lines: (max, score). If 'nb_line' becomes bigger than max, score is incremented by score. Ignored lines are not included in 'nb_line'. Default value:

```
>>> log.max_nb_line
(100, 1.0)
```

To disable the maximum of line number, set 'max_nb_line' to None.

There is a similar option for the minimum number of line, but it's disabled by default (no minimum). Example to add -50% to the score if there is fewer than 10 lines of output:

```
>>> log.min_nb_line = (10, -0.5)
```

3.8.8 Pattern matching

For each text line, FileWatch calls processLine(). First it checks if the line matchs one ignore pattern. If not, it tries all patterns and uses the one with the biggest absolute score.

```
>>> log.init()
>>> log.processLine('This is an error')
>>> print log.score
0.5
```

Attributes:

- show_matching (default: False): use True to show matching lines (use ERROR log level instead of WARNING)
- show_not_matching (default: False): use True to show not matching lines (–debug option enable this option)
- log_not_matching (default: False): use True to log not matching lines. By default, lines are not logged because the output is already written to session "stdout" file.

3.9 Tools for C code manipulation

The fusil.c_tools module contains many tools to manipulation C code.

3.9.1 String manipulation

```
>>> from fusil.c_tools import quoteString, encodeUTF32
>>> quoteString('Hello World\n\0')
'"Hello World\\n\\0"'
>>> encodeUTF32("Hello")
'H\x00\x00\x00e\x00\x00\x00l\x00\x00\x00l\x00\x00\x00o\x00\x00\x00'
```

encodeUTF32() use host endian.

3.9.2 Generate C script

Hello World!

```
>>> from fusil.c_tools import CodeC
>>> from sys import stdout
>>> hello = CodeC()
>>> hello.includes.append('<stdio.h>')
>>> main = hello.addMain()
>>> main.callFunction('printf', [quoteString("Hello World\n")])
```

```
>>> hello.useStream(stdout)
>>> hello.writeCode()
#include <stdio.h>

int main() {
    printf(
        "Hello World\n"
    );

    return 0;
}
```

FunctionC

addMain() is an helper to create main() function, but you can write your own functions with addFunction() method:

```
>>> from fusil.c_tools import FunctionC
>>> testcode = CodeC()
>>> test = testcode.addFunction( FunctionC('test', type='int') )
>>> test.variables.append('int x')
>>> test.add('x = 1+1')
>>> test.add('return x')
>>> testcode.useStream(stdout)
>>> testcode.writeCode()
int test() {
    int x;

    x = 1+1;
    return x;
}
```

You can get the function with:

```
>>> hello['main']
<FunctionC "int main()">
>>> testcode['test']
<FunctionC "int test()">
```

3.9.3 Write to a file

To write a code to a file, use writeIntoFile() method:

```
>>> from pprint import pprint
>>> hello.writeIntoFile('hello.c')
>>> pprint(open('hello.c').readlines())
['#include <stdio.h>\n',
 '\n',
 'int main() {\n',
 '    printf(\n',
 '        "Hello World\\n"\n',
 '    );\n',
 '\n',
 '    return 0;\n',
 '}\n',
 '\n']
```

3.9.4 Compile the code

To compile the code, use compile() method:

```
>>> from os import system, WEXITSTATUS, unlink
>>> from fusil.mockup import Logger
>>> logger = Logger()
>>> hello = CodeC()
>>> main = hello.addMain(footer='return 2*3*7;')
>>> hello.compile(logger, 'hello.c', 'hello')
>>> WEXITSTATUS(system('./hello'))
42
>>> unlink('hello.c')
>>> unlink('hello')
```

3.9.5 Misc attributes

You can customize write() output:

- 'indent' is the indententation string (default: 4 spaces)
- 'eol' is the end of line string (default: "n")

Set gnu_source to True to get:

```
#define _GNU_SOURCE
```

3.9.6 FuzzyFunctionC

Ok, let's play with fuzzing! FuzzFunctionC has methods to generate values:

```
>>> from fusil.c_tools import FuzzyFunctionC
>>> fuzzy = CodeC()
>>> main = fuzzy.addFunction(FuzzyFunctionC('main', type='int'))
```

Methods to generate data:

- createInt32()
- createInt()
- createString()
- createRandomBytes()

Example:

```
>>> main.add('return %s' % main.createInt())
```

3.10 Scoring system

3.10.1 Problematic

Guess a fuzzing session success or failure is a complex task. We can use different parameters like process exit code, stdout, session duration, etc. But for each project, the meaning of the values may change. For some projects, session timeout is a success whereas you may ignore timeout for other projects.

3.10.2 Fusil probes

That's why Fusil use a scoring system. You can use multiple "probes" and each probe compute its own score. Session score is the sum of all scores. A probe score is a value between -1.0 and 1.0 where:

- 1.0 is a success (eg. program crash)
- 0.0 means "nothing special"
- -1.0 means that the application just rejects your input, you may try next session with less noise

Each probe score is normalized in -1.0..1.0 interval. Session score is not normalized, 130% value is allowed.

You can also set a probe "weight" ('score_weight' attribute, default value: 1.0) to change its importance in session score (see example above).

3.10.3 Example

Let's take a project with 4 probes:

- WatchProcess(A)
- WatchProcess(B)
- TimeWatch: weight=0.5 (less important)
- FileWatch: weight=2 (more important)

At the end of the session, the scores are:

- WatchProcess(A): score=0.25
- WatchProcess(B): score=None (no score)
- TimeWatch: score=-0.10
- FileWatch: score=0.15

Session score is:

```
0.25 + -0.10 * 0.5 + 0.15 * 2 = 0.50
```

Since minimum score for a success is 'project.success_score' (default: 50%), we can say that the session is a success!

3.11 Time managment

3.11.1 Session timeout

You can set maximum session duration using Project.session_timeout option (value in second). If session reachs the timeout, it's stopped.

3.11.2 Logging

Messages are written with a timestamp.

3.11.3 TimeWatch

To compute session score, you can use TimeWatch probe. It has two parameters:

- too_fast: mininum duration of a valid session, faster session would have 'too_fast_score' score (default: -100%)
- too_long: maximum duration of a valid session, slower session would have 'too_long_score' score (default: 100%)

3.11.4 Process

A process have a default timeout set to 10 seconds. If the timeout is reached, the process is directly killed using SIGKILL signal and WatchProcess will use its 'timeout_score' attribute as score (default: 100%).

CHAPTER 4

Pages

4.1 Crash list

See also *tested programs*.

4.1.1 CVE found using Fusil

- 2007-05-22: CVE-2007-2754
  - Security hole in FreeType
  - Freetype TT_Load_Simple_Glyph() TTF File Integer Overflow Vulnerability
- 2007-05-11: CVE-2007-2650
  - ClamAV OLE2 Parser Denial of Service
  - ClamAV OLE2 Parser Large Property Size Processing Denial of Service Vulnerability
  - FIXED in version 0.90.3
- 2007-05-10: CVE-2007-2645
  - Libexif "exif_data_load_data_entry()" EXIF Image Handling Integer Overflow Vulnerability

4.1.2 ImageMagick

- ImageMagick: image manipulation on the command line
  - Report bug in the forum
- Version: last version (2007-05-07)
  - FIXED: Crash in EXIF parser with invalid IFD count
  - WON'T FIX: Bug report in TGA and XCF files

4.1.3 rpm

- rpm: package manipulation of Redhat, Fedora Core, Mandriva and other Linux distributions
- Version: last version (2007-05-10)
  - FIXED: Bug report #239557: patch to fix the bug

4.1.4 glibc

- glibc is library used by all programs on a computer :-)
  - Bugtracker
- Version: last version (2007-04-30)
  - FIXED: vfprintf() bug – bug fixed in version 2.5.1

4.1.5 FreeType2

- FreeType2: Font library to render text supporting many font file format (eg. TTF and OTF)
- Version: last version (2007-04-28)
  - FIXED: cmap bug
  - FIXED: Negative number of points bug
- 2008-02-18: bug #22356: TrueType: crash in Ins_IUP() (closed 2h later)

4.1.6 ClamAV

- ClamAV: open source antivirus
  - Bugzilla
- Version: 0.90.2
  - FIXED: Loop in FAT of OLE2 document (fixed in 0.90.3)
  - FIXED: OLE2: Allocate too much memory with invalid file (fixed in 0.91)
  - FIXED: bitset_realloc() is not atomic (fixed in 0.91)
  - FIXED: OLE2: Long (slow) loop in ole2_walk_property_tree() with huge prop_index value (fixed in 0.91)

4.1.7 poppler (PDF)

- Poppler: library to diused by Kpdf and Evince
  - Bugzilla
- Version: 0.5.4
  - FIXED: Crash on fuzzed PDF at Parser.cc:192
- Version trunk (2007-05-11):
  - Crash on fuzzed PDF: recursive call of Parser::getObj()
- 2008-02-18 (poppler 0.6.0)
  - Bug 14549: Crash in Lexer::getObj() if xref is NULL (fixed)

4.1.8 libexif

- libexif is a library to read/write EXIF data
- libexif is used by nautilus, gwenview, gimp, exiftran and a lot of other programs!
- Version: 0.6.13
  - FIXED: Serious security bug in exif_data_load_data_entry() => patch
  - Crash in exif_entry_fix() when e->data is NULL

4.1.9 binutils (bfd)

- binutils are as, ld, nm, libbfd, etc.
  - Bugtracker
- Version 2.17 and CVS HEAD
  - FIXED: bfd_elf_string_from_elf_section() doesn't check shindex value => patch

4.1.10 Gimp

- Gimp 2.2.13
- FIXED: missing input validation in several file plug-ins (fixed in Gimp 2.2.16)

4.1.11 Tremor (Ogg/Vorbis)

- Tremor trunk (2007-11-01)
- Bug #1254

4.1.12 Gstreamer

- Gstreamer 0.10 (gst-plugins-good 0.10.6)
- Bug 510592 – Race condition in WAVE parser (fixed the 2008-01-19)
- Bug 510982 – gst_tag_demux_trim_buffer: invalid return value
- Bug 525665 – Crash on Ogg/Vorbis with chain=NULL (fixed)

4.1.13 PHP

- array_slice(&$offset, $offset) crash PHP 5.2.5: FIXED
- count_chars() crashes if both arguments are the same reference: crash PHP 5.2.6
- levenshtein() crashs with invalid arguments: crash PHP 5.2.6 : FIXED

4.1.14 Python

See dedicated page: [[Python]].

4.1.15 xterm

- xterm crashs with long PATH environment path with no ":" character (fixed in xterm patch 236)

4.1.16 vim

```
$ VIMRUNTIME=$(python -c 'print "a"*10000') vim
Erreur de segmentation
$ VIM=$(python -c 'print "a"*10000') vim
Erreur de segmentation
```

4.1.17 dpkg-query (apt/dpkg)

```
$ COLUMNS=10000000 dpkg-query -l
Souhait=inconnU/Installé/suppRimé/Purgé/H=à garder
| État=Non/Installé/fichier-Config/dépaqUeté/échec-conFig/H=semi-installé
|/ Err?=(aucune)/H=à garder/besoin Réinstallation/X=les deux (État,Err: majuscule=mauvais)
Erreur de segmentation
```

It was a bug in vfprintf() from glic (see above).

4.2 Tested programs

See also the *list of crashes*.

4.2.1 Bug reported

- ClamAV
- FreeType2
- gettext: LANGUAGE='../../../../../tmp' with non-suid programs
- glibc
- ImageMagick
- libpoppler
- libexif
- nm
- rpm

4.2.2 Bug not yet reported

- **file**
  - crash with invalid magic.mgc file, you can be specified using -m
- **mplayer**
  - `HOME=''` make mplayer crash and it never leaves

  - command line: `-quiet -vo null -ao null -endpos 2`

- **Flash** browser plugin
  - many crash

4.2.3 gettext

- gettext: library to translate (localize) string (i18n/l10n things)
  - gettext on savannah
- It's possible to use your own .mo file with this little hack:

```
mkdir /tmp/LC_MESSAGES
cp hello.mo /tmp/LC_MESSAGES/
LANGUAGE='../../../../../../../../tmp' ./hello
Erreur de segmentation
```

- Found crash with fuzzing near intl/dcigettext.c:934 but //Bruno Haible// (author of gettext) will not fix them (add more verifications) because it « would not serve the purpose of a maximally efficient lookup of translations »

4.2.4 gstreamer

```
cd gst-plugins-base
./configure --prefix=/opt/gstreamer CFLAGS="-O0 -ggdb" PKG_CONFIG_PATH=/opt/gstreamer/lib/pkgconfig/
```

4.2.5 python

```
./configure --prefix=/opt/pythonsvn --with-pydebug && make && make install
```

With pydebug, use "export PYTHONTHREADDEBUG=1" to trace threading operations.

4.2.6 php

```
sudo apt-get build-dep php5
cvs -z9 -d :pserver:cvsread@cvs.php.net:/repository checkout -r PHP_5_3 php5
cd php5
./buildconf
CFLAGS="-O0 -ggdb" ./configure --enable-debug --prefix=/opt/php && make && make install
```

and then use/copy the program sapi/cli/php.

Other configure options:

```
--enable-mbstring \
--enable-ftp --enable-calendar --enable-exif --enable-intl \
--enable-soap  --enable-sockets --enable-zip
```

Notes:

- php_error_cb() displays a PHP error like the "Fatal errors"
- Real function name is "**zif_**" + name (eg. zif_count_chars for count_chars)
- Interesting files: main/php.h, main/php_config.h, Zend/zend.h

4.3 Linux process limits: setrlimit

4.3.1 Linux limits

- RLIMIT_AS: maximum size of the process's virtual memory in bytes.
- RLIMIT_CORE: Maximum size of core file.
- RLIMIT_CPU: CPU time limit in seconds.
- RLIMIT_FSIZE: Maximum size of files that the process may create.
- RLIMIT_LOCKS: Combined number of flock() locks and fcntl() leases
- RLIMIT_MEMLOCK: Maximum number of bytes of memory that may be locked into RAM.
- RLIMIT_MSGQUEUE: Limit on the number of bytes that can be allocated for POSIX message queues
- RLIMIT_NICE: Ceiling to which the process's nice value can be raised
- RLIMIT_NOFILE: Value one greater than the maximum file descriptor number that can be opened by this process.
- RLIMIT_NPROC: The maximum number of processes that can be created
- RLIMIT_RTPRIO: Ceiling on the real-time priority
- RLIMIT_SIGPENDING: Limit on the number of signals that may be queued
- RLIMIT_STACK: Maximum size of the process stack in bytes

4.3.2 Not implemented in Linux

- RLIMIT_DATA
- RLIMIT_RSS

4.4 Python

fusil-python is the most active fuzzer. I'm using it on Python since april 2007 (3 years ago). I tried fusil-python on CPython 2.5, 2.6, 2.7 (alpha), 3.0, 3.1 and 3.2 (alpha), PyPy 1.1, 1.2 (alpha) with the JIT, and Unladen Swallow.

4.4.1 CPython bugs found by fusil-python (not fixed yet)

- 2010-01-11: audioop: check that length is a multiple of the size - Module: audioop, cause: insufficient input validations, consequence: write to uninitialized memory
- 2010-01-10: _sqlite3: Block *all* operations on a closed Connection object - Module: sqlite3, cause: operations on a closed connection, consequence: ?
- 2009-12-19: Fatal error on thread creation in low memory condition - Module: thread, cause: insufficient error handling, consequence: quit python (fatal error)
- 2009-12-13: _lsprof (cProfile): Profiler.clear() keeps references to detroyed nodes - Module: cProfile, cause: ?, consequence: read freed memory (may raise a segmentation fault)
- 2009-12-11: _sqlite3 doesn't catch PyDict_SetItem error - Module: sqlite3, cause: missing error handling, consequence: ?

4.4.2 CPython bugs found by fusil-python (fixed)

- 2010-01-14: fix output string length for binascii.b2a_uu() (commit) - Module: binascii, cause: error in output buffer size computation, consequence: write to uninitialized memory
- 2009-12-08: crash in str.rfind() with an invalid start value (commit 1, commit 2, not written by me) - Module: ~__builtin~__ (str type), cause: insufficient input validations, consequence: read uninitialized memory
- 2009-12-08: cPickle: stack underflow in load_pop() (commit) - Module: cPickle, cause: ?, consequence: read uninitialized memory
- 2009-09-24: _json crash on scanner/encoder initialization error (commit) - Module: json, cause: insufficient error handling, consequence: ?
- 2009-07-22: bytearray.translate(): error in error handling (commit) - Module: ~__builtin~__ (bytearray type), cause: insufficient error handling, consequence: read uninitialized memory
- 2009-05-05: _json: _convertPyInt_AsSsize_t() never raise any error (commit) - Module: json, cause: insufficient input validation, consequence: read uninitialiazed memory
- 2009-01-30: Invalid UTF-8 ("%s") length in PyUnicode_FromFormatV() (commit) - Module: ~__builtin~__ (unicode type), cause: ?, consequence: write into uninitialiazed memory
- 2008-09-30: fix security issue 2: imageop's poor validation of arguments could result in segfaults (commit) - Module: imageop, cause: insufficient input validation, consequence: read and write from/into uninitialiazed memory
- 2008-09-26: Check PyInt_AsSsize_t/PyLong_AsSsize_t error (commit 1, commit 2) - Module: struct and io (BytesIO and StringIO), cause: insufficient error handling, consequence: ?
- 2008-09-25: bytearray().count() (commit) - Module: ~__builtin~__ (bytearray type), cause: insufficient input validations, consequence: read uninitialized memory
- 2008-09-24: _hotshot: invalid error control in logreader() (commit) - Module: hotshot, cause: invalid error handling, consequence: unexpected exception during garbage collection (fatal error)
- 2008-09-24: _lsprof: clear() should call flush_unmatched() (invalid, unable to reproduce the crash) - Module: cProfile, cause: ?, consequence: ?
- 2008-09-16: errors on _bsddb creation and dealloc (commit) - Module: bsddb, cause: non initialized memory, consequence: read uninitialized memory - Module: bsddb, cause: invalid error handling, consequence: unexpected exception during garbage collection (fatal error)
- 2008-09-16: _tkinter._flatten() doesn't check PySequence_Size() error code (commit) - Module: Tkinter, cause: missing error handling, consequence: unexpected exception during garbage collection (fatal error)
- 2008-08-22: Add more checks to testcapi (commit 1, commit 2) - Module: testcapi, cause: insufficient input validation, consequence: unexpected exception during garbage collection (fatal error)
- 2008-08-21: Remove module level functions in _tkinter that depend on TkappObject (commit for py3k, commit for trunk)
- Module: Tkinter, cause: ?, consequence: write into non initialized memory
- 2008-08-21: invalid result value of _weakref.~__init~__() (commit)
- Module: weakref, cause: bug in error handler, consequence: unexpected exception during garbage collection (fatal error)
- 2008-08-21: use string_print() in gdb (commit). Not a bug, but it helps development.
- 2008-08-20: _json: fix raise_errmsg(), py_encode_basestring_ascii() and linecol() (commit 1, commit 2) - Module: json, cause: missing error handling, consequence: read uninitialized memory

- 2008-08-20: possible deadlock in python IO implementation (fixed by the new io library) - Module: io, cause: profiling callback using writing to stdout while a function is also writing to stdout, consequence: dead lock
- 2008-08-20: Invalid exception context (commit 1, commit 2)
- 2008-07-08: bugs in scanstring_str() and scanstring_unicode() of _json module (commit) - Module: json, cause: insufficient input validation, consequence: read uninitialized memory
- 2008-07-08: _multiprocessing.Connection() doesn't check handle (commit) - Module: multiprocessing, cause: insufficient input validation, consequence: write into uninitialized memory
- 2008-07-07: dlopen() error with no error message from dlerror() (commit) - Module: dl and ctypes, cause: missing error handling, consequence: read uninitialized memory (NULL pointer)
- 2008-07-07: bugs in _sqlite module (commit) - Module: sqlite3, cause: missing error handling, consequence: read uninitialized memory (NULL pointer)
- 2008-07-07: block operation on closed socket/pipe for multiprocessing (commit)
- 2008-07-07: missing lock release in BZ2File_iternext() (commit) - Module: bz2, cause: invalid error handling, consequence: dead lock
- 2008-07-06: invalid check of _bsddb creation failure (commit) - Module: bsddb, cause: invalid error handling, consequence: read uninitialized memory
- 2008-07-06: audioop.findmax() crashs with negative length (commit) - Module: audioop, cause: insufficient input validation, consequence: read uninitialized memory
- 2008-07-06: Use Py_XDECREF() instead of Py_DECREF() in MultibyteCodec and MultibyteStreamReader (commit) - Module: multibytecodec, cause: invalid error handling, consequence: read uninitialized memory (NULL pointer)
- 2008-07-06: invalid call to PyMem_Free() in fileio_init() (commit) - Module: io (FileIO class), cause: invalid error handling, consequence: ?
- 2008-07-06: invalid ref count on locale.strcoll() error (commit) - Module: locale, cause: invalid error handling, consequence: invalid reference count
- 2008-07-06: segfault on gettext(None) (commit) - Module: locale, cause: insufficient input validation, consequence: read uninitialized memory (NULL pointer)
- 2008-07-06: DoS when lo is negative in bisect.insort_right() / _left() (commit) - Module: bisect, cause: insufficient input validation, consequence: unlimited loop (denial of service)
- 2007-04-10: Segfaults on memory error (commit) - Module: ~__builtin~__ (Exception, long and object types), cause: missing error handling, consequence: read uninitialized memory (NULL pointer)

4.4.3 PyPy and Unladen Swallow

PyPy and Unladen Swallow were also tested using Fusil :

- Finding Bugs in PyPy with a Fuzzer
- Fuzzing on #@make_function() and #@exec() (Unladen Swallow) : FIXED

4.5 Ideas

4.5.1 Ideas of new projects

- Use process signals as input vector

- Regular expression
- xfs (X11 Font Server), Fusil already found [[CrashList#FreeType2|bugs in FreeType2 library]]
- Python modules: pickle, imageop, tarfile, locale
- Rewrite internet browser fuzzer for new fuzzing implementation
- Start to fuzz Windows?
- Network proxy fuzzer
- Fuzzing linux kernel: bugzilla, MOKB, ioctl, IEEE802.11, ALSA, ptrace, sysfs, netfilter and netlink, ipv6, coredump, iso9660, /proc, ext3, ...

4.5.2 Reuse existing libraries and projects

- http://www.nongnu.org/failmalloc/

4.5.3 Ideas to crash programs

- Don't create stdin, stdout or stderr to check if first open file gets file descriptor #0
- Continue to analyze gettext :-)
- write C library to inject errors in libc calls (eg. malloc() failure)
  - Generate [http://michael-prokop.at/blog/2007/06/12/error-handling-enospc/ ENOSPC] errors?
  - file: open(), close(), read(), write()
  - directory: opendir(), chdir()
  - memory: malloc(), realloc(), calloc()
  - network: socket(), setsockopt()
  - time: time(), gettimeofday() http://software.inl.fr/trac/trac.cgi/wiki/Macfly
- network socket proxy fuzzer

Signals

Send signals like SIGINT, SIGTERM, SIGSTOP, SIGUSR1, SIGUSR2.

Old bugs:

- broken pipe (SIGPIPE) https://bugzilla.mindrot.org/show_bug.cgi?id=85
- libc deadlock http://sourceware.org/bugzilla/show_bug.cgi?id=838
- openssh pre-authentification denial of service https://bugzilla.mindrot.org/show_bug.cgi?id=1129

4.5.4 Score

- Code coverage:
  - gcov: http://gcc.gnu.org/onlinedocs/gcc/Gcov.html
  - Valgrind: http://www.valgrind.org/ Valgrind
  - DynamoRio: http://www.cag.lcs.mit.edu/dynamorio/

- Check invalid use of memory using Valgrind (or any memory checker tool)
- increment score if one of these function is called:
  - fgets(), memcpy(), strcpy()
  - input comes from user (?)
  - bytes read by the program
  - memory usage

4.6 Fusil TODO list

- replay.py is unable to open a file as stdin, required by fusil-gimp
- Factorize code responsible to rename the session on process exit (share code between Debugger and CreateProcess)
- Protect the terminal using setsid(), setpgrp(), or setpgid()
- Use initgroups() in CreateProcess?
- Remove the class WatchProcess: move code to CreateProcess to avoid duplicate events (agent score) and duplicate code

4.7 Links

4.7.1 Fuzzer using Fusil

Parrot fuzzer:

- Download parrot-fuzzer
- Link to Trac

4.7.2 References to Fusil

Articles:

- Fusil: a Python fuzzing library (LWN)

Registration:

- ohloh.net code metrics
- Freshmeat page

News:

- Fusil the fuzzer version 1.0beta3 (2008-09-03) on the Full Disclosure mailing list
- Fusil the fuzzer version 1.0beta3 (2008-09-03) on the pen-test mailing list
- Sortie de Fusil le fuzzer en version 1.0beta3 (2008-09-03)
- Full-disclosure: Fusil the fuzzer version 0.9 released (8 July 2008)
- Fusillez vos applications avec Fusil 0.6 (2007-12-19)

- Blog entry on security-protocols.com (2007-11-27)
- linuxfr.org journal (2007-11-27)
- Fusil Fuzzer 0.7 - Fuzzing Functions in Python (darknet)

4.7.3 Fuzzer list (non tested)

Last changed in 2007:

- ioctlizer 0.1 (Windows)
- Bunny, last release: 2007-11-06, author: Michal Zalewski, written in C
- Flayer, last release: 2007-08-09, read also paper about Flayer, authors: Will Drewry and Tavis Ormandy
- zzuf: multi-purpose fuzzer, last release: 2007-11-03, author: Sam Hocevar
- Sulley, last change: 2007-08-02, author: Pedram Amini and Aaron Portnoy
- pff (PHP fuzzer), author: calcite, last change: 2007-07-03, rewritten in Fusil: [[browser:trunk/project/php.py|project/php.py]]
- mangle.c, author: Ilja van Sprundel, rewritten in Fusil: [[browser:trunk/fusil/mangle.py|fusil/mangle.py]]
- sysfuzz.c, rewritten in Fusil: [[browser:trunk/project/linux_syscall.py|project/linux_syscall.py]]
- ‘ Evolutionary Fuzzing System (EFS) <http://www.vdalabs.com/tools/efs_gpf.html>‘_, last change: 2007-04-13, written in Python, author: Pedram Amini and Jared !DeMott
- Peach, author: Michael Eddington, written in Python, last change: 2007-07-30
- petardfs, FUSE filesystem, last change: 2007-09-19, author: Ben Martin
- Scapy, author: Philippe Biondi, written in Python
- Schemer, last change: 2007, written in C#

Last changed in 2006:

- Autodafe, last change: 2006-08-05, written in C
- fsfuzz, last release: 2006-10-24, author: L.M.H, written in bash (and C for mangle.c)
- dfuz, last release: 2006-06-18, author: Diego Bauche, written in C
- Mistress, last release: 2006-03-02, author: posidron, written in Python
- FileFuzz from iDefense (Windows), last change: 2006-11-15, written in C#

Last changed in 2005:

- PROTOS
- antiparser, last release: 2005-08-17, written in Python

Last changed in 2004:

- Scatch, last change: 2004, written in Python
- SPIKE, last change: 2004, author: Dave Aitel and Dug Song, written in C
- mangleme, HTML manglizer, author: Michal Zalewski, written in C (CGI), last change: 2004-10-18
- ISIC (IP Stack Integrity Checker), last release: 2004-11-11, author: Shu Xiao

Older projects:

- lxapi, last chage: 2003, author: : endee, written in Python

- Advanced Fuzz Experiment (afx), author: Michal Zalewski, last change: 2001-04-21
- fuzz.c, first version: 1989, last changes: 2006, author: Barton Miller, written in C

4.7.4 List of fuzzer list

- http://www.hacksafe.com.au/blog/2006/08/21/fuzz-testing-tools-and-techniques/
- Fuzzing Tools list on secwiki
- Fuzzing on owasp.org

4.7.5 Other fuzzer links

- Automated Whitebox Fuzz Testing (May 2007)

4.7.6 Vulnerabilities publication

- Secunia - full-disclosure (@lists.grok.org.uk, mailing list)
- FrSIRT
- heise Security UK
- CERT
- CVE (Common Vulnerabilities and Exposures)
- SecurityFocus - BugTraq (@securityfocus.com, full disclosure, mailing list)
- National (USA) Vulerability Database
- SecurityTracker
- OSVDB (Open Source Vulnerability Database)
- netVigilance

4.7.7 Vulnerability labs

- Xforce ISS
- Rapid7
- iDefense (labs)
- GamaSEC (GamaLAB)
- Matousec
- Chris Evans (Software security hole)
- Scanit
- Zero Day Initiative (3COM)

4.7.8 Free software audit

- Open Source Quality Project (OSQ)
- Open source vulnerability database (OSVDB)
- OpenBSD Security Audit
- Gentoo Linux Security Project
- Debian Security Audit - Security Audit Bugs
- Ubuntu Security Team - Bugs related to Ubuntu Security Team
- Mandriva Security (no audit team?)
- Fedora Security
- Slackware Security

4.7.9 Sell vulnerabilities

- Zero Day Initiative
- iDefense Vulnerability Contributor Program
- WabiSabiLabi
- 1000€ for a remote exploit in Dovecot

4.8 Notes

Recompile Debian package:

```
apt-get build-dep program
cd /tmp
apt-get source program
cd program(...)
# make sure that noopt is supported
# if not, set CFLAGS to -O0 -ggdb
DEB_BUILD_OPTIONS="nostrip noopt" dpkg-buildpackage -rfakeroot -us -uc
```

CHAPTER 5

Presentations

- (en) Fusil : FOSDEM 2009, Bruxelles (Belgium)
- (fr) Assurance qualité avec Fusil le fuzzer : RMLL 2008, Mont de Marsan (France)
- (fr) Rump session : SSTIC 2007, Rennes (France)

CHAPTER 6

Articles

- (en) Fusil: a Python fuzzing library by By Jake Edge (March 11, 2009) on LWN.net
- (fr) Pratiquer le fuzzing avec Fusil : MISC magazine #39 (September/October 2008), pages 38-41
- (fr) Comment réaliser un fuzzer : MISC magazine #36, March/April 2008, pages 68-73

CHAPTER 7

News

- 2011-02-16: Release of Fusil 1.4, read the ChangeLog
- 2010-01-09: Release of Fusil 1.3.2, read the ChangeLog
- 2009-11-09: Release of Fusil 1.3.1
- 2009-09-18: Release of Fusil 1.3
- 2009-08-05: The website moved to a new server (bitbucket), the source code is now stored in a Mercurial repository instead of a Subversion repository
- 2009-02-04: Release of Fusil 1.2
- 2008-10-22: Release of Fusil 1.1
- 2008-09-13: Release of Fusil 1.0 final
  - Create zzuf and vlc fuzzers
  - Replace replay.sh and gdb.sh by replay.py which has many more options (eg. –valgrind or –user)
  - Basic Windows support