netcrawl Documentation

Release 0.4.0-beta

Wyko ter Haar

Mar 29, 2017

Contents

CHAPTER 1

Network Information Gathering Made Easy

Netcrawl is a tool designed to discover and poll one or more devices, inventory them, and then provide useful data on the processed devices.

This package is still in development.

Indices

- genindex
- modindex
- search

Features

- **Switchport Tracing**: Discover which devices and interfaces have seen a particular MAC
- **Wireless Audit**: Discovers likely matches for rogue wireless devices among physically connected devices on a subnet
- **MAC Audit**: Discover potential unauthorized switches on your network
- SSH and Telnet connections to network devices
- Automatically backs up device configurations
- Stores a neighbor database to find layer two connection mappings
- Auto-detect system type of newly discovered devices
- Works with Nmap to allow for discovery of both neighboring and seperated devices
- Securely stores credentials using keyring and cryptography
- Can use multiple credentials in case the first fails

- Stores device inventory using a PostgreSQL database
- Offers a single device scan to quickly get data on one device
- Concurrently runs multiple subprocesses to quickly scan devices
- Multiple `netcrawl` top-level processes can run concurrently to scan different network segments (do not use `-c` while doing this), or to run an Nmap scan and inventory hosts as they are discovered.

Example

Scan one host with no logging output

```
C:\netcrawl>run.py -sS -t 10.1.120.1 -v0

Device Name:       my-device-dist-1
Unique Name:       MY-DEVICE-DIST-1_EC032
Management IP:     10.1.120.1
First Serial:      Name: [Switch System], Desc: [WS-C4500X-32], Serialnum:␣
↪[JAE14350G30]
Serial Count:      28
Dynamic MAC Count: 920
Interface Count:   88
Neighbor Count:    22
Config Size:       26573

+----------------------------+-------------------------+-----------------------+---------
↪----+
| Neighbor Name              | Source Interface        | Platform              | IP␣
↪Address    |
+----------------------------+-------------------------+-----------------------+---------
↪----+
| DVCOPS-MIS-1               | TenGigabitEthernet1/1   | cisco WS-C3750-48P    | 10.1.
↪220.11 |
| DVCOPS-MIS-2               | TenGigabitEthernet1/2   | cisco WS-C3750-48P    | 10.1.
↪220.10 |
| DVCOPS-sceast-sc01         | TenGigabitEthernet1/3   | cisco WS-C3850-48P    | 10.1.
↪139.12 |
| DVCOPS-sccent-sc01         | TenGigabitEthernet1/4   | cisco WS-C3850-48P    | 10.1.
↪139.11 |
| DVCOPS-dcgsc-sc01          | TenGigabitEthernet1/16  | cisco WS-C3850-48P    | 10.1.
↪139.26 |
| DVCOPS-wlcprm-vd01         | TenGigabitEthernet1/17  | AIR-CT5520-K9         | 10.1.
↪139.51 |
+----------------------------+-------------------------+-----------------------+---------
↪----+
```

Locate a device on the network

```
C:\netcrawl>locate_mac.py 00FEC89232B0

MAC:  00FEC89232B0
Manufacturer:  Cisco ,  Cisco Systems, Inc
+------------------------+---------------------+------------------------+
| Device Name            | Interface           | CDP Neighbors          |
+------------------------+---------------------+------------------------+
| DVCOPSDS01             | Ethernet2/24        | DVCOPSMGT1             |
| DVCOPSMGT1             | GigabitEthernet0/23 | None                   |
| DVCOPS-mgmt-sd01       | FastEthernet1/0/39  | DVCOPSDS01             |
+------------------------+---------------------+------------------------+
```

Built With

- Netmiko - SSH and Telnet connection manager
- Manuf - OUI lookup

Authors

- **Wyko ter Haar** - *Initial work* - Wyko

Installation

These instructions will help install Netcrawl in your environment.

Netcrawl

```
pip install -U netcrawl
```

PostgreSQL

1. Download and install PostgreSQL
2. Set up the `main` and `inventory` databases. If these are not created netcrawl will attempt to create them automatically.

Credentials

Add device and database credentials to the credential vault using `netcrawl -m`

Nmap

Installing this will permit you to be able to use the -sN function.

- Nmap - Manually download and install
- python-nmap - Python insterface for Nmap

Testing

```
pip install Faker pytest
```

Netcrawl Usage

Recursive Scan

`netcrawl.core.`**`recursive_scan`**`(`***kwargs*`)`

Starts a **Recursive Scan** (-sR) run. This is the main scanning method for netcrawl.

1. If a `target` kwarg is given, add that seed device to the list of pending deivces, even if it was already visited.
2. Create workers (subprocesses) to perform the scanning work, up to 16 per CPU core, or up to the `processes` kwarg per core if that kwarg was given.
3. Query the Pending table in the Main database for pending devices.
4. Autodetect the Netmiko platform for each device if needed.
5. Inventory the device using *`netcrawl.devices.base.NetworkDevice.process_device()`*

6.Add each discovered device to the Inventory database

Keyword Arguments

- **`skip_named_duplicates`** (*`bool`*) – If True, this will cause netcrawl to skip neighbors which have the same hostname as a device that was previously visited.

  Note: While this can potentially save a lot of time when scanning devices, if multiple different devices share the same hostname, they will not be scanned!

- **`target`** (*`str`*) – The IP address of a seed device to add to the pending devices database
- **`netmiko_platform`** (*`str`*) – The Netmiko platform of the `target` device, if one was given.
- **`processes`** (*`int`*) – The number of worker processes to create, multiplied by the CPU count

Note: If there are any remaining keyword arguments in `**kwargs`, they will be passed to *`netcrawl.io_sql.main_db`* and *`netcrawl.io_sql.device_db`*

Single Scan

`netcrawl.core.`**`single_scan`**`(`*target*, *netmiko_platform='unknown'*`)`

Starts a **Single Scan** (-sS) run. This scan polls a single device and presents information about the device to the console. Useful for testing a connection, as well as getting a quick overview of the target.

Keyword Arguments

- **`target`** (*`str`*) – The network address of the device to scan
- **`netmiko_platform`** (*`str`*) – The Netmiko platform of the `target` device. If one is not given, it will attempt to autodetect the device type.

Nmap Scan

`netcrawl.core.`**`nmap_scan`**`(`*target*, ***kwargs*`)`

Ping each host in a given range one at a time. When a live host is found, add it to the pending hosts database.

Parameters **`target`** (*`str`*) – An Nmap compatible target specifier as outlined in the Nmap documentation

Keyword Arguments **`**kwargs`** – Arguments to pass to *`netcrawl.io_sql.main_db`*

Dependencies

netmiko

- Provides core SSH and Telnet connection functionality
- **Minimum Version Required**: 1.3.0

psycopg2

- A package to interact with the PostgreSQL backend

cryptography

- Encrypts the database and device logon credentials

keyring

- Stores the encryption key

Note: Linux users may have to install keyring with added consideration. Please see Running keyring on Linux. To ease this you can install the keyrings.alt package, but that has possible security implications. Use at your discretion.

- python-nmap
- netaddr
- prettytable

netcrawl package

Subpackages

netcrawl.credentials package

Submodules

netcrawl.credentials.manage module

Created on Mar 11, 2017

@author: Wyko

`netcrawl.credentials.manage.`**`add_device_cred`**(*_cred*)

`netcrawl.credentials.manage.`**`delete_device_cred`**(*_cred=None*, *index=None*)

`netcrawl.credentials.manage.`**`get_database_cred`**()

`netcrawl.credentials.manage.`**`get_device_creds`**()

`netcrawl.credentials.manage.`**`list_creds`**()
Lists all credentials in secure form

`netcrawl.credentials.manage.`**`write_database_cred`**(*_cred*)

netcrawl.credentials.menu module

class `netcrawl.credentials.menu.`**`DeleteDeviceCred`**(*completekey='tab'*, *stdin=None*, *stdout=None*)
Bases: *`netcrawl.credentials.menu.UserPrompt`*

`do_1`(*args*)
Delete by index

`do_2`(*args*)
Delete exact credential

`intro` = **'\nDelete a credential:\n1) By index\n2) Enter exact username and password\nR) Return to main menu \nQ) Exi**

`preloop`()

`prompt` = **'main:devices:delete> '**

class `netcrawl.credentials.menu.`**`MainMenu`**(*completekey='tab'*, *stdin=None*, *stdout=None*)
Bases: *`netcrawl.credentials.menu.UserPrompt`*

`do_1`(*args*)
List current device usernames and a hash of their passwords

`do_2`(*args*)

`do_3`(*args*)
Replace the current database login

`do_r`(*args*)

`intro` = **'Modify device credentials and database service accounts.\nType help or ? to list commands.\n\nChoose from th**

`prompt` = **'netcrawl> '**

class `netcrawl.credentials.menu.`**`ModifyDevice`**(*completekey='tab'*, *stdin=None*, *stdout=None*)
Bases: *`netcrawl.credentials.menu.UserPrompt`*

`do_1`(*args*)
Add a device credential to secure storage

`do_2`(*args*)
Delete a credential

`intro` = **'\nModify device credentials\n\nChoose from the following:\n1) Add device credential\n2) Delete device credenti**

`prompt` = **'netcrawl:devices> '**

class `netcrawl.credentials.menu.`**`UserPrompt`**(*completekey='tab'*, *stdin=None*, *stdout=None*)

Bases: `cmd.Cmd`

`do_q`(*args*)

Quits the program.

`do_r`(*args*)

`emptyline`()

`precmd`(*line*)

Accepts lowervase or uppercase input

`netcrawl.credentials.menu.`**`start`**()

Module contents

netcrawl.devices package

Submodules

netcrawl.devices.base module

class `netcrawl.devices.base.`**`Interface`**(***kwargs*)

Bases: `object`

Generic network device interface

`get_network_ip`()

class `netcrawl.devices.base.`**`NetworkDevice`**(***kwargs*)

Bases: `object`

Generic network device

`add_ip`(*ip*)

Adds an IP address to the list of other IPs

Parameters **`ip`** (*`string`*) – An IP address

`alert`(*msg*, *proc*, *failed=False*, *v=2*, *ip=None*)

Populates the failed messages variable for the device

`all_neighbors`()

`credentials`(*username=None*, *password=None*, *cred_type=None*)

Gets or sets the last successful credential used to log in to the device

`first_serial_str`()

`get_ips`()

Returns a list of IP addresses aggregated from interfaces.

`get_serials`()

`interfaces_to_string`()

`merge_interfaces`(*new_interfaces*)

Merges a list of interfaces with the ones currently in the device. If the interface name matches, then the new interface will overwrite any old data it has new entries for.

Parameters **`new_interfaces`** (*`List of interface objects`*) – One or more interface objects

`neighbor_table`(*sh_src=True*, *sh_name=True*, *sh_ip=True*, *sh_platform=True*)
Create a formatted table of neighbors.

Keyword Arguments

- **`sh_src`** (*`bool`*) – When true, show the source interface for each entry
- **`sh_name`** (*`bool`*) – When true, show the hostname for each entry
- **`sh_ip`** (*`bool`*) – When true, show the IP address for each entry
- **`sh_platform`** (*`bool`*) – When true, show the system platform for each entry

Returns A string representation of the PrettyTable containing this device's neighbors.

Return type str

`process_device`()
Main method which fully populates the network_device

`save_config`()

`short_pass`()

`unique_name`
Returns a unique identifier for this device

netcrawl.devices.cisco_device module

Created on Feb 19, 2017

@author: Wyko

class `netcrawl.devices.cisco_device.`**`CiscoDevice`**(**args*, ***kwargs*)
Bases: *`netcrawl.devices.base.NetworkDevice`*

`get_serials`()

`match_partial_to_full_interface`(*partial*)
Given a partial MAC address, iterate through all of this device's interfaces and match the address to an interface. Return the interface.

1. Split the partial interface by name and number
2. For each interface, check if the interface name starts with the partial name
3. If so, check if the interface number matches the partial interface number
4. Return the full interface name

`parse_neighbor`(*cdp_input*)
Accepts a single CDP neighbor entry and parses it into a dictionary.

`parse_netmiko_platform`(*cdp_input*)

`split_interface_name`(*interface_name*)
Returns a tuple containing (interface_type, interface_number)

netcrawl.devices.ios_device module

Created on Feb 19, 2017

@author: Wyko

class `netcrawl.devices.ios_device.`**`IosDevice`**`(`**args*, ***kwargs*`)`

Bases: *`netcrawl.devices.cisco_device.CiscoDevice`*

netcrawl.devices.nxos_device module

Created on Feb 18, 2017

@author: Wyko

class `netcrawl.devices.nxos_device.`**`NxosDevice`**`(`**args*, ***kwargs*`)`

Bases: *`netcrawl.devices.cisco_device.CiscoDevice`*

`get_interfaces_config``()`

`get_interfaces_xml``()`

`get_serials``()`

Returns serials based on XML output

Module contents

netcrawl.tools package

Subpackages

netcrawl.tools.manuf package

Subpackages

netcrawl.tools.manuf.test package

Submodules

netcrawl.tools.manuf.test.test_manuf module

class `netcrawl.tools.manuf.test.test_manuf.`**`ManufTestCase`**`(`*methodName='runTest'*`)`

Bases: `unittest.case.TestCase`

`MANUF_URL` **= 'https://raw.githubusercontent.com/coolbho3k/manuf/master/manuf/test/manuf'**

`setUp``()`

`test_getAllWithComplexNetmask_returnCorrectMatch``()`

`test_getAllWithSimpleNetmask_returnCorrectMatch``()`

`test_getAll_returnClosestMatch``()`

`test_getAll_supportAllMacFormats``()`

test_getAll_whenMacValid_getVendor()

test_getComment_getComment()

test_getManuf_getManuf()

test_update_update()

Module contents

Submodules

netcrawl.tools.manuf.manuf module

Parser library for Wireshark's OUI database.

Converts MAC addresses into a manufacturer using Wireshark's OUI database.

See README.md.

class `netcrawl.tools.manuf.manuf.`**MacParser**(*manuf_name='/home/docs/checkouts/readthedocs.org/user_builds/netcraw update=False*)

Bases: `object`

Class that contains a parser for Wireshark's OUI database.

Optimized for quick lookup performance by reading the entire file into memory on initialization. Maps ranges of MAC addresses to manufacturers and comments (descriptions). Contains full support for netmasks and other strange things in the database.

See https://www.wireshark.org/tools/oui-lookup.html

Parameters

- **manuf_name** (*str*) – Location of the manuf database file. Defaults to "manuf" in the same directory.
- **update** (*bool*) – Whether to update the manuf file automatically. Defaults to False.

Raises `IOError` – If manuf file could not be found.

MANUF_URL = 'https://code.wireshark.org/review/gitweb?p=wireshark.git;a=blob_plain;f=manuf'

get_all(*mac*)

Get a Vendor tuple containing (manuf, comment) from a MAC address.

Parameters **mac** (*str*) – MAC address in standard format.

Returns Vendor namedtuple containing (manuf, comment). Either or both may be None if not found.

Return type *Vendor*

Raises `ValueError` – If the MAC could not be parsed.

get_comment(*mac*)

Returns comment from a MAC address.

Parameters **mac** (*str*) – MAC address in standard format.

Returns String containing comment, or None if not found.

Return type string

Raises `ValueError` – If the MAC could not be parsed.

`get_manuf`(*mac*)

Returns manufacturer from a MAC address.

Parameters **`mac`** (*`str`*) – MAC address in standard format.

Returns String containing manufacturer, or None if not found.

Return type string

Raises `ValueError` – If the MAC could not be parsed.

`refresh`(*manuf_name=None*)

Refresh/reload manuf database. Call this when manuf file is updated.

Parameters **`manuf_name`** (*`str`*) – Location of the manuf data base file. Defaults to "manuf" in the same directory.

Raises `IOError` – If manuf file could not be found.

`search`(*mac, maximum=1*)

Search for multiple Vendor tuples possibly matching a MAC address.

Parameters

- **`mac`** (*`str`*) – MAC address in standard format.
- **`maximum`** (*`int`*) – Maximum results to return. Defaults to 1.

Returns List of Vendor namedtuples containing (manuf, comment), with closest result first. May be empty if no results found.

Raises `ValueError` – If the MAC could not be parsed.

`update`(*manuf_url=None, manuf_name=None, refresh=True*)

Update the Wireshark OUI database to the latest version.

Parameters

- **`manuf_url`** (*`str`*) – URL pointing to OUI database. Defaults to database located at code.wireshark.org.
- **`manuf_name`** (*`str`*) – Location to store the new OUI database. Defaults to "manuf" in the same directory.
- **`refresh`** (*`bool`*) – Refresh the database once updated. Defaults to True. Uses database stored at manuf_name.

Raises `URLError` – If the download fails

class `netcrawl.tools.manuf.manuf.`**`Vendor`**(*manuf, comment*)

Bases: `tuple`

`comment`

Alias for field number 1

`manuf`

Alias for field number 0

`netcrawl.tools.manuf.manuf.`**`main`**()

Simple command line wrapping for MacParser.

Module contents

Submodules

netcrawl.tools.find_unknown_switches module

Created on Mar 17, 2017

@author: Wyko

`netcrawl.tools.find_unknown_switches.`**`run_find_unknown_switches`**(*filter_device_name=[]*, *filter_interface_name=[]*, *filter_manufacturer=[]*, *min_macs=3*)

netcrawl.tools.locate_mac module

netcrawl.tools.locate_mac – Lists the devices and ports that the specified MAC was seen on

@author: Wyko ter Haar @license: MIT @contact: vegaswyko@gmail.com

`netcrawl.tools.locate_mac.`**`locate`**(*macs*)

`netcrawl.tools.locate_mac.`**`main`**(*argv=None*)
Command line options.

netcrawl.tools.mac_audit module

`netcrawl.tools.mac_audit.`**`evaluate_mac`**(*mac1*, *mac2*)

`netcrawl.tools.mac_audit.`**`main`**()
Begins the audit. Outputs a csv file containing the audit results.

`netcrawl.tools.mac_audit.`**`run_audit`**(*csv_path*)
Given a CSV of subnets and MAC addresses, search the database for all MACs on subnets which match those in the CSV. Compare each MAC and output a new csv with any matching MAC's listed by confidence (number of matching characters, starting from the OUI. This can be used, for example, for a Wireless Rogue SSID audit, for which the MAC address of the radios is known and you want to find out which rogue AP's are physically connected to your network.

`netcrawl.tools.mac_audit.`**`sort_csv_by_subnet`**(*csv_rows*)
Takes a list of dicts with 'network_ip' and 'mac' as keys, then produces a dict of lists containing subnets and the mac addresses associated with them

`netcrawl.tools.mac_audit.`**`write_csv`**(*rows*)

`netcrawl.tools.mac_audit.`**`write_report`**(*rows*)

Module contents

netcrawl.wylog package

Submodules

netcrawl.wylog.logging module

Created on Mar 4, 2017

@author: Wyko

`netcrawl.wylog.logging.`**`log`**(*msg*, ***kwargs*)

Writes a message to the log.

Parameters **`msg`** (*`str`*) – The message to write.

Keyword Arguments

- **`ip`** (*`str`*) – The IP address of whatever device we are connected to
- **`proc`** (*`str`*) – The process which caused the log entry, in the form of *'module.method_name'*
- **`log_path`** (*`str`*) – The filepath of the directory where to save the log file. Uses config.cc.log_path by default
- **`print_out`** (*`bool`*) – If True, copies the message to console
- **`v`** (*`int`*) – Verbosity level. Logs with verbosity above the global verbosity level will not be printed out. v= 1: Critical alerts v= 2: Non-critical alerts v= 3: High level info v= 4: Common info v= 5-6: Debug level info
- **`error`** (*`Exception`*) – An exception object to be included in the log output

Returns True if write was successful.

Return type bool

class `netcrawl.wylog.logging.`**`log_snip`**(*proc*, *v=5*)

Bases: `object`

`netcrawl.wylog.logging.`**`logf`**(*f*, ***kwargs*)

netcrawl.wylog.multi module

Created on Mar 4, 2017

@author: Wyko

class `netcrawl.wylog.multi.`**`logged_lock`**(*name*)

Bases: `object`

This is a wrapper around the Multiprocessing Lock class that includes some logging.

Module contents

Submodules

netcrawl.cli module

Created on Feb 28, 2017

@author: Wyko

`netcrawl.cli.`**`connect`**(*handler=None*, *netmiko_platform=None*, *ip=None*, *cred=None*, *port=None*)

Starts a CLI session with a remote device.

Uses Netmiko to start a SSH or Telnet session with a target device. It will attempt to use SSH first, and if it fails it will try Telnet. For each connection method, it will attempt each credential specified in the cred argument (if specified) or the config.cc.credentials list otherwise.

Keyword Arguments

- **`cred`** (*`dict`*) – If supplied, this method will only use the specified credential. Uses the config.cc.credentials list otherwise.
- **`port`** (*`int`*) – If supplied, this method will connect only on this port
- **`ip`** (*`str`*) – The IP address to connect to
- **`netmiko_platform`** (*`str`*) – The platform of the device, in the Netmiko format
- **`handler`** (*`ConnectHandler`*) – A Netmiko-type handler to use. Currently using one of Netmiko.ConnectHandler, Netmiko.ssh_autodetect.SSHDetect. Uses Netmiko.ConnectHandler by default.

Returns

A dict containing:

- **connection** (*ConnectHandler*): A Netmiko ConnectHandler object with a successfully opened connection
- **tcp_22** (*bool*): True if port 22 is open
- **tcp_23** (*bool*): True if port 23 is open
- **username** (*str*): The first successful credential's username
- **password** (*str*): The first successful credential's password
- **cred_type** (*str*): The first successful credential's type

Return type dict

Raises

- `IOError` – If a connection could not be established
- `AssertionError` – If error checking failed

netcrawl.config module

class `netcrawl.config.`**`Config`**

Bases: `object`

`check_credentials`()

`set_all_database_creds`(*username*, *password*)

class `netcrawl.config.`**`Database`**(*dbname*)

Bases: `object`

`args`

Returns a dict used to populate a psycopg2 connection

`netcrawl.config.`**`parse_config`**()

netcrawl.core module

netcrawl.core.**nmap_scan**(*target*, ***kwargs*)

Ping each host in a given range one at a time. When a live host is found, add it to the pending hosts database.

Parameters **target** (*str*) – An Nmap compatible target specifier as outlined in the Nmap documentation

Keyword Arguments ****kwargs** – Arguments to pass to *netcrawl.io_sql.main_db*

netcrawl.core.**print_report**()

Prints a brief report of the state of the databases to the console

netcrawl.core.**recursive_scan**(***kwargs*)

Starts a **Recursive Scan** (-sR) run. This is the main scanning method for netcrawl.

1. If a `target` kwarg is given, add that seed device to the list of pending deivces, even if it was already visited.

2. Create workers (subprocesses) to perform the scanning work, up to 16 per CPU core, or up to the `processes` kwarg per core if that kwarg was given.

3. Query the Pending table in the Main database for pending devices.

4. Autodetect the Netmiko platform for each device if needed.

5. Inventory the device using *netcrawl.devices.base.NetworkDevice.process_device()*

6.Add each discovered device to the Inventory database

Keyword Arguments

- **skip_named_duplicates** (*bool*) – If True, this will cause netcrawl to skip neighbors which have the same hostname as a device that was previously visited.

  Note: While this can potentially save a lot of time when scanning devices, if multiple different devices share the same hostname, they will not be scanned!

- **target** (*str*) – The IP address of a seed device to add to the pending devices database
- **netmiko_platform** (*str*) – The Netmiko platform of the `target` device, if one was given.
- **processes** (*int*) – The number of worker processes to create, multiplied by the CPU count

Note: If there are any remaining keyword arguments in `**kwargs`, they will be passed to *netcrawl.io_sql.main_db* and *netcrawl.io_sql.device_db*

netcrawl.core.**single_scan**(*target*, *netmiko_platform='unknown'*)

Starts a **Single Scan** (-sS) run. This scan polls a single device and presents information about the device to the console. Useful for testing a connection, as well as getting a quick overview of the target.

Keyword Arguments

- **target** (*str*) – The network address of the device to scan
- **netmiko_platform** (*str*) – The Netmiko platform of the `target` device. If one is not given, it will attempt to autodetect the device type.

netcrawl.device_dispatcher module

Controls selection of proper class based on the device type.

Credit: Kirk Byers

`netcrawl.device_dispatcher.`**`autodetect`**`(`*target*`)`

This method invokes Netmiko's autodetect functionality to determine the correct device class, then returns that class as a netmiko_platform.

Parameters **`target`** (*`String`*) – The hostname or IP address to connect to

Raises

- `TypeError` – Could not find an appropriate class to inherit
- `IOError` – Could not connect to the device

Returns

The netmiko_platform representation of the proper device class.

Return type String

`netcrawl.device_dispatcher.`**`create_instantiated_device`**`(`**args*, ***kwargs*`)`

Factory function selects the proper network device class and creates the object based on netmiko_platform.

netcrawl.io_sql module

class `netcrawl.io_sql.`**`device_db`**`(`***kwargs*`)`

Bases: *`netcrawl.io_sql.sql_database`*

`add_device_nd``(`*_device*`)`

Appends a device to the database

Parameters **`_device`** (*`network_device`*) – A single network_device

Returns False if write was unsuccessful Int: Index of the device that was added, if successful

Return type Boolean

`create_table``(`*drop_tables=True*`)`

`delete_device_record``(`*id*, *cur=None*`)`

Removes a record from the devices table

`device_macs``(`*device_id*`)`

`devices_on_subnet``(`*subnet*`)`

`exists``(`*device_id=None*, *unique_name=None*, *device_name=None*`)`

Checks whether a device record is present in the devices table. Tries each supplied identifier in order until a match is found, then returns the device_id of the found record.

Keyword Arguments

- **`device_id`** (*`int`*) – If not None, check the device_id column for a match
- **`unique_name`** (*`str`*) – If not None, check the unique_name column for a match
- **`device_name`** (*`str`*) – If not None, check the device_name column for a match

Returns int: The device_id of the first match found bool: False if not found

get_device_record(*column*, *value*)
Get a device record based on a lookup column. 'WHERE column = value'

Returns psycopg2 dict object

insert_device_entry(*device*, *cur*)

insert_interface_entry(*device_id*, *interf*, *cur*)

insert_mac_entry(*device_id*, *interface_id*, *mac_address*, *cur*)

insert_neighbor_entry(*device_id*, *interface_id*, *neighbor*, *cur*)

insert_neighbor_ip_entry(*neighbor_id*, *ip*, *cur*)

insert_serial_entry(*device_id*, *serial*, *cur*)

ip_exists(*ip*)

locate_mac(*mac*, *cur=None*)

macs_on_subnet(*subnet*)

process_duplicate_device(*device*)
Parent method for handling an existing device which needs to be updated.

1. Determine if the `device` exists and, if so, get the device_id
2. Overwrite all entries in the device with the new device
3. Set a new updated time for all dependent tables
4. Delete any interfaces and serials which no longer exist
5. Add any new interfaces and serials
6. Add any new MAC addresses
7. Update any newly non-existent MAC addresses

Parameters **device** (`NetworkDevice`) – A network device object to check against for duplicates

Returns True if a duplicate was found and updated

Return type bool

set_dependents_as_updated(*device_id*, *cur=None*)
Sets the last touched time on all dependents of the given device_id to now

update_device_entry(*device*, *cur=None*, *device_id=None*, *unique_name=None*)
Overwrites all entries in the Devices table with a matching `device_id` or `unique_name` with the information in `device`.

Parameters **device** (`NetworkDevice`) – The device to source updates from

Keyword Arguments

- **cur** (*psycopg2.cursor*) – Cursor object used to update the database
- **device_id** (*int*) – If not None, overwrites the row at this index with `device`
- **unique_name** (*str*) – If not None, overwrites any row with a matching `unique_name` field with `device`.

Note: If both `device_id` and `unique_name` are given, the method will update **all** entries that match **either** key.

Raises `ValueError` – No `unique_name` or `device_id` passed to the method

class `netcrawl.io_sql.`**`main_db`**`(`***kwargs*`)`

Bases: *`netcrawl.io_sql.sql_database`*

`add_device_pending_neighbors``(`*_device=None*, *_list=None*`)`

Appends a device or a list of devices to the database

Optional Args: _device (network_device): A single device _list (List): List of devices

Returns True if write was successful, False otherwise.

Return type Boolean

`add_pending_device_d``(`*device_d=None*, *cur=None*, ***kwargs*`)`

`add_visited_device_d``(`*device_d=None*, *cur=None*, ***kwargs*`)`

`add_visited_device_nd``(`*_device=None*, *_list=None*, *cur=None*`)`

Appends a device or a list of devices to the database

Optional Args: _device (network_device): A single network_device _list (List): List of network_device objects

Returns True if write was successful, False otherwise.

Return type Boolean

`count_pending``()`

Counts the number of rows in the table

`count_unique_visited``()`

Counts the number of unique devices in the database

`create_table``(`*drop_tables=True*`)`

`get_next``()`

Gets the next pending device.

Returns

The next pending device as a dictionary object with the names of the rows as keys.

Return type Dict

`remove_pending_record``(`*_id*`)`

Removes a record from the pending table

`remove_visited_record``(`*ip*`)`

Removes a record from the pending table

class `netcrawl.io_sql.`**`sql_database`**`(`***kwargs*`)`

Bases: `object`

A base class to facilitate SQL database operations.

Keyword Arguments **clean** (*bool*) – If True, this causes all database tables to be dropped in order to start with a clean database.

Warning: Obviously, this is really dangerous.

close()
Closes the connection to the database if it is open

count(*table*, *column='*'*, *value=None*, *partial_value=None*, *distinct=False*, *cur=None*)
Counts the occurrences of the specified `column` in a given `table`.

Parameters **table** (*str*) – The table to search in

Keyword Arguments

- **column** (*str*) – The column to count
- **distinct** (*bool*) – If True, count only unique matches
- **value** (*str*) – If not None, adds a where clause to the count in the format:

```
WHERE column = 'value'
```

- **partial_value** (*str*) – If not None, adds a where clause which will match a partial string in the format:

```
WHERE column like '%partial_value1%'
```

Returns The number of matches

Return type int

create_database(*new_db*)
Creates a new database

Parameters **new_db** (*str*) – Database name to create

database_exists(*db*)
Returns true is the specified database exists

Parameters **db** (*str*) – A database name

Returns True if the database exists

Return type bool

delete_database(*dbname*)
Deletes a database

Returns True if the database was created

Return type bool

Raises

- `FileExistsError` – If the database to be deleted does not exist.
- `IOError` – If the database could not be deleted and still exists after execution

execute_sql(**args*, *proc=None*, *fetch=True*)
Executes a SQL snippet and optionally gets the results

Parameters ***args** – The arguments to pass along to `pyscopg2.cursor.execute()`. Usually a string containing the SQL statement, and potentially a tuple of parameters.

Keyword Arguments

- **proc** (*str*) – The name of the parent process, for logging purposes
- **fetch** (*bool*) – If True, fetches all results from the query

Returns The results of `pyscopg2.cursor.fetchall()`

Return type tuple

execute_sql_gen(**args*, *proc=None*)

Executes a SQL snippet and gets the results in a generator

Parameters ***args** – The arguments to pass along to `pyscopg2.cursor.execute()`. Usually a string containing the SQL statement, and potentially a tuple of parameters.

Keyword Arguments **proc** (*str*) – The name of the parent process, for logging purposes

Returns The results of `pyscopg2.cursor.fetchall()`

Return type generator

ip_exists(*ip*, *table*)

Check if a given IP exists in the database

Parameters

- **ip** (*str*) – The IP address to check for
- **table** (*str*) – The table to check

Raises `ValueError` – If an argument is an improper type

ip_name_exists(*ip*, *name*, *table*, *cur=None*)

Check if a given IP OR Name exists in the database

class `netcrawl.io_sql.`**sql_logger**(*proc*, *ignore_duplicates=True*)

Bases: `object`

Utility class to enable logging and timing SQL execute statements, as well as handling specific errors

`netcrawl.io_sql.`**useCursor**(*func*)

Decorator that creates a cursor object to pass to the wrapped method in case one wasn't passed originally. The wrapped function should accept `cur` as an argument.

netcrawl.util module

class `netcrawl.util.`**benchmark**(*name*)

Bases: `object`

Context manager which times the surrounded code and prints the results to the console

`netcrawl.util.`**cidr_to_netmask**(*cidr*)

Changes CIDR notation to subnet masks. I honestly have no idea how this works. I just added some error checking.

class `netcrawl.util.`**cleanExit**

Bases: `object`

Context manager who's only purpose is to cleanly exit when the code execution is interrupted by the user

`netcrawl.util.`**clean_ip**(*ip*)

Removes all non-digit or period characters from the source string

`netcrawl.util.`**`contains_mac_address`**(*mac*)
Simple boolean operator to determine if a string contains a mac anywhere within it.

`netcrawl.util.`**`getCreds`**()
Get stored credentials using a the credentials module. Requests credentials via prompt otherwise.

Returns

[{username, password, cred_type},]

If the username and password had to be requested, the list will only have one entry.

Return type List of Dicts

`netcrawl.util.`**`is_ip`**(*raw_input*)
Returns true if the given string is an IPv4 address

`netcrawl.util.`**`netmask_to_cidr`**(*netmask*)
Translates a netmask to a CIDR format

Parameters **`netmask`** (*`str`*) – A netmask in four octet ip address format

Returns The CIDR representation of the netmask

Return type int

`netcrawl.util.`**`network_ip`**(*ip*, *subnet*)
Returns the network IP address calculated from the given `ip` and `subnet`.

`netcrawl.util.`**`parse_ip`**(*raw_input*)
Returns a list of strings containing each IP address matched in the input string.

`netcrawl.util.`**`port_is_open`**(*port*, *address*, *timeout=5*)
Checks a socket to see if the specified `port` is open.

Parameters

- **`port`** (*`int`*) – The numbered TCP port to check
- **`address`** (*`str`*) – The address of the host to check

Keyword Arguments **`timeout`** (*`int`*) – The number of seconds to wait before timing out. Defaults to 5 seconds. Zero seconds disables timeout

Returns True if the port is open

Return type bool

`netcrawl.util.`**`timeit`**(*method*)
Decorator method which times the wrapped method and prints the result to the console

`netcrawl.util.`**`ucase_letters`**(*raw_input*)
Returns the input string stripped of everything but letters, numbers, and underscores.

Module contents

Python Module Index

Index