SecureCRT Tools Documentation

Release 2.1.0

Jamie Caesar

Feb 17, 2021

Contents:

CHAPTER 1

Introduction

This repository contains a collection of SecureCRT scripts that automate various tasks, primarily around interacting with Cisco routers and switches.

These scripts should work on any version of SecureCRT that supports python. If you find that a script won't work on your machine, please post an issue to let us know!

CHAPTER 2

Important Note For Users of Older Versions

The settings files for these scripts have been changed from using JSON files to the Python built-in ConfigParse module. In addition instead of each script uses indivdiual settings having its own JSON file, now that settings are saved in the common "settings.ini" file under a separate heading for that script. **There is no code to migrate your settings from the old JSON format to the INI format, so please check your settings and remove the old JSON files**

In addition to the new format for the settings, the newer version of these scripts now have support for initiating connections via Telnet and SSH to remote devices, as well as connecting via a jump/bastion host. In addition there are methods for pushing configuration changes to devices that were not available previously.

If you are looking for previous versions of the scripts, they can be found in the branches below:

- Please see the *Pre-2017* branch if you need to access the original versions (1.0) that were all function based.
- Please see the *2017* branch if you want the original class-based scripts use the JSON based settings files. (2.0)

CHAPTER 3

What These Scripts Do

While the documentation has a detailed list of every script in this collection and the specifics on how they work, below is a summarized list of the kinds of things these scripts will do.

- Save command outputs from devices into files that are automatically named with the hostname of the device (from the prompt) and a time/date stamp. There are some different versions depending on if you want a single output or multiple outputs and from one or multiple devices.
- Capture device inventory data (code version, model number, serial number, mfg. date, etc.) for a list of devices provided to the script in CSV format.
- Write the detailed CDP neighbor information into a spreadsheet (CSV format) for easier viewing and re-use of the data.
- Creation of SecureCRT sessions from the CDP information of a device, to quickly build your collection of sessions in SecureCRT's session manager.
- Summarize the route table of a device to see a list of all next-hops found in the route table and how many routes from which routing protocols are sending routes to each next-hop. This script is useful either as a validate tool after routing changes (see a summary of route behavior before and after the change), or to help with discovery of new devices (There are 4000 routes in the table, but are there 3 or 30 exits that packets can take?)
- Write the ARP table for a device into a spreadsheet (CSV) file, either for manual lookups or to be leveraged by other scripts (see below). There is also version that will build a single large ARP table from multiple devices (For when HSRP priorities are split across 2 cores, or multiple VRFs route different VLANs upstream).
- Create a spreadsheet that maps out every device on the switch, including interface description, MAC Address, MAC Vendor and IP address. This script uses the ARP table created above as input if you want MAC to IP mappings shown in the output.
- Capture the interface stats from all interfaces on a device into a spreadsheet to more quickly see which ports have errors, high rates of traffic, etc.
- Search devices for specific existing IP helper/DHCP relay addresses and add new relays (optionally remove old) on any interface where the current relays are found. There are versions of this script for working with a single device or a list of devices.

These various scripts are included in the repository so that someone can quickly download them and get started, but majority of the work has been put into building the *securecrt_tools* module which is designed to handle all of the low-level interactions with SecureCRT and make it as easy as possible to write new scripts. This module handles discovering the remote device OS, its prompt and hostname, and the interactions with the device. For example a single method call can send a command, collect the output and write it to a file named after the device. This way a script should be able to gather the output needed in a few lines of code and anything beyond that is the processing required to parse that output (TextFSM makes this much easier) and take the appropriate follow up steps. All of this is discussed in more detail in the "Writing Your Own Scripts" sectin of the documentation.

CHAPTER 4

Using a Jumpbox/Bastion Host

In some cases you can only access a remote device by proxying through a jump box/bastion host. Fortunately, SecureCRT already has a method of handling this and so I don't have to build the code directly into the securecrt_tools module to do it. The steps for proxying a connection through another device are:

1) Create an SSH2 session to connect to the jump box. Make sure you can use this session to connect to the jump box directly.
2) Create a session to connect to the remote device by IP or name (that the jump box can resolve).
3) While editing the remote device session, go to the SSH2/SSH1/Telnet section and look for the *Firewall* drop down.
4) Choose *Select Session* and select the session for the jump box.
5) When you launch the remote device session, you'll first be prompted for the jump box credentials (unless you've saved them) and then you'll be prompted for the remote device credentials. You should now be connected to the remote device by proxying through the jump box.

For more information, watch the video on VanDyke Software's YouTube channel at https://youtu.be/XHOVTuv-LKY.

CHAPTER 5

Running The Scripts

There are 2 types of scripts in this repository:

1) Scripts that interact with a single device, AFTER you have logged in manually (starts with 's_'), and
2) Scripts that interat with multiple devices, where the script performs the login action (starts with 'm_')

A list of all the single- and multi-device scripts and descriptions on what they do can be found in the documentation below.

The run any of these scripts, you need to download the entire repo to your computer. You can either clone the repository or download an archive to extact on your machine.

NOTE While the scripts are running they will lock the tab until they complete. If for some reason (error, etc) the script ends and the tab is still locked, you can unlock it by going to **File->Unlock Session** in the menus.

5.1 Single Device Scripts

To run SINGLE device scripts, do the following:

1) **AFTER** connecting and logging into a device with SecureCRT, go to the *Scripts* menu and select "Run"
2) Choose the script you want to run (that starts with 's_')
3) The script looks for your *settings.ini* file. If this file doesn't exist (and it won't the first time you run one of these scripts) the script will create the file.
4) If the script produces an output, it will be saved in the directory specified in the *settings/settings.ini* file. If this diretory does not exist, you will be prompted to create it. You can modify this path in the *settings.ini* file to change where the scripts save the output they produce.

The output files are automatically named based on the hostname of the device connected to. This name is taken from the prompt of the device, so these scripts will work whether you are directly connected, or connected via a jumpbox or other intermediate device.

5.2 Multiple Device Scripts

1) While **NOT** connected to a device, go to the *Scripts* menu and select "Run"
2) The script will prompt you to select a CSV file that contains all the required information for the devices the script should connect to. You will be prompted for credentials, if required. **A sample device file can be found at templates/sample_device_list.csv**
3) The script will connect to each device and execute the script logic. The script will process one device at a time in the same tab. While this it the case because SecureCRT does not support multi-threading within scripts, you can manually multi-thread by breaking your devices file into multiple files and lauching the same script in multiple tabs with differnet device files.

CHAPTER 6

Settings

All settings files are stored in the *settings/settings.ini* file from the root of the scripts directory.

6.1 Global Settings

Global settings that are used by all scripts are under the *Global* heading in the *settings.ini* file. The following options are available in the global settings file:

- '**output_dir**': This is the path where you want the output from scripts to be saved.
- '**date_format**': Default is '%Y-%m-%d-%H-%M-%S'. This string specifies how the date stamp in output filenames is formatted. - %Y - 4-digit Year - %m - numeric month - %d - day of the month - %H - Hours - %M - Minutes - %S - Seconds
- '**modify_term**': True or False. When True, the script will attempt to modify the terminal length and width to 0 so that output flows continuously. When the output is complete the script will return the length and width to their original values. If False, it will not change the values, but instead auto-advance when a "More" prompt is encountered.
- '**debug_mode**': True or False. If True, a log file will be written that contains debug messages from the script execution. This can be helpful for troubleshooting scripts that are failing. The debug files will be saved in a *debugs* directory under your configured output directory.
- '**use_proxy**': True or False. If True, scripts that initiate connections (multi-device scripts) will use the *proxy_session* option below to specify which SecureCRT Session to use as a SOCKS proxy. When enabled, this option uses the *Firewall* setting in the SecureCRT sessions settings to specify the device to proxy the connection through.
- '**proxy_session**': The name of the SecureCRT session that should be used to proxy connections. This **MUST** be a session that uses SSH2. Use the forward slash (/) to specify folders in the path to the session, i.e. *proxy_session = Site 1/Core/S1_Core1*.

6.2 Script-Specific Settings

Some scripts have settings that are used to change certain behaviors while running. If such a settings are used, the setting will be saved under a heading named for the script in the *settings.ini* file. Details about the settings used by a script are described in the documentation for that script, or in the docstring in the script file itself.

CHAPTER 7

Contributing

While I've tried to create an assortment of scripts that would be useful to most network professionals, I would love for people to contribute to this repository by adding script and making improvements via pull request. These improvements can include bug fixes or support for additional devices beyond the few Cisco OSes I have access to test against. The majority of these scripts were created to do things that I've found useful over time, but I'm sure there are plenty more great ideas for scripts that I haven't thought of.

If you have a need for a script but do not feel confident that you can write one yourself, please post the idea in the issues log and perhaps someone will find the time to write it. Ultimately, if you have the interest, the best way to learn both Python and how to write your own scripts using these tools is by coming up with something you want to build and just keep working at it. Blank script templates (in the *templates* folder) are provided to help with getting started quickly and all of the existing scripts can be used as examples or modified to suit your needs. Since there are currently very few contributors to this repository the fastest way to get a new script to do what you need is to try to write it yourself and reach out for feedback and help. I can't guarantee that anyone will have the time to build a suggested script if suggested, but I'd still love to have those ideas posted even if it doesn't meet your timeline.

To help support involvement from others in the community, I've tried to write comprehensive documentation about both the high-level design/logic of the modules and scripts, as well as detailed documentation about all of the functions/methods in the modules. This include docstrings and comments within the code to make it as easy as possible for people new to this repository to understand what it is doing and to understand the existing capabilities thta can be used to save time writing new scripts. Please reach out with any feedback you have on the documentation so it can be continuously improved, even for simple typos and grammar errors that you find (or better yet, create a pull request to fix the file as practice using git and github!)

CHAPTER 8

Scripts

Below is a list of scripts that are available in this repository. Documentation for each script (including any functions within the script) can be found by clicking on the link below.

8.1 Single Device Scripts

Single Device Scripts are scripts that are designed to be lauched in a SecureCRT tab that is already connected to a remote device. These scripts will make necessary terminal changes, perform their task, and return the terminal back to the original state so that the user can continue working after the script executes.

These scripts can also be imported into Multi-Device scripts to reduce the amount of copying and pasting of code that is required to make a multi-device version.

8.1.1 s_add_global_config

s_add_global_config.**script_main**(*session*)

SINGLE device script

Author: Jamie Caesar

Email: jcaesar@presidio.com

This script will add global configuration commands to the connected device. The commands sent will depend on the operating system of the connected device. For example, IOS devices get the commands listed in the 'ios' section of the settings for this script. If the device is running NX-OS, it will get the commands from the 'nxos' section of the settings, etc.

This script will prompt you to run in "Check Mode", where the configuration changes the script would be pushed to the device are ONLY written to a file and NO CHANGES will be made to the device. If you select "No" when prompted this script will push the configuration changes to the device. Also, when the changes are pushed to the device this script will save the running config before and after the changes are made, and will also output a log of the configuration session showing all the commands pushed.

Script Settings (found in settings/settings.ini):

- **show_instructions** - When True, displays a pop-up upon launching the script explaining where to modify the list of commands sent to devices. This window also prompts the user if they want to continue seeing this message. If not, the script changes this setting to False.
- **ios** - A comma separated list of commands that will be sent to IOS devices.
- **ios-xr** - A comma separated list of commands that will be sent to IOS-XR devices.
- **nxos** - A comma separated list of commands that will be sent to NX-OS devices.
- **asa** - A comma separated list of commands that will be sent to ASA devices.

Parameters **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)

8.1.2 s_AireOS_collect_ap_detail

s_AireOS_collect_ap_detail.**script_main**(*session*)

SINGLE device script
Morphed: Gordon Rogier grogier@cisco.com
Framework: Jamie Caesar jcaesar@presidio.com

This script will capture details about the APs that are connected to the Wireless LAN Controller that this script is being run against and will output details for each AP into a CSV file.

Parameters **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)

s_AireOS_collect_ap_detail.**get_ap_detail**(*session*, *to_cvs=False*)
A function that captures the WLC AireOS ap details and returns an output list

Parameters **session** (*session.Session*) – The script object that represents this script being executed

Returns A list AP details

Return type list of dicts

8.1.3 s_AireOS_collect_ap_summ

s_AireOS_collect_ap_summ.**script_main**(*session*)

SINGLE device script
Morphed: Gordon Rogier grogier@cisco.com
Framework: Jamie Caesar jcaesar@presidio.com

This script will capture the WLC AireOS ap summary table and returns an output list

Parameters **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)

s_AireOS_collect_ap_summ.**get_ap_summ_table**(*session*)
A function that captures the WLC AireOS ap summary table and returns an output list

Parameters **session** (*session.Session*) – The script object that represents this script being executed

Returns A list of MAC information for AP summary

Return type list

8.1.4 s_AireOS_collect_auth_list

s_AireOS_collect_auth_list.**script_main**(*session*)

SINGLE device script

Morphed: Gordon Rogier grogier@cisco.com

Framework: Jamie Caesar jcaesar@presidio.com

This script will capture all WLC AireOS authorization lists and output them to a CSV file.

Parameters **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)

s_AireOS_collect_auth_list.**get_auth_list**(*session*, *to_cvs=False*)

A function that captures the WLC AireOS auth-list and returns an output list

Parameters **session** (*session.Session*) – The script object that represents this script being executed

Returns A list of MAC auth-list

Return type list of lists

8.1.5 s_AireOS_collect_interface_detail

s_AireOS_collect_interface_detail.**script_main**(*session*)

SINGLE device script

Morphed: Gordon Rogier grogier@cisco.com

Framework: Jamie Caesar jcaesar@presidio.com

This script will capture the WLC AireOS interface details and write them to a CSV file.

Parameters **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)

s_AireOS_collect_interface_detail.**get_interface_detail**(*session*, *to_cvs=False*)

A function that captures the WLC AireOS interface detail table and returns an output list

Parameters **session** (*session.Session*) – The script object that represents this script being executed

Returns A list of interface details

Return type list of lists

8.1.6 s_AireOS_collect_mobility_group

s_AireOS_collect_mobility_group.**script_main**(*session*)

SINGLE device script

Morphed: Gordon Rogier grogier@cisco.com

Framework: Jamie Caesar jcaesar@presidio.com

This script will capture the WLC AireOS mobility summary and returns an output list

Parameters **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)

s_AireOS_collect_mobility_group.**get_mobility_group**(*session*, *to_cvs=False*)

A function that captures the WLC AireOS mobility summary and returns an output list

Parameters **session** (*session.Session*) – The script object that represents this script being executed

Returns A list of mobility group peers

Return type list of lists

8.1.7 s_AireOS_collect_wlan_detail

s_AireOS_collect_wlan_detail.**script_main**(*session*)

SINGLE device script

Morphed: Gordon Rogier grogier@cisco.com

Framework: Jamie Caesar jcaesar@presidio.com

This script will capture the WLC AireOS wlan & remote-lan & guest-lan details and returns an output list

Parameters **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)

s_AireOS_collect_wlan_detail.**get_wlan_detail**(*session*, *to_cvs=False*)

A function that captures the WLC AireOS wlan & remote-lan & guest-lan details and returns an output list

Parameters **session** (*session.Session*) – The script object that represents this script being executed

Returns A list of wlan details

Return type list of lists

8.1.8 s_arp_to_csv

s_arp_to_csv.**script_main**(*session*)

SINGLE device script

Author: Jamie Caesar

Email: jcaesar@presidio.com

This script will capture the ARP table of the attached device and output the results as a CSV file. While this script can be used to capture the ARP table, the primary purpose is to create the ARP associations that the "s_switchport_mapping.py" script can use to map which MAC and IP addresses are connected to each device.

Parameters **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)

8.1.9 s_cdp_to_csv

s_cdp_to_csv.**script_main**(*session*)

SINGLE device script
Author: Jamie Caesar
Email: jcaesar@presidio.com

This script will grab the detailed CDP information from a Cisco IOS or NX-OS device and export it to a CSV file containing the important information, such as Remote Device hostname, model and IP information, in addition to the local and remote interfaces that connect the devices.

Script Settings (found in settings/settings.ini):

- **strip_domains** - A list of domain names that will be stripped away if found in the CDP remote device name.

Parameters **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)

8.1.10 s_create_sessions_from_cdp

s_create_sessions_from_cdp.**script_main**(*session*)

SINGLE device script
Author: Jamie Caesar
Email: jcaesar@presidio.com

This script will grab the detailed CDP information from a Cisco IOS or NX-OS device and create SecureCRT sessions based on the information. By default all sessions will be created as SSH2, so you may have to manually change some sessions to make them work, depending on the device capabilities/configuration.

Only devices that contain "Router" or "Switch" in their capabilities field of the CDP information will have sessions created for them. This skips phones, hosts like VMware or Server modules, and other devices that we don't usually log into directly).

NOTE ON DEFAULTS: This script uses the SecureCRT Default Session settings as a base for any sessions that are created. The folder where the sessions are saved is specified in the 'settings.ini' file, and the hostname and IP are extracted from the CDP information. All other setting defaults are configured within SecureCRT.

Script Settings (found in settings/settings.ini):

- **folder** - The path starting from the <SecureCRT Config>/Sessions/ directory where the sessions will be created.
- **strip_domains** - A list of domain names that will be stripped away if found in the CDP remote device name.

Parameters **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)

s_create_sessions_from_cdp.**create_session_list**(*cdp_list*)
This function takes the TextFSM output of the CDP information and uses it to create a list of new SecureCRT sessions to create (system name and IP address).

Parameters **cdp_list** (*list*) – The TextFSM output after processing the "show cdp neighbor detail" output

Returns A list (system name and IP address) of the sessions that need to be created.

Return type list

8.1.11 s_document_device

s_document_device.**document**(*session*, *command_list_name*, *folder_per_device*, *prompt_create_dirs=True*)

This function captures the output of the provided commands and writes them to files. This is separated into a separate function so it can be called by both the single-device and multi-device version of this script.

Parameters

- **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)
- **command_list** (*list*) – A list of commands that will be sent to the connected device. Each output will be saved to a different file.
- **output_dir** (*str*) – The full path to the directory where the output files are written.
- **folder_per_device** (*bool*) – A boolean that if true will create a separate folder for each device

Returns

s_document_device.**script_main**(*session*)

SINGLE device script

Author: Jamie Caesar

Email: jcaesar@presidio.com

This script will grab the output for a list of commands from the connected device. The list of commands is taken from the 'settings/settings.ini' file. There is a separate list for each supported network operating system (IOS, NXOS and ASA) and by default the list that matches the network operating system of the connected device will be used.

Custom lists of commands are supported. These lists can be added manually to the [document_device] section of the 'settings/settings.ini' file. To be able to choose one of these lists when running the script, the 'prompt_for_custom_lists' setting needs to be changed to 'True' in the settings.ini file. Once this option is enabled, the script will prompt for the name of the list that you want to use. If the input is left blank then the default behavior (based on network OS) will choose the list.

Script Settings (found in settings/settings.ini):

- **show_instructions** - When True, displays a pop-up upon launching the script explaining where to modify the list of commands sent to devices. This window also prompts the user if they want to continue seeing this message. If not, the script changes this setting to False.
- **folder_per_device** - If True, Creates a folder for each device, based on the hostname, and saves all files inside that folder WITHOUT the hostname in the output file names. If False, it saves all the files directly into the output folder from the global settings and includes the hostname in each individual filename.
- **prompt_for_custom_lists** - When set to True, the script will prompt the user to

type the name of a list of commands to use with the connected device. This list name must be found as an option in the [document_device] section of the settings.ini file. The format is the same as the default network OS lists, 'ios', 'nxos', etc.

- **ios** - The list of commands that will be run on IOS devices
- **nxos** - The list of commands that will be run on NXOS devices
- **asa** - The list of commands that will be run on ASA devices

Any additional options found in this section would be custom added by the user and are expected to be lists of commands for use with the 'prompt_for_custom_lists' setting.

By default, The outputs will be saved in a folder named after the hostname of the device, with each output file being saved inside that directory. This behavior can be changed in the settings above.

Parameters **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)

8.1.12 s_eigrp_topology_summary

s_eigrp_topology_summary.**script_main**(*session*, *ask_vrf=True*, *vrf=None*)

SINGLE device script
Author: Jamie Caesar
Email: jcaesar@presidio.com

This script will grab the EIGRP topology from a Cisco IOS or NXOS device and export a summary of how many networks are learned from each successor/feasible successor and output it into a CSV file. It will also give a detailed breakdown of every network that was learned from a particular successor

Parameters

- **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)
- **ask_vrf** (*bool*) – A boolean that specifies if we should prompt for which VRF. The default is true, but when this module is called from other scripts, we may want avoid prompting and supply the VRF with the "vrf" input.
- **vrf** (*str*) – The VRF that we should get the route table from. This is used only when ask_vrf is False.

s_eigrp_topology_summary.**process_topology**(*topology_list*)
Invert the topology table so that we have a list of all the networks learned from a particular successor.

Parameters **topology_list** – <list> The EIGRP topology output from TextFSM

Returns

8.1.13 s_eigrp_topology_to_csv

s_eigrp_topology_to_csv.**script_main**(*session*, *ask_vrf=True*, *vrf=None*)

SINGLE device script
Author: Jamie Caesar

Email: jcaesar@presidio.com

This script will grab the EIGRP topology table from a Cisco IOS or NXOS device and export it as a CSV file.

Parameters

- **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)
- **ask_vrf** (*bool*) – A boolean that specifies if we should prompt for which VRF. The default is true, but when this module is called from other scripts, we may want avoid prompting and supply the VRF with the "vrf" input.
- **vrf** (*str*) – The VRF that we should get the route table from. This is used only when ask_vrf is False.

8.1.14 s_interface_stats

s_interface_stats.**script_main**(*session*)

SINGLE device script
Author: Jamie Caesar
Email: jcaesar@presidio.com

This script will scrape some stats (packets, rate, errors) from all the UP interfaces on the device and put it into a CSV file.

Parameters **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)

8.1.15 s_mac_to_csv

s_mac_to_csv.**script_main**(*session*)

SINGLE device script
Author: Jamie Caesar
Email: jcaesar@presidio.com

This script will grab the MAC address table from a Cisco IOS or NX-OS device and export it to a CSV file.

Parameters **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)

8.1.16 s_nexthop_summary

s_nexthop_summary.**script_main**(*session*, *ask_vrf=True*, *vrf=None*)

SINGLE device script
Author: Jamie Caesar
Email: jcaesar@presidio.com

This script will grab the route table information from a Cisco IOS or NXOS device and export details about each next-hop address (how many routes and from which protocol) into a CSV file. It will also list all connected networks and give a detailed breakdown of every route that goes to each next-hop.

Parameters

- **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)
- **ask_vrf** (*bool*) – A boolean that specifies if we should prompt for which VRF. The default is true, but when this module is called from other scripts, we may want avoid prompting and supply the VRF with the "vrf" input.
- **vrf** (*str*) – The VRF that we should get the route table from. This is used only when ask_vrf is False.

s_nexthop_summary.**update_empty_interfaces**(*route_table*)

Takes the routes table as a list of dictionaries (with dict key names used in parse_routes function) and does recursive lookups to find the outgoing interface for those entries in the route-table where the outgoing interface isn't listed.

Parameters **route_table** (*list of dict*) – Route table information as a list of dictionaries (output from TextFSM)

Returns The updated route_table object with outbound interfaces filled in.

Return type list of dict

s_nexthop_summary.**parse_routes**(*fsm_routes*)

This function will take the TextFSM parsed route-table from the *textfsm_parse_to_dict* function. Each dictionary in the TextFSM output represents a route entry. Each of these dictionaries will be updated to convert IP addresses into ip_address or ip_network objects (from the ipaddress.py module). Some key names will also be updated also.

Parameters **fsm_routes** (*list of dict*) – TextFSM output from the *textfsm_parse_to_dict* function.

Returns An updated list of dictionaries that replaces IP address strings with objects from the ipaddress.py module

Return type list of dict

s_nexthop_summary.**nexthop_summary**(*textfsm_dict*)

A function that builds a CSV output (list of lists) that displays the summary information after analyzing the input route table.

Parameters **textfsm_dict** (*list of dict*) – The route table information in list of dictionaries format.

Returns The nexthop summary information in a format that can be easily written to a CSV file.

Return type list of lists

8.1.17 s_save_output

s_save_output.**script_main**(*session*)

SINGLE device script

Author: Jamie Caesar

Email: jcaesar@presidio.com

This script will prompt the user for a command for a Cisco device and save the output into a file. The path where the file is saved is specified in settings.ini file. This script assumes that you are already connected to the device before running it.

Parameters **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)

8.1.18 s_save_running

s_save_running.**script_main**(*session*)

SINGLE device script

Author: Jamie Caesar

Email: jcaesar@presidio.com

This script will grab the running configuration of a Cisco IOS, NX-OS or ASA device and save it into a file. The path where the file is saved is specified in settings.ini file. This script assumes that you are already connected to the device before running it.

Parameters **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)

8.1.19 s_switchport_mapping

s_switchport_mapping.**script_main**(*session*)

SINGLE device script

Author: Jamie Caesar

Email: jcaesar@presidio.com

This script will first prompt for the location of the ARP table to use for processing. Next, the mac address table of the active session will be captured and a CSV file showing the MAC address and IP of what is attached to each port on the device will be output.

Parameters **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)

s_switchport_mapping.**get_int_status**(*session*)

A function that captures the "show interface status" command and returns the processed output from TextFSM

Parameters **session** (sessions.Session) – The script object that represents this script being executed

Returns TextFSM output from processing the "show interface status" command

Return type list of list

s_switchport_mapping.**get_mac_table**(*session*)

A function that captures the mac address table and returns an output dictionary that can be used to look up the MAC address and VLAN associated with an interface.

Parameters **session** (*session.Session*) – The script object that represents this script being executed

Returns A dictionary that allows lookups of MAC and VLAN information for interfaces

Return type dict

s_switchport_mapping.**get_desc_table**(*session*)

A function that creates a lookup dictionary that can be used to get the description of an interface.

Parameters **session** (sessions.Session) – The script object that represents this script being executed

Returns A dictionary that allows getting the description of an interface by using the interface as the key.

Return type dict

s_switchport_mapping.**get_arp_info**(*script*)

A function that reads in the "show ip arp" CSV file that should be taken from the default gateway device for the switch being port mapped, so we can fill in the correct IP addresses for each device.

Parameters **script** (scripts.Script) – The script object that represents this script being executed

Returns A dictionary that can be used to lookup both the MAC and IP associated with an interface, or the IP and VLAN associated with a MAC address.

Return type dict

s_switchport_mapping.**mac_to_vendor**(*mac_lookup_table*, *mac*)

Lookup MAC Vendor Info

Parameters **mac** – MAC address to Lookup Vendor Info on

Returns MAC Vendor

8.1.20 s_update_dhcp_relay

s_update_dhcp_relay.**script_main**(*session*)

SINGLE device script

Author: Jamie Caesar

Email: jcaesar@presidio.com

This script will scan the running configuration of the connected device looking for instances of old IP helper/DHCP relay addresses (IOS/NXOS) on interfaces and if found will update the helper/relay addresses with the newer ones. The new and old addresses that the script looks for is saved in the settings.ini file, as documented below.

This script will prompt you to run in "Check Mode", where the configuration changes the script would be pushed to the device are ONLY written to a file and NO CHANGES will be made to the device. If you select "No" when prompted this script will push the configuration changes to the device. Also, when the changes are pushed to the device this script will save the running config before and after the changes are made, and will also output a log of the configuration session showing all the commands pushed.

Script Settings (found in settings/settings.ini):

- **show_instructions** - When True, displays a pop-up upon launching the script explaining where to modify the list of commands sent to devices. This window also prompts the user if they want to continue seeing this message. If not, the script changes this setting to False.
- **old_relays** - This is a comma separated list of IP addresses that the script should search for as relay addresses in the device's configuration.
- **new_relays** - This is a comma separated list of IP addresses that are the new relay addresses that should be added to any interface that has at least one of the old helper/relay addresses on it.

- **remove_old_relays** - If True, the script will add the new relays and REMOVE the old relays immediately after adding the new ones. If False (default), the script will only add the new relays to interfaces where at least one old relay is found. This is useful when you want to push out new relays as part of a migration process without removing the old relays. Since this script will not try to push new relay addresses that already exist on an interface, the script can be run again with this option set to True to later remove the old relays.

Parameters **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)

8.1.21 s_update_interface_desc

s_update_interface_desc.**script_main**(*session*, *prompt_check_mode=True*, *check_mode=True*, *enable_pass=None*)

SINGLE device script
Author: Jamie Caesar
Email: jcaesar@presidio.com

This script will grab the detailed CDP information from a Cisco IOS or NX-OS device and port-channel information and generate the commands to update interface descriptions. The user will be prompted to run in "Check Mode" which will write the configuration changes to a file (for verification or later manual application). If not, then the script will push the configuration commands to the device and save the configuration.

Script Settings (found in settings/settings.ini):

- **strip_domains** - A list of domain names that will be stripped away if found in the CDP remote device name.
- **take_backups** - If True, the script will save a copy of the running config before and after making changes.
- **rollback_file** - If True, the script will generate a rollback configuration script and save it to a file.

Parameters

- **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)
- **prompt_check_mode** (*bool*) – A boolean that specifies if we should prompt the user to find out if we should run in "check mode". We would make this False if we were using this function in a multi-device script, so that the process can run continually without prompting the user at each device.
- **check_mode** (*bool*) – A boolean to specify whether we should run in "check mode" (Generate what the script would do only – does not push config), or not (Pushes the changes to the device). The default is True for safety reasons, and this option will be overwritten unless prompt_checkmode is False.
- **enable_pass** (*str*) – The enable password for the device. Will be passed to start_cisco_session method if available.

s_update_interface_desc.**extract_cdp_data**(*cdp_table*)
Extract only remote host and interface for each local interface in the CDP table

Parameters **cdp_table** (*list of list*) – The TextFSM output for CDP neighbor detail

Returns A dictionary for each local interface with corresponding remote host and interface.

Return type dict

s_update_interface_desc.**add_port_channels**(*desc_data*, *pc_data*)

Adds port-channel information to our CDP data so that we can also put descriptions on port-channel interfaces that have members found in the CDP table.

Parameters

- **desc_data** – Our CDP description data that needs to be updated
- **pc_data** –

Type pc_data:

8.1.22 s_vlan_to_csv

s_vlan_to_csv.**script_main**(*session*)

SINGLE device script

Author: Jamie Caesar

Email: jcaesar@presidio.com

This script will output the VLAN database to a CSV file.

One possibly use of this script is to take the .CSV outputs from 2 or more devices, paste them into a single XLS file and use Excel to highlight duplicate values, so VLAN overlaps can be discovered prior to connecting switches together via direct link, OTV, etc. This could also be used to find missing VLANs between 2 large tables that should have the same VLANs.

Parameters **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)

s_vlan_to_csv.**normalize_port_list**(*vlan_data*)

When TextFSM processes a VLAN with a long list of ports, each line will be a separate item in the resulting list. This fuction combines all of those entries into a single string that contains all of the ports in a comma-separated list.

Parameters **vlan_data** – The VLAN data from TextFSM that will be modified in-place

8.2 Multiple Device Scripts

Multiple Device Scripts are scripts that are designed to be executed while NOT attached to a remote device (You will get an error if you try to launch a multi-device script from a connected tab). The script will prompt for an input CSV containing all of the hosts and other required details (credentials, jumpbox, etc) to be able to connect all of the devices in the list automatically. The logic of the script will be performed on each device, one at a time.

8.2.1 Device Import CSV File

The device list CSV has required columns that must be included. These are the Hostname, Protocol, Username columns (case sensitive). If these columns are missing from your CSV file an error will be returned, because these are the minimum pieces of information required to connect to a device. **A sample device input CSV file is located at 'templates/sample_device_list.csv'**

Additional columns are allowed to be in the CSV file and will be accessible from the scripts, although the data will never be used if a script is not written to look for a particular column. For example, the "m_document_device" script will use a column called "Command List" to override which list of commands (in the settings.ini file) are used when documenting that particular device. The same device list can be used in other multi-device scripts, but they will not try to access that column.

Some standard columns that will be added with default vaules if missings are "Password", "Enable" and "Proxy Session":

- The "Password" field lets you override the password used with that device. Generally this is left blank because the script will prompt for a password when one isn't defined so that you aren't required to save a password in the file. This field may still be useful when a particular device has a different password for the same username.
- The "Enable" column will override the default enable password that the script uses (based on prompting the user).
- The "Proxy Session" column can override the default SecureCRT session to use as a proxy to reach the target device. The default SecureCRT session is in the settings.ini file (as well as a flag on whether to use a proxy or not).

8.2.2 Available Scripts

Below is a list of multi-device scripts available with this release of SecureCRT Tools.

8.2.2.1 m_add_global_config

m_add_global_config.**script_main**(*script*)

MULTIPLE device script
Author: Jamie Caesar
Email: jcaesar@presidio.com

This script will add global configuration commands to the provided list of devices. The commands sent will depend on the operating system of each connected device. For example, IOS devices get the commands listed in the 'ios' section of the settings for this script. If the device is running NX-OS, it will get the commands from the 'nxos' section of the settings, etc.

Any devices that cannot be connected to will be logged in a separate file saved in the output directory.

This script will prompt you to run in "Check Mode", where the configuration changes the script would be pushed to the devices are ONLY written to a file and NO CHANGES will be made to the devices. If you select "No" when prompted this script will push the configuration changes to the devices. Also, when the changes are pushed to the devices this script will save the running config before and after the changes are made, and will also output a log of the configuration sessions showing all the commands pushed.

Script Settings (found in settings/settings.ini):

- **show_instructions** - When True, displays a pop-up upon launching the script explaining where to modify the list of commands sent to devices. This window also prompts the user if they want to continue seeing this message. If not, the script changes this setting to False.
- **ios** - A comma separated list of commands that will be sent to IOS devices.
- **ios-xr** - A comma separated list of commands that will be sent to IOS-XR devices.
- **nxos** - A comma separated list of commands that will be sent to NX-OS devices.

- **asa** - A comma separated list of commands that will be sent to ASA devices.

Parameters **script** (scripts.Script) – A subclass of the scripts.Script object that represents the execution of this particular script (either CRTScript or DirectScript)

m_add_global_config.**per_device_work**(*session*, *check_mode*, *enable_pass*, *settings_header*)

This function contains the code that should be executed on each device that this script connects to. It is called after establishing a connection to each device in the loop above.

You can either put your own code here, or if there is a single-device version of a script that performs the correct task, it can be imported and called here, essentially making this script connect to all the devices in the chosen CSV file and then running a single-device script on each of them.

8.2.2.2 m_cdp_to_csv

m_cdp_to_csv.**script_main**(*script*)

MULTIPLE device script

Author: Jamie Caesar

Email: jcaesar@presidio.com

This script will grab the detailed CDP information from each Cisco IOS or NX-OS device in the provided device list CSV file and export each to a CSV file containing the important information, such as Remote Device hostname, model and IP information, in addition to the local and remote interfaces that connect the devices.

Script Settings (found in settings/settings.ini):

- **strip_domains** - A list of domain names that will be stripped away if found in the CDP remote device name.

Parameters **script** (scripts.Script) – A subclass of the scripts.Script object that represents the execution of this particular script (either CRTScript or DirectScript)

m_cdp_to_csv.**per_device_work**(*session*, *enable_pass*)

This function contains the code that should be executed on each device that this script connects to. It is called after establishing a connection to each device in the loop above.

You can either put your own code here, or if there is a single-device version of a script that performs the correct task, it can be imported and called here, essentially making this script connect to all the devices in the chosen CSV file and then running a single-device script on each of them.

8.2.2.3 m_document_device

m_document_device.**script_main**(*script*)

MULTIPLE device script

Author: Jamie Caesar

Email: jcaesar@presidio.com

This script will grab the output for a list of commands from the provided list of devices. The list of commands is taken from the 'settings/settings.ini' file. There is a separate list for each supported network operating system (IOS, NXOS and ASA) and by default the list that matches the network operating system of the connected device will be used.

Custom lists of commands are supported. These lists can be added manually to the [document_device] section of the 'settings/settings.ini' file. To be able to choose one of these lists when running the script, the 'prompt_for_custom_lists' setting needs to be changed to 'True' in the settings.ini file. Once this option is enabled, the script will prompt for the name of the list that you want to use. If the input is left blank then the default behavior (based on network OS) will choose the list.

NOTE: The Custom list can be set on a PER DEVICE basis if a column names "Command List" (case sensitive) is added to the device list CSV file that is selected when running this script. If the "Command List" column is missing or the field is left blank for a device then the list will be chosen using the default behavior (i.e. use the list specified when running the script, or based on the network OS of each device).

Script Settings (found in settings/settings.ini):

- **show_instructions** - When True, displays a pop-up upon launching the script explaining where to modify the list of commands sent to devices. This window also prompts the user if they want to continue seeing this message. If not, the script changes this setting to False.
- **folder_per_device** - If True, Creates a folder for each device, based on the hostname, and saves all files inside that folder WITHOUT the hostname in the output file names. If False, it saves all the files directly into the output folder from the global settings and includes the hostname in each individual filename.
- **prompt_for_custom_lists** - When set to True, the script will prompt the user to type the name of a list of commands to use with the connected device. This list name must be found as an option in the [document_device] section of the settings.ini file. The format is the same as the default network OS lists, 'ios', 'nxos', etc.
- **ios** - The list of commands that will be run on IOS devices
- **nxos** - The list of commands that will be run on NXOS devices
- **asa** - The list of commands that will be run on ASA devices

Any additional options found in this section would be custom added by the user and are expected to be lists of commands for use with the 'prompt_for_custom_lists' setting.

By default, The outputs will be saved in a folder named after the hostname of the device, with each output file being saved inside that directory. This behavior can be changed in the settings above.

Parameters **session** (sessions.Session) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)

m_document_device.**per_device_work**(*session*, *enable_pass*, *command_list_name*, *folder_per_device*)

This function contains the code that should be executed on each device that this script connects to. It is called after establishing a connection to each device in the loop above.

This function simply calls the imported "document()" function from the s_document_device script on each device connected to.

8.2.2.4 m_find_macs_by_vlans

m_find_macs_by_vlans.**script_main**(*script*)

MULTIPLE device script
Author: Jamie Caesar

Email: jcaesar@presidio.com

This script will provide a list of all the switches that have locally connected MAC addresses in their mac address table for a range of VLANs.

After launching the script, it will prompt for a CSV file with all of the devices the script should connect to. It will also prompt for a range of VLANs that it should look for MAC addresses This script checks that it will NOT be run in a connected tab.

Parameters **script** (scripts.Script) – A subclass of the scripts.Script object that represents the execution of this particular script (either CRTScript or DirectScript)

m_find_macs_by_vlans.**per_device_work**(*session*, *enable_pass*, *vlan_set*)

This function contains the code that should be executed on each device that this script connects to. It is called after establishing a connection to each device in the loop above.

You can either put your own code here, or if there is a single-device version of a script that performs the correct task, it can be imported and called here, essentially making this script connect to all the devices in the chosen CSV file and then running a single-device script on each of them.

8.2.2.5 m_inventory_report

m_inventory_report.**get_manufacture_date**(*serial*)

A function that will decode the manufacture date of a device from its serial number.

Parameters **serial** (*str*) – The serial number of a Cisco device, which should be 11 digits long.

Returns The month and year the device was manufactured in string format. (e.g. "September 2010")

m_inventory_report.**script_main**(*script*)

MULTIPLE device script

Author: Jamie Caesar

Email: jcaesar@presidio.com

This script will connect to all devices in the provided CSV file and create an output report (also CSV format) containing inventory data about the devices, such as hostname, model number, code version, serial number, manufacture date, etc.

This script checks that it will NOT be run in a connected tab.

Parameters **script** (scripts.Script) – A subclass of the scripts.Script object that represents the execution of this particular script (either CRTScript or DirectScript)

m_inventory_report.**per_device_work**(*session*, *enable_pass*)

This function contains the code that should be executed on each device that this script connects to. It is called after establishing a connection to each device in the loop above.

You can either put your own code here, or if there is a single-device version of a script that performs the correct task, it can be imported and called here, essentially making this script connect to all the devices in the chosen CSV file and then running a single-device script on each of them.

8.2.2.6 m_merged_arp_to_csv

m_merged_arp_to_csv.**script_main**(*script*)

MULTIPLE device script

Author: Jamie Caesar

Email: jcaesar@presidio.com

This script will pull the ARP tables from multiple devices and combine their data into a single ARP CSV file.

The main intention for this script is to be used with the 's_switchport_mapping' script, which will prompt for an ARP table CSV file to list IPs connected to switch ports. This script is intended to speed up the collection of ARP data in cases such as:

1) There are 2 core switches running a first hop redundancy protocol (HSRP, GLBP, etc) and both devices are actively passing traffic so we need both ARP tables

2) The switch has VLANs that are being used in multiple upstream VRFs (not all with SVIs on the same devices) so the ARP tables from many devices may be needed to fully map the switch.

NOTE: Since this script merges ARP tables of multiple devices which may have duplicate entries, the interface is NOT written to the output file like it is with the single device version of this script.

This script checks that it will NOT be run in a connected tab. This script initiates the connection to all devices based on the input of the device CSV file that the script requests.

Parameters **script** (scripts.Script) – A subclass of the scripts.Script object that represents the execution of this particular script (either CRTScript or DirectScript)

m_merged_arp_to_csv.**per_device_work**(*session*, *selected_vrf*, *add_header*)

This function contains the code that should be executed on each device that this script connects to. It is called after establishing a connection to each device in the loop above.

This function gathers the ARP table information and returns it (in list format) to the calling program.

8.2.2.7 m_save_output

m_save_output.**script_main**(*script*)

MULTIPLE device script

Author: Jamie Caesar

Email: jcaesar@presidio.com

This script will prompt for a CSV list of devices, then will prompt for a command to run on each device in the list. The output from each device will be saved to a file. The path where the file is saved is specified in the settings.ini file.

Parameters **script** (scripts.Script) – A subclass of the scripts.Script object that represents the execution of this particular script (either CRTScript or DirectScript)

m_save_output.**per_device_work**(*session*, *enable_pass*, *send_cmd*)

This function contains the code that should be executed on each device that this script connects to. It is called after establishing a connection to each device in the loop above.

You can either put your own code here, or if there is a single-device version of a script that performs the correct task, it can be imported and called here, essentially making this script connect to all the devices in the chosen CSV file and then running a single-device script on each of them.

8.2.2.8 m_update_dhcp_relay

m_update_dhcp_relay.**script_main**(*script*)

MULTIPLE device script

Author: Jamie Caesar

Email: jcaesar@presidio.com

This script will scan the running configuration of the provided list of devices, looking for instances of old IP helper/DHCP relay addresses (IOS/NXOS) on interfaces and if found will update the helper/relay addresses with the newer ones. The new and old addresses that the script looks for is saved in the settings.ini file, as documented below.

Any devices that cannot be connected to will be logged in a separate file saved in the output directory.

This script will prompt you to run in "Check Mode", where the configuration changes the script would be pushed to the devices are ONLY written to a file and NO CHANGES will be made to the devices. If you select "No" when prompted this script will push the configuration changes to the devices. Also, when the changes are pushed to the devices this script will save the running config before and after the changes are made, and will also output a log of the configuration sessions showing all the commands pushed.

Script Settings (found in settings/settings.ini):

- **show_instructions** - When True, displays a pop-up upon launching the script explaining where to modify the list of commands sent to devices. This window also prompts the user if they want to continue seeing this message. If not, the script changes this setting to False.
- **old_relays** - This is a comma separated list of IP addresses that the script should search for as relay addresses in the device's configuration.
- **new_relays** - This is a comma separated list of IP addresses that are the new relay addresses that should be added to any interface that has at least one of the old helper/relay addresses on it.
- **remove_old_relays** - If True, the script will add the new relays and REMOVE the old relays immediately after adding the new ones. If False (default), the script will only add the new relays to interfaces where at least one old relay is found. This is useful when you want to push out new relays as part of a migration process without removing the old relays. Since this script will not try to push new relay addresses that already exist on an interface, the script can be run again with this option set to True to later remove the old relays.

Parameters **script** (scripts.Script) – A subclass of the scripts.Script object that represents the execution of this particular script (either CRTScript or DirectScript)

m_update_dhcp_relay.**per_device_work**(*session*, *check_mode*, *enable_pass*, *old_helpers*, *new_helpers*, *remove_old_helpers*)

This function contains the code that should be executed on each device that this script connects to. It is called after establishing a connection to each device in the loop above.

You can either put your own code here, or if there is a single-device version of a script that performs the correct task, it can be imported and called here, essentially making this script connect to all the devices in the chosen CSV file and then running a single-device script on each of them.

8.2.2.9 m_update_interface_desc

m_update_interface_desc.**script_main**(*script*)

MULTIPLE device script
Author: Jamie Caesar

Email: jcaesar@presidio.com

This script will grab the detailed CDP information from a Cisco IOS or NX-OS device and port-channel information and generate the commands to update interface descriptions. The user will be prompted to run in "Check Mode" which will write the configuration changes to a file (for verification or later manual application). If not, then the script will push the configuration commands to the device and save the configuration.

IMPORTANT: This script imports the script_main() function from the s_update_interface_desc.py to run a majority of the script logic. Much of this script is only handling multiple logins and calling the single-device version of this script.

Script Settings (found in settings/settings.ini):

- **strip_domains** - A list of domain names that will be stripped away if found in the CDP remote device name.
- **take_backups** - If True, the script will save a copy of the running config before and after making changes.
- **rollback_file** - If True, the script will generate a rollback configuration script and save it to a file.

Parameters **script** (scripts.Script) – A subclass of the scripts.Script object that represents the execution of this particular script (either CRTScript or DirectScript)

m_update_interface_desc.**per_device_work**(*session*, *check_mode*, *enable_pass*)

This function contains the code that should be executed on each device that this script connects to. It is called after establishing a connection to each device in the loop above.

You can either put your own code here, or if there is a single-device version of a script that performs the correct task, it can be imported and called here, essentially making this script connect to all the devices in the chosen CSV file and then running a single-device script on each of them.

8.3 No Device Scripts

These scripts only interact with the SecureCRT application when they are run and do NOT attempt to connect to any devices nor interact with any existing sessions that may be open.

8.3.1 import_sessions_from_csv

import_sessions_from_csv.**script_main**(*script*)

Author: Michael Ethridge Email: michael@methridge.com

This script will import a list of sessions to create in SecureCRT from a CSV file. It does not connect to any devices.

Parameters **script** (scripts.Script) – A subclass of the sessions.Session object that represents this particular script session (either SecureCRTSession or DirectSession)

CHAPTER 9

Writing Your Own Scripts

9.1 Introduction To Writing SecureCRT Scripts

For those who have a need to either modify the existing scripts or write completely new ones for specific tasks (I expect that will be the majority), this section contains documentation on the classes used to simplify script creation.

The purpose of building the below classes is to encapsulate many of the common tasks that will be performed on any device so that those methods can simply be called, instead of requiring each person to either copy and paste a lot of blocks of code, or understand the intricate details of interacting with a remote device through SecureCRT. While that may be required in certain cases, the goal is that a vast majority of the interaction can be handled using the existing methods documented below.

9.1.1 Script Templates

When creating a new script it is **highly recommended** to look at the *templates/* directory in this repository, which contains boilerplate starting points for new scripts. The sections where code is meant to be added has been marked with comments saying "ADD YOUR CODE HERE". Unless you have a good reason not to, it is recommended to start with these templates because they already handle a few extra steps required to use the full functionality of the securecrt_tools module. For example, some of the things handled by the starter template are: * Since SecureCRT uses its own Python interpreter that is bundled with the application, some special logic had to be added to support importing both securecrt_tools and the third porty modules used in some scripts. * The logging facility is initialized * Some code to make the script run either when run from SecureCRT or in a debug (simulation) mode when run directly with your local Python installation/IDE. This is explained further below.

9.1.2 Module Types

There are 2 categories of modules shown below: 1) The modules written to handle the interaction between these scripts and the remote devices (via SecureCRT), and 2) 3rd Party modules that are used within some of the scripts to help process our device outputs better.

9.1.2.1 SecureCRT Tools Modules

The code used to interact with SecureCRT is kept in the *securecrt_tools* module folder. This folder contains both the securecrt_tools and 3rd party modules in addition to a small number of other files. The files that are a part of securecrt_tools are: * scripts.py * sessions.py * utilities.py * settings.py * message_box_const.py

These files break up the code into groupings based on function so that the repository has a better organizational structure. The purpose of the files is explained below, while the documentation of all methods/functions defined in these files is in a later section.

scripts.py - This file contains the code that interacts with the SecureCRT application itself. It has the code for doing things like creating message boxes, file selector windows, importing a CSV with a list of devices that should each be connected to, etc. Anything that requires interaction with the SecureCRT API but isn't related to a specific connection to a remote device is kept here.

sessions.py - This file contains the code that interacts with a device via SecureCRT. There are methods in this file for connecting to devices, finding the prompt of a device, sending data to a remote device, capturing command output to a variable, writing the output of a command to a file, disconnecting from a device, etc.

utilities.py - This file contains utility functions that are independent of SecureCRT and so are kept in a separate file. This file contains functions that do things such as parsing TextFSM output into either a list or dictionary format, writing a 2-dimensional list to a CSV file, sorting text in "human" format ('1,2,10,11' instead of '1,10,11,2'), etc.

settings.py - This file contains the code used to handle the settings.ini file. Functions in this file perform tasks such as reading settings from the INI file, creating the file if it exists, adding missing variables that are required for operation of these scripts, etc.

message_box_const.py - This file, while related to SecureCRT, holds information used in the formatting of message boxes and prompts. These SecureCRT constructs accept an optional number in the function call that dictates the layout of the box (what buttons are available, what icon is in the box, etc). There is a specific number assigned to each option and I have created a list of constants that are assigned the appropriate number. This allows you define the format of the message box by adding together the constants you want into an appropriate value. An example of how to use these constants is in the comments of the file.

9.1.2.2 Debug (Simulation Mode)

The debug mode was added to speed development and troubleshooting of script logic that doesn't involve SecureCRT's API, such as testing/debugging parsing of the output from a device. When the script is run directly from your local Python 2.x install, then instead of trying to send (for example) *show cdp neighbor* to a device, it will prompt you to specify a file that contains the output for *show cdp neighbor*, so that you can step through the program logic with a debugging tool. When trying to debug from within SecureCRT you are limited to either logging or printing debug info to the screen with a Message Box (which are both available if you prefer)

This is done by using the class system in Python. For those who understand Object-Oriented Programming and Classes work, both the Script and Session objects have an abstract base class (e.g. *class Session*) as well as a sub-class for use with SecureCRT (e.g. *class CRTSession(Session)*) and a subclass for use with your local Python install (e.g. *class DebugSession(Session)*). There is code at the end of all the provided scripts that will use the *CRTSession* sub-class if the script is called from SecureCRT and use the *DebugSession* sub-class when the script is run from the local Python install. This allows us to simulate running the script against a device from our local Python installation, generally by either prompting the user for a file input when the script would gather output from a remote device, or printing to the console instead of creating message boxes on the screen.

For those less familiar with OOP and Classes, the *Session* abstract base class defines what methods/functions are required to be defined for any class that is a sub-class of *Session*. For example, the *get_command_output()* is defined under the *Session* base class and required to be implemented in both the *CRTSession* class and the *DebugSession* class. In the *Session* base class there **is not any code** for that method. It is simply defined with nothing but a *pass* statement in it. Each sub-class implements this method differently, though. The *CRTSession* class is meant to be used when

SecureCRT is launching the script, so it has code that will send the provided command to the device and capture the output returned and save it into a variable. The *DebugSession* class is meant to be used when the script is run by the local Python interpreter and so we do not have the ability to actually connect to a remote device. Instead, the *DebugSession* class prompts the user for a filename on the console, and then opens that file and saves the contacts into the variable. In this way, we are simulating a connection to a remote device and can now test our parsing logic on that output without needing to connect to the remote device (except to initially capture a real output to a file to test against).

Because the script file has the code (if you used the provided templates, at least) that detects which program is executing it (SecureCRT or Python directly) and selects the appropriate class type to use, you can write your scripts once with the same method names (e.g. *get_command_output()* and it will execute differently but appropriately for the way it is being executed.

9.1.2.3 3rd Party Modules

The 3rd Party modules are included because someone has already done the work to create modules that perform specific funtions, so there is no reason we shouldn't take advantage of that instead of writing our own (and probably more buggy) implementations. For better or worse, SecureCRT includes it's own Python environment within the application, which means we cannot install new modules like we can for a local installation of Python. Instead we have to include the source code in this repository for the 3rd party modules so we can access them. The most commonly used 3rd party modules in these scripts are: * *TextFSM* - A module that lets you build a template using regular expression to parse semi-structured (i.e. meant for human readability) outputs to collect only the information you want. * *ipaddress module* - A module that lets you create IP address and IP network objects and contains many methods to comparing and displaying IP information.

9.2 A Note on TextFSM

TextFSM is a module written to simply the process of extracting information out of semi-structured outputs (such as CLI command output) that are meant for human readability. TextFSM uses a template file to define what values we are looking for, as well as how those values get extracted from the output. You can read some examples of using the TextFSM Wiki and the Code Lab section for additional examples.

In addition, there is a large repository of TextFSM templates located in Network To Code's Github Repository. This repository contains TextFSM templates for a variety of vendors and to process a large number of commands. At best, they'll have the template you need that is already created, but in some cases you may need to modify the template to get what you want from it.

In the end, TextFSM still boils down to text matching with *Regular Expressions* (RegEx), although TextFSM puts some structure around it that makes it much easier than trying to write matching by-hand – especially for more complicated outputs like a routing table. If you are unfamiliar with RegEx, I'm sure there are plenty of primers on how they work online. I would highly recommend that if you are trying to develop some new regular expressions or a TextFSM template that you use a site such as regex101.com to see real-time feedback of what your expressions will match against some sample data you've pasted into the site.

9.3 Module Documentation

9.3.1 SecureCRT Tools Modules

The following custom modules include the majority of the functionality that is used in the single and multi-device scripts. The classes described below are designed to handle the common actions that scripts would need into a single method to avoid the need to copy and paste chunks of code into multiple scripts, and to make writing scripts faster and easier than when you have to use the low-level SecureCRT API to interact with devices.

It is important to understand the relationship between the classes listed below.

The classes in the "scripts" module and used to represent the execution of the script itself and that script's interactions with the calling application (primarily SecureCRT). Since a SecureCRT script has the ability to open multiple tabs to different devices and interact with all of them, any attribute or method that is common to the script and not specific to each open session is defined in the script class. This class tracks a reference to the main session object, which represents the SecureCRT tab that the script was launched from.

The "sessions" classes represent a session to a remote device, which will usually be the SecureCRT tab that the script was initially launched from. The Script class has the ability to open a new connection in a new tab, and it will return a reference to a separate Session object so that the script can interact with each session independently. **NOTE:** SecureCRT does **NOT** support multi-threading so the use of multiple sessions are best used in cases where you do not want to close an existing session before performing actions on another device. One example can be validating login to a remote device after making AAA changes without disconnecting the original session and potentially locking ourselves out. A rollback of the changes could be performed if the second session is unable to log into the device.

The "settings" classes are simply used to manage importing, exporting and updating entries in the settings.ini file. The script object keeps a reference to the settings class so that settings can be retrieved or changed. This class will also attempt to update/re-write the local settings.ini file (while preserving existing settings) should the default_settings.ini contains settings that do not exist in the local settings file. This may happen when a new script is created and needs settings available in the settings.ini file.

The "utilities" class contains a bunch of helper functions that can be used to simplify certain common tasks. These funtions are pure Python which means they do not require interactions with scripts, sessions or settings in any way. For example, some of these functions will convert a string containing an interface name to the short version (Gi0/0) or the long version (GigabitEthernet0/0), while others perform action like sorting in "human" order (device1, device2, device10) instead of alphanumeric (device1, device10, device2). Functions of this sort should all be saved in this file when they need to be used by multiple scripts.

9.3.1.1 securecrt_tools.scripts

This module contains classes for representing the execution of a script in SecureCRT. The attributes and methods defined with these classes are more "global" in nature, meaning that they focus on either the interaction with the application, or anything that is common to the entire script regardless of how many sessions (in tabs) are open to remote devices.

The Script Base Class

class securecrt_tools.scripts.**Script**(*script_path*)

This is a base class for the script object. This class cannot be used directly, but is instead a blueprint that enforces what any sub-classes must implement. The reason for using this design (base class with sub-classes) is to allow the script to be run in different contexts without needing to change the code, as long as the correct sub-class is being used.

For example, the most important sub-class is the CRTScript subclass which is used when the script is executed from SecureCRT. This class is written to interact with SecureCRT's Python API to be able to control the applications. If the script author wants to display something to the user, they can use the message_box() method to use SecureCRT's pop-up message box (crt.Dialog.MessageBox() call). The other sub-class (currently) is the DebugScript sub-class, which was created to allow easier debugging of a script's logic by letting you execute the script using a local python installation – ideally in your IDE of choice. This would allow you to use the fully debugging features of the IDE which are otherwise not available when executing a script inside SecureCRT. When the message_box() is called on the DebugScript sub-class, the message will be printed to the console.

This sub-class design can also allow for additional classes to be created in the future – perhaps one that uses Netmiko to connect to the remote devices. In this way, if a Netmiko sub-class was created, then all of the

same scripts can be executed without needing to change them, because the Netmiko class would be required to implement all of the same methods that are defined in the base class (just like CRTScript and DebugScript)

DebugScript class is to allow the programmer to debug their code in their favorite IDE or debugger, which cannot be done when executing the script from SecureCRT (in which case you are forced to either use debug messages or write outputs to a messagebox. DebugScript allows the same code to run locally without SecureCRT and the class will prompt for the information it needs to continue running the main script.

Any methods that are not prepended with the @abstractmethod tag preceding the method definition will be inherited and available to the sub-classes without needing to define them specifically in each sub-class. Methods designed this way would use the exact same code in all sub-classes, and so there is no reason to re-create them in each subclass.

Methods defined with the @abstractmethod tag should be left empty in this class. They are required to be implemented in each sub-class. Methods are defined this way when they are required to exist in all sub-classes for consistency, but the code would be written completely different depending on which class is being used. One example is the message_box method below. Under the CRTScript class, this method uses the SecureCRT API to print messages and format the text box that should pop up to the user, but in the DebugScript class this method only prints the message to the console. In this way, a call to this method will work either way the script is called as long as the correct Script sub-class is being used (and the template are already written to do this).

get_main_session()
Returns a CRTSession object that interacts with the SecureCRT tab that the script was lauched within. This is the primary tab that will be used to interact with remote devices. While new tabs can be created to connect to other devices, SecureCRT does not support multi-threading so multiple devices cannot be interacted with simultaneously via a script.

Returns A session object that represents the tab where the script was launched

Return type *sessions.Session*

get_template(*name*)
Retrieve the full path to a TextFSM template file.

Parameters **name** (*str*) – Filename of the template

Returns Full path to the template location

Return type str

import_device_list()
This function will prompt for a device list CSV file to import, returns a list containing all of the devices that were in the CSV file and their associated credentials. The CSV file must be of the format, and include a header row of ['Hostname', 'Protocol', 'Username', 'Password', 'Enable', 'Proxy Session']. An example device list CSV file is at 'template/sample_device_list.csv'

The 'Proxy Session' options is looking for a SecureCRT session name that can be used to proxy this connection through. This sets the 'Firewall' option under a SecureCRT session to perform this connection proxy.

Some additional information about missing items from a line in the CSV: - If the hostname field is missing, the line will be skipped. - If the protocol field is empty, the script will try SSH2, then SSH1, then Telnet. - If the username is missing, this method will prompt the user for a default usernaem to use - If the password is missing, will prompt the user for a password for each username missing a password - If the enable password is missing, the method will ask the user if they want to set a default enable to use - If the IP is included then the device will be reached through the jumpbox, otherwise connect directly.

Returns A list where each entry is a dictionary representing a device and the associated login information.

Return type list of dict

validate_dir(*path*, *prompt_to_create=True*)
Verifies that the path to the supplied directory exists. If not, prompt the user to create it.

Parameters **path** (*str*) – A directory path (not including filename) to be validated

CRTScript Class

class securecrt_tools.scripts.**CRTScript**(*crt*)
Bases: *securecrt_tools.scripts.Script*

This class is a sub-class of the Script base class, and is meant to be used in any scripts that are being executed from inside of SecureCRT. This sub-class is designed to interact with the SecureCRT application itself (not with tabs that have connections to remote devices) and represent a script being executed from within SecureCRT. This class inherits the methods from the Script class that are documented above, and is required to implement all of the abstract classes defined in the Script class.

connect(*host*, *username*, *password*, *protocol=None*, *proxy=None*, *prompt_endings=('#', '>')*)
Attempts to connect to a device by any available protocol, starting with SSH2, then SSH1, then telnet

Parameters

- **host** (*str*) – The IP address of DNS name for the device to connect
- **username** (*str*) – The username to login to the device with
- **password** (*str*) – The password that goes with the provided username. If a password is not specified, the user will be prompted for one.
- **protocol** (*str*) – A string with the desired protocol (telnet, ssh1, ssh2, ssh). If left blank it will try all starting with SSH2, then SSH1 then Telnet. "ssh" means SSH2 then SSH1.
- **proxy** (*str*) – The name of a SecureCRT session object that can be used as a jumpbox to proxy the SSH connection through. This is the same as selecting a session under the "Firewall" selection under the SSH settings screen for a SecureCRT session.
- **prompt_endings** (*list*) – A list of strings that are possible prompt endings to watch for. The default is for Cisco devices (">" and "#"), but may need to be changed if connecting to another type of device (for example "$" for some linux hosts).

connect_ssh(*host*, *username*, *password*, *version=None*, *proxy=None*, *prompt_endings=('#', '>')*)
Connects to a device via the SSH protocol. By default, SSH2 will be tried first, but if it fails it will attempt to fall back to SSH1.

Parameters

- **host** (*str*) – The IP address of DNS name for the device to connect
- **username** (*str*) – The username to login to the device with
- **password** (*str*) – The password that goes with the provided username. If a password is not specified, the user will be prompted for one.
- **version** (*int*) – The SSH version to connect with (1 or 2). Default is None, which will try 2 first and fallback to 1 if that fails.
- **proxy** (*str*) – The name of a SecureCRT session object that can be used as a jumpbox to proxy the SSH connection through. This is the same as selecting a session under the "Firewall" selection under the SSH settings screen for a SecureCRT session.

- **prompt_endings** (*list*) – A list of strings that are possible prompt endings to watch for. The default is for Cisco devices (">" and "#"), but may need to be changed if connecting to another type of device (for example "$" for some linux hosts).

connect_telnet(*host*, *username*, *password*, *proxy=None*, *prompt_endings=('#', '>')*)
Connects to a device via the Telnet protocol.

Parameters

- **host** (*str*) – The IP address of DNS name for the device to connect
- **username** (*str*) – The username to login to the device with
- **password** (*str*) – The password that goes with the provided username. If a password is not specified, the user will be prompted for one.
- **proxy** (*str*) – The name of a SecureCRT session object that can be used as a jumpbox to proxy the SSH connection through. This is the same as selecting a session under the "Firewall" selection under the SSH settings screen for a SecureCRT session.
- **prompt_endings** (*list*) – A list of strings that are possible prompt endings to watch for. The default is for Cisco devices (">" and "#"), but may need to be changed if connecting to another type of device (for example "$" for some linux hosts).

create_new_saved_session(*session_name*, *ip*, *protocol='SSH2'*, *folder='_imports'*)
Creates a session object that can be opened from the Connect menu in SecureCRT.

Parameters

- **session_name** (*str*) – The name of the session
- **ip** (*str*) – The IP address or hostname of the device represented by this session
- **protocol** (*str*) – The protocol to use for this connection (TELNET, SSH1, SSH2, etc)
- **folder** (*str*) – The folder (starting from the configured Sessions folder) where this session should be saved.

disconnect(*command='exit'*)
Disconnects the main session used by the script by calling the disconnect method on the session object.

Parameters **command** (*str*) – The command to be issued to the remote device to disconnect. The default is 'exit'

file_open_dialog(*title*, *button_label='Open'*, *default_filename=''*, *file_filter=''*)
Prompts the user to select a file that will be processed by the script. In SecureCRT this will give a pop-up file selection dialog window, and will return the full path to the file chosen.

Parameters

- **title** (*str*) – Title for the File Open dialog window (Only displays in Windows)
- **button_label** (*str*) – Label for the "Open" button
- **default_filename** (*str*) – If provided a default filename, the window will open in the parent directory of the file, otherwise the current working directory will be the starting directory.
- **file_filter** (*str*) – Specifies a filter for what type of files can be selected. The format is: <Name of Filter> (.*<extension>)*|.<extension>|| For example, a filter for CSV files would be "CSV Files (.*csv)*|.csv||" or multiple filters can be used: "Text Files (.*txt)*|.txt|Log File (.*log)*|.log||"

Returns The absolute path to the file that was selected

Return type str

message_box(*message*, *title=''*, *options=0*)

Prints a message for the user. In SecureCRT, the message is displayed in a pop-up message box with a variety of buttons, depending on which options are chosen. The default is just an "OK" button.

This window can be customized by setting the "options" value, using the constants listed at the top of the sessions.py file. One constant from each of the 3 categories can be OR'd (|) together to make a single option value that will format the message box.

Parameters

- **message** (*str*) – The message to send to the user
- **title** (*str*) – Title for the message box
- **options** (*int*) – Sets the display format of the messagebox. (See Message Box constants in sessions.py)

Returns The return code that identifies which button the user pressed. (See Message Box constants)

Return type int

prompt_window(*message*, *title=''*, *hide_input=False*)

Prompts the user for an input value. In SecureCRT this will open a pop-up window where the user can input the requested information.

The "hide_input" input will mask the input, so that passwords or other senstive information can be requested.

Parameters

- **message** (*str*) – The message to send to the user
- **title** (*str*) – Title for the prompt window
- **hide_input** (*bool*) – Specifies whether to hide the user input or not. Default is False.

Returns The value entered by the user

Return type str

DebugScript Class

class securecrt_tools.scripts.**DebugScript**(*full_script_path*)

Bases: *securecrt_tools.scripts.Script*

This class is a sub-class of the Script base class, and is meant to be used in any scripts that are being executed directly from a local python installation. This sub-class is designed to simulate the interaction with SecureCRT while the script is being run from a local python installation. For example, when a script attempts to create a pop-up message box in SecureCRT, this class will simply print the information to the console (or request information from the user via the console).

This class inherits the methods from the Script class that are documented above, and is required to implement all of the abstract classes defined in the Script class. This way, it is a complete replacement for the CRTScript class if a script is run directly.

connect(*host*, *username*, *password*, *protocol=None*, *proxy=None*, *prompt_endings=('#', '>')*)

Pretends to connect to a device. Simply marks the state of the session as connected. Never fails.

Parameters

- **host** (*str*) – The IP address of DNS name for the device to connect
- **username** (*str*) – The username to login to the device with
- **password** (*str*) – The password that goes with the provided username. If a password is not specified, the user will be prompted for one.
- **protocol** (*str*) – A string with the desired protocol (telnet, ssh1, ssh2, ssh). If left blank it will try all starting with SSH2, then SSH1 then Telnet. "ssh" means SSH2 then SSH1.
- **proxy** (*str*) – The name of a SecureCRT session object that can be used as a jumpbox to proxy the SSH connection through. This is the same as selecting a session under the "Firewall" selection under the SSH settings screen for a SecureCRT session.
- **prompt_endings** (*list*) – A list of strings that are possible prompt endings to watch for. The default is for Cisco devices (">" and "#"), but may need to be changed if connecting to another type of device (for example "$" for some linux hosts).

connect_ssh(*host*, *username*, *password*, *version=None*, *proxy=None*, *prompt_endings=('#', '>')*)

Pretends to connect to a device via SSH. Simply tracks that we are now connected to something within this session (this method never fails).

Parameters

- **host** (*str*) – The IP address of DNS name for the device to connect
- **username** (*str*) – The username to login to the device with
- **password** (*str*) – The password that goes with the provided username. If a password is not specified, the user will be prompted for one.
- **version** (*int*) – The SSH version to connect with (1 or 2). Default is None, which will try 2 first and fallback to 1 if that fails.
- **proxy** (*str*) – The name of a SecureCRT session object that can be used as a jumpbox to proxy the SSH connection through. This is the same as selecting a session under the "Firewall" selection under the SSH settings screen for a SecureCRT session.
- **prompt_endings** (*list*) – A list of strings that are possible prompt endings to watch for. The default is for Cisco devices (">" and "#"), but may need to be changed if connecting to another type of device (for example "$" for some linux hosts).

connect_telnet(*host*, *username*, *password*, *proxy=None*, *prompt_endings=('#', '>')*)

Pretends to connect to a device via the Telnet protocol, just like connect_ssh above. Never fails.

Parameters

- **host** (*str*) – The IP address of DNS name for the device to connect
- **username** (*str*) – The username to login to the device with
- **password** (*str*) – The password that goes with the provided username. If a password is not specified, the user will be prompted for one.
- **proxy** (*str*) – The name of a SecureCRT session object that can be used as a jumpbox to proxy the SSH connection through. This is the same as selecting a session under the "Firewall" selection under the SSH settings screen for a SecureCRT session.
- **prompt_endings** (*list*) – A list of strings that are possible prompt endings to watch for. The default is for Cisco devices (">" and "#"), but may need to be changed if connecting to another type of device (for example "$" for some linux hosts).

create_new_saved_session(*session_name*, *ip*, *protocol='SSH2'*, *folder='_imports'*)
Pretends to create a new SecureCRT session. Since we aren't running in SecureCRT, it does nothing except print a message that a device was created.

Parameters

- **session_name** (*str*) – The name of the session
- **ip** (*str*) – The IP address or hostname of the device represented by this session
- **protocol** (*str*) – The protocol to use for this connection (TELNET, SSH1, SSH2, etc)
- **folder** (*str*) – The folder (starting from the configured Sessions folder) where this session should be saved.

disconnect(*command='exit'*)
Disconnects the main session used by the script by calling the disconnect method on the session object.

Parameters **command** (*str*) – The command to be issued to the remote device to disconnect. The default is 'exit'

file_open_dialog(*title*, *button_label='Open'*, *default_filename=''*, *file_filter=''*)
Prompts the user to select a file that will be processed by the script. In a direct session, the user will be prompted for the full path to a file.

Parameters

- **title** (*str*) – Title for the File Open dialog window (Only displays in Windows)
- **button_label** (*str*) – Label for the "Open" button
- **default_filename** (*str*) – If provided a default filename, the window will open in the parent directory of the file, otherwise the current working directory will be the starting directory.
- **file_filter** (*str*) – Specifies a filter for what type of files can be selected. The format is: <Name of Filter> (.*<extension>*)|.<extension>|| For example, a filter for CSV files would be "CSV Files (*.csv)*|.csv||" or multiple filters can be used: "Text Files (*.txt)*|.txt|Log File (*.log)*|.log||"

Returns The absolute path to the file that was selected

Return type str

message_box(*message*, *title=''*, *options=0*)
Prints a message for the user. When used in a DirectSession, the message is printed to the console and the user is prompted to type the button that would be selected.

This window can be customized by setting the "options" value, using the constants listed at the top of the sessions.py file. One constant from each of the 3 categories can be OR'd (|) together to make a single option value that will format the message box.

Parameters

- **message** (*str*) – The message to send to the user
- **title** (*str*) – Title for the message box
- **options** (*int*) – Sets the display format of the messagebox. (See Message Box constants in sessions.py)

Returns The return code that identifies which button the user pressed. (See Message Box constants)

Return type int

prompt_window(*message*, *title=''*, *hide_input=False*)
Prompts the user for an input value. In a direct session, the user will be prompted at the console for input.

The "hide_input" input will mask the input, so that passwords or other senstive information can be requested.

Parameters

- **message** (*str*) – The message to send to the user
- **title** (*str*) – Title for the prompt window
- **hide_input** (*bool*) – Specifies whether to hide the user input or not. Default is False.

Returns The value entered by the user

Return type str

ssh_in_new_tab(*host*, *username*, *password*, *prompt_endings=('#', '>')*)
Pretends to open a new tab. Since this is being run directly and no tabs exist, the function really does nothing but return a new Session object.

Parameters

- **host** (*str*) – The IP address of DNS name for the device to connect (only for API compatibility - not used)
- **username** (*str*) – The username to login to the device with (only for API compatibility - not used)
- **password** (*str*) – The password that goes with the provided username. If a password is not specified, the user will be prompted for one. (only for API compatibility - not used)
- **prompt_endings** (*list*) – A list of strings that are possible prompt endings to watch for. The default is for Cisco devices (">" and "#"), but may need to be changed if connecting to another type of device (for example "$" for some linux hosts). (only for API compatibility - not used)

Script Class Exceptions:

exception securecrt_tools.scripts.**ScriptError**
An exception type that is raised when there is a problem with the main scripts, such as missing settings files.

exception securecrt_tools.scripts.**ConnectError**
An exception type that is raised when there are problems connecting to a device.

9.3.1.2 securecrt_tools.sessions

This module includes a collection of "session" objects that represent a session to a remote device. To some degree, a session object can be thought of as a tab in SecureCRT, since you can disconnect from a device and then connect to others with the same session object. These classes are intended as a wrapper around the SecureCRT Python API to simplify common tasks that are performed against network devices such as routers and switches. As with the Script class, there is also a DebugSession class as a part of this module to allow running the code by the local python interpreter so that the scripts can be debugged.

The base class is named "Session" is the parent for more specific session types and includes all methods that must be implemented by all sub-classes.

The primary subclass is called "CRTSession" which is specific to interacting with the Python API for SecureCRT. Whenever a script is launched from SecureCRT, this class should be used to interact with SecureCRT.

A second subclass called "DebugSession" is used to run scripts outside of SecureCRT, such as in an IDE. This is useful for debugging because you can run your script along with the IDE debugging tools while simulating the interactions with SecureCRT. The class has the same API as CRTSession so that no modifications are needed to run a script directly. For example, while a CRTSession may directly pull input from a device before processing it, if the script is launched from an IDE, the DebugSession object will instead prompt for a filename containing the same output that would be been received from the remote device. The script can then be debugged step-by-step to help the programmer better understand where their logic is having trouble with a particular output.

The Session Base Class

class securecrt_tools.sessions.**Session**

This is a base class for the other Session types. This class simply exists to enforce the required methods any sub-classes have to implement. There are also a couple methods that are common to all sessions so they are defined under this class and automatically inherited by the sub-classes.

create_output_filename(*desc*, *ext='.txt'*, *include_hostname=True*, *include_date=True*, *base_dir=None*)

Generates a filename (including absoluate path) based on details from the session.

Parameters

- **desc** (*str*) – A short description to include in the filename (i.e. "show run", "cdp", etc)
- **ext** (*str*) – (Optional) Extension for the filename. Default: ".txt"
- **include_hostname** (*bool*) – (Optional) If true, includes the device hostname in the filename.
- **include_date** (*bool*) – (Optional) Include a timestamp in the filename. The timestamp format is taken from the settings file. Default: True
- **base_dir** (*str*) – (Optional) The directory where this file should be saved. Default: output_dir from settings.ini

Returns The generated absolute path for the filename requested.

Return type str

validate_os(*valid_os_list*)

This method checks if the remote device is running an OS in a list of valid OSes passed into the method. If the OS is not in the list then an exception is raised, which can either be allowed to cause the script to exit or be caught in a "try, except" statement and allow the script to take another action based on the result. If the remote OS is in the valid list, then nothing happens.

Parameters **valid_os_list** – A list of OSs that

CRTSession Class

class securecrt_tools.sessions.**CRTSession**(*script*, *tab*, *prompt_endings=None*)

Bases: *securecrt_tools.sessions.Session*

This sub-class of the Session class is used to wrap the SecureCRT API to simplify writing new scripts. An instance of this class represents a tab in SecureCRT and the methods in this class are used to connect to devices, disconnect from devices or interact with devices that are connected within the specific SecureCRT tab that this object represents.

close()

A method to close the SecureCRT tab associated with this CRTSession.

disconnect(*command='exit'*)
Disconnects the connected session by sending the "exit" command to the remote device. If that does not make the disconnect happen, attempt to force and ungraceful disconnect.

Parameters **command** (*str*) – The command to be issued to the remote device to disconnect. The default is 'exit'

end_cisco_session()
End the session by returning the device's terminal parameters that were modified by start_session() to their previous values.

This should always be called before a disconnect (assuming that start_cisco_session was called after connect)

get_command_output(*command*)
Captures the output from the provided command and saves the results in a variable.

** NOTE ** Assigning the output directly to a variable causes problems with SecureCRT for long outputs. It will gradually get slower and slower until the program freezes and crashes. The workaround is to save the output directly to a file (line by line), and then read it back into a variable. This is the procedure that this method uses.

Keyword Arguments:

param command Command string that should be sent to the device

type command str

Returns The result from issuing the above command.

Return type str

is_connected()
Returns a boolean value that describes if the session is currently connected.

Returns True if the session is connected, False if not.

Return type bool

save(*command='copy running-config startup-config'*)
Sends a "copy running-config startup-config" command to the remote device to save the running configuration.

send_config_commands(*command_list*, *output_filename=None*)
This method accepts a list of strings, where each string is a command to be sent to the device.

This method will send "conf t", then all the commands from the list and finally send "end" to the device. If an output_filenameThe results returned from entering the commands into the device are written to a file.

NOTE: This method is new and does not have any error checking for how the remote device handles the commands you are trying to send. USE IT AT YOUR OWN RISK.

Parameters

- **command_list** (*list*) – A list of strings, where each string is a command to be sent. This should NOT include 'config t' or 'end'. This is added automatically.
- **output_filename** (*str*) – (Optional) If a absolute path to a file is specified, the config session output from applying the commands will be written to this file.

start_cisco_session(*enable_pass=None*)
Performs initial setup of the session to a Cisco device by detecting parameters (prompt, hostname, network

OS, etc) of the connected device and modifying the terminal length if configured to do so in the settings file.

If the device is not at an enable prompt and an enable password is supplied, then this method will also enter enable mode on the device before proceeding.

This should always be called before trying to interact with a Cisco device so that the majority of other methods will work correctly. This should be one of the first calls in a script that is intended to run when already connected to the device, or called right after connecting to a device with the "connect_ssh" or similar method.

Parameters **enable_pass** (*str*) – The enable password that should be sent if the device is not in enable mode.

telnet_login(*username*, *password*)
Looks for username and password prompts and then responds to them with the provided credentials. Should only be needed when doing a telnet login.

Parameters

- **username** (*str*) – The username that needs to be sent to the device.
- **password** (*str*) – The password that needs to be sent to the device

write_output_to_file(*command*, *filename*, *prompt_to_create=True*)
Send the supplied command to the remote device and writes the output to a file.

This function was written specifically to write output line by line because storing large outputs into a variable will cause SecureCRT to bog down until it freezes. A good example is a large "show tech" output. This method can handle any length of output

Parameters

- **command** (*str*) – The command to be sent to the device
- **filename** (*str*) – A string with the absolute path to the filename to be written.

DebugSession Class

class securecrt_tools.sessions.**DebugSession**(*script*)
Bases: *securecrt_tools.sessions.Session*

This class is used when the scripts are executed directly from a local Python installation instead of from SecureCRT. This class is intended to simulate connectivity to remote devices by prompting the user for what would otherwise be extracted from SecureCRT. For example, when this class tries to get the output from a show command, it will instead prompt the user for a location of a file with the associated output. This allows the scripts to be run directly in an IDE for development and troubleshooting of more complicated logic around parsing command outputs.

close()
A method to close the SecureCRT tab associated with this CRTSession. Does nothing but print to the console.

disconnect(*command='exit'*)
Pretends to disconnects the connected session. Simply marks our session as disconnected.

Parameters **command** (*str*) – The command to be issued to the remote device to disconnect. The default is 'exit'

end_cisco_session()
End the session by returning the device's terminal parameters that were modified by start_session() to their previous values.

This should always be called before a disconnect (assuming that start_cisco_session was called after connect)

get_command_output(*command*)
Captures the output from the provided command and saves the results in a variable.

** NOTE ** Assigning the output directly to a variable causes problems with SecureCRT for long outputs. It will gradually get slower and slower until the program freezes and crashes. The workaround is to save the output directly to a file (line by line), and then read it back into a variable. This is the procedure that this method uses.

Keyword Arguments:

param command Command string that should be sent to the device

type command str

Returns The result from issuing the above command.

Return type str

is_connected()
Returns a boolean value that describes if the session is currently connected.

Returns True if the session is connected, False if not.

Return type bool

save(*command='copy running-config startup-config'*)
Pretends to send a "copy running-config startup-config" command to the remote device to save the running configuration. Only prints to the console.

send_config_commands(*command_list*, *output_filename=None*)
This method accepts a list of strings, where each string is a command to be sent to the device.

This method will pretend to send "conf t", then all the commands from the list and finally send "end" to the device. If an output_filename is specified, the (fake) results returned from entering the commands into the (fake) device are written to a file.

Parameters

- **command_list** (*list*) – A list of strings, where each string is a command to be sent. This should NOT include 'config t' or 'end'. This is added automatically.
- **output_filename** (*str*) – (Optional) If a absolute path to a file is specified, the config session output from applying the commands will be written to this file.

start_cisco_session(*enable_pass=None*)
Performs initial setup of the session to a Cisco device by detecting parameters (prompt, hostname, network OS, etc) of the connected device and modifying the terminal length if configured to do so in the settings file.

Always assumes that we are already in enable mode (privilege 15)

This should always be called before trying to interact with a Cisco device so that the majority of other methods will work correctly. This should be one of the first calls in a script that is intended to run when already connected to the device, or called right after connecting to a device with the "connect_ssh" or similar method.

Parameters **enable_pass** (*str*) – The enable password that should be sent if the device is not in enable mode.

write_output_to_file(*command*, *filename*, *prompt_to_create=True*)
Send the supplied command to the remote device and writes the output to a file.

This function was written specifically to write output line by line because storing large outputs into a variable will cause SecureCRT to bog down until it freezes. A good example is a large "show tech" output. This method can handle any length of output

Parameters

- **command** (*str*) – The command to be sent to the device
- **filename** (*str*) – A string with the absolute path to the filename to be written.

Session Class Exceptions:

exception securecrt_tools.sessions.**InteractionError**
An exception type used when an expected response isn't received when interacting with a device.

exception securecrt_tools.sessions.**UnsupportedOSError**
An exception type used when the remote device is running an OS that isn't supported by the script.

9.3.1.3 securecrt_tools.settings

SettingsImporter

class securecrt_tools.settings.**SettingsImporter**(*settings_file*, *create=False*)
A class to handle validating, retrieving and updating settings as needed.

validate_settings()
A method to check if the user's settings.ini file contains all of the correct settings.

Returns A boolean describing if the user's settings file is valid

Return type bool

correct_settings()
A method to update the user's settings file to match the current correct version while carrying over current values to the new file. Adds anything in defaults that isn't in the user's settings to the settings.ini file. This does not remove any additions that may have been added to the user's configuration file.

get(*section*, *setting*)
A wrapper function to simplify the retrieval of an individual setting.

Parameters

- **section** (*str*) – The section of the settings file where the setting can be found.
- **setting** (*str*) – The name of the setting we want to retrieve

Returns The value of the setting requested

Return type str

update(*section*, *setting*, *value*)
A wrapper function to update a setting

Parameters

- **section** (*str*) – The section of the settings file where the setting can be found.
- **setting** (*str*) – The name of the setting we want to retrieve
- **value** (*str*) – The value to store for this setting

getboolean(*section*, *setting*)
A wrapper function to simplify the retrieval of an individual setting as a boolean value.

Parameters

- **section** (*str*) – The section of the settings file where the setting can be found.
- **setting** (*str*) – The name of the setting we want to retrieve

Returns The value of the setting requested as a boolean

Return type bool

getint(*section*, *setting*)
A wrapper function to simplify the retrieval of an individual setting as an integer.

Parameters

- **section** (*str*) – The section of the settings file where the setting can be found.
- **setting** (*str*) – The name of the setting we want to retrieve

Returns The value of the setting requested as an integer

Return type int

getlist(*section*, *setting*)
A wrapper function to simplify the retrieval of an individual setting as a list. Requires the setting to be a comma separated list, with no quotations.

Parameters

- **section** (*str*) – The section of the settings file where the setting can be found.
- **setting** (*str*) – The name of the setting we want to retrieve

Returns The value of the setting requested as a list.

Return type int

9.3.1.4 securecrt_tools.utilities

securecrt_tools.utilities.**expand_number_range**(*num_string*)
A function that will accept a text number range (such as 1,3,5-7) and convert it into a list of integers such as [1, 3, 5, 6, 7]

Parameters **num_string** – <str> A string that is in the format of a number range (e.g. 1,3,5-7)

Returns <list> A list of all integers in that range (e.g. [1,3,5,6,7])

securecrt_tools.utilities.**extract_system_name**(*device_id*, *strip_list=[]*)
In the CDP output some systems return a Hostname(Serial Number) format, while others return Serial(Hostname) output. This function tries to extract the system name from the CDP output and ignore the serial number.

Parameters

- **device_id** – The device_id as learned from CDP.
- **strip_list** – A list of strings that should be removed from the hostname, if found

Returns

securecrt_tools.utilities.**human_sort_key**(*s*)
A key function to sort alpha-numerically, not by string

From http://nedbatchelder.com/blog/200712/human_sorting.html This function can be used as the key for a sort algorithm to give it an understanding of numbers, i.e. [a1, a2, a10], instead of the default (ASCII) sorting, i.e. [a1, a10, a2].

Parameters **s** –

Returns

securecrt_tools.utilities.**list_of_dicts_to_csv**(*data*, *filename*, *header*, *add_header=True*)

Parameters

- **data** –
- **filename** –
- **header** –

Returns

securecrt_tools.utilities.**list_of_lists_to_csv**(*data*, *filename*)
Takes a list of lists and writes it to a csv file.

This function takes a list of lists, such as:

[[“IP”, “Desc”], [“1.1.1.1”, “Vlan 1”], [“2.2.2.2”, “Vlan 2”]]

and writes it into a CSV file with the filename supplied. Each sub-list in the outer list will be written as a row. If you want a header row, it must be the first sub-list in the outer list.

Parameters

- **data** – <2d-list> A list of lists data structure (one row per line of the CSV)
- **filename** – <str> The output filename for the CSV file, that will be placed in the ‘save path’ directory under the global settings.

securecrt_tools.utilities.**long_int_name**(*short_name*)
This function expands a short interface name to the full name

Parameters **short_name** – The input string (short interface name)

Returns The shortened interface name

securecrt_tools.utilities.**normalize_protocol**(*raw_protocol*)
A function to normalize protocol names between IOS and NXOS. For example, IOS uses ‘C’ and NXOS uses ‘direct” for connected routes. This function will return ‘connected’ in both cases.

Parameters **raw_protocol** – <str> The protocol value found in the route table output

Returns A normalized name for that type of route.

securecrt_tools.utilities.**path_safe_name**(*input_string*)
This function will remove or replace characters in the input string so that the output is suitable to be used as a file or directory name.

Parameters **input_string** (*str*) – The string that should be converted into a filename safe version.

Returns The filename safe version of the input string

securecrt_tools.utilities.**remove_empty_or_invalid_file**(*l_filename*)
Check if file is empty or if we captured an error in the command. If so, delete the file.

Parameters **l_filename** – Name of file to check

securecrt_tools.utilities.**short_int_name**(*long_name*)
This function shortens the interface name for easier reading

Parameters **long_name** – The input string (long interface name)

Returns The shortened interface name

securecrt_tools.utilities.**textfsm_parse_to_dict**(*input_data*, *template_filename*)
Use TextFSM to parse the input text (from a command output) against the specified TextFSM template. Convert each list from the output to a dictionary, where each key in the TextFSM Value name from the template file.

Parameters

- **input_data** – Path to the input file that TextFSM will parse.
- **template_filename** – Path to the template file that will be used to parse the above data.

Returns A list, with each entry being a dictionary that maps TextFSM variable name to corresponding value.

securecrt_tools.utilities.**textfsm_parse_to_list**(*input_data*, *template_name*, *add_header=False*)
Use TextFSM to parse the input text (from a command output) against the specified TextFSM template. Use the default TextFSM output which is a list, with each entry of the list being a list with the values parsed. Use add_header=True if the header row with value names should be prepended to the start of the list.

Parameters

- **input_data** – Path to the input file that TextFSM will parse.
- **template_name** – Path to the template file that will be used to parse the above data.
- **add_header** – When True, will return a header row in the list. This is useful for directly outputting to CSV.

Returns The TextFSM output (A list with each entry being a list of values parsed from the input)

9.3.2 Third Party Modules

9.3.2.1 securecrt_tools.textfsm

Template based text parser.

This module implements a parser, intended to be used for converting human readable text, such as command output from a router CLI, into a list of records, containing values extracted from the input text.

A simple template language is used to describe a state machine to parse a specific type of text input, returning a record of values for each input entity.

class securecrt_tools.textfsm.**CopyableRegexObject**(*pattern*)
Bases: object

Like a re.RegexObject, but can be copied.

match(**args*, ***kwargs*)

sub(**args*, ***kwargs*)

exception securecrt_tools.textfsm.**Error**
Bases: exceptions.Exception

Base class for errors.

exception securecrt_tools.textfsm.**FSMAction**
Bases: exceptions.Exception

Base class for actions raised with the FSM.

exception securecrt_tools.textfsm.**SkipRecord**
Bases: *securecrt_tools.textfsm.FSMAction*

Indicate a record is to be skipped.

exception securecrt_tools.textfsm.**SkipValue**
Bases: *securecrt_tools.textfsm.FSMAction*

Indicate a value is to be skipped.

class securecrt_tools.textfsm.**TextFSM**(*template*, *options_class=<class 'securecrt_tools.textfsm.TextFSMOptions'>*)
Bases: object

Parses template and creates Finite State Machine (FSM).

Attributes: states: (str), Dictionary of FSMState objects. values: (str), List of FSMVariables. value_map: (map), For substituting values for names in the expressions. header: Ordered list of values. state_list: Ordered list of valid states.

GetValuesByAttrib(*attribute*)
Returns the list of values that have a particular attribute.

MAX_NAME_LEN = 48

ParseText(*text*, *eof=True*)
Passes CLI output through FSM and returns list of tuples.

First tuple is the header, every subsequent tuple is a row.

Args: text: (str), Text to parse with embedded newlines. eof: (boolean), Set to False if we are parsing only part of the file. Suppresses triggering EOF state.

Raises: TextFSMError: An error occurred within the FSM.

Returns: List of Lists.

Reset()
Preserves FSM but resets starting state and current record.

comment_regex = <_sre.SRE_Pattern object>

header
Returns header.

state_name_re = <_sre.SRE_Pattern object>

exception securecrt_tools.textfsm.**TextFSMError**
Bases: *securecrt_tools.textfsm.Error*

Error in the FSM state execution.

class securecrt_tools.textfsm.**TextFSMOptions**
Bases: object

Class containing all valid TextFSMValue options.

Each nested class here represents a TextFSM option. The format is "option<name>". Each class may override any of the methods inside the OptionBase class.

A user of this module can extend options by subclassing TextFSMOptionsBase, adding the new option class(es), then passing that new class to the TextFSM constructor with the 'option_class' argument.

class Filldown(*value*)
Bases: securecrt_tools.textfsm.OptionBase

Value defaults to the previous line's value.

OnAssignVar()
Called when a matched value is being assigned.

OnClearAllVar()
Called when a value has clearalled.

OnClearVar()
Called when value has been cleared.

OnCreateOptions()
Called after all options have been parsed for a Value.

class Fillup(*value*)
Bases: securecrt_tools.textfsm.OptionBase

Like Filldown, but upwards until it finds a non-empty entry.

OnAssignVar()
Called when a matched value is being assigned.

classmethod GetOption(*name*)
Returns the class of the requested option name.

class Key(*value*)
Bases: securecrt_tools.textfsm.OptionBase

Value constitutes part of the Key of the record.

class List(*value*)
Bases: securecrt_tools.textfsm.OptionBase

Value takes the form of a list.

OnAssignVar()
Called when a matched value is being assigned.

OnClearAllVar()
Called when a value has clearalled.

OnClearVar()
Called when value has been cleared.

OnCreateOptions()
Called after all options have been parsed for a Value.

OnSaveRecord()
Called just prior to a record being committed.

class OptionBase(*value*)
Bases: object

Factory methods for option class.

Attributes: value: A TextFSMValue, the parent Value.

OnAssignVar()
Called when a matched value is being assigned.

OnClearAllVar()
Called when a value has clearalled.

OnClearVar()
Called when value has been cleared.

OnCreateOptions()
Called after all options have been parsed for a Value.

OnGetValue()
Called when the value name is being requested.

OnSaveRecord()
Called just prior to a record being committed.

name

class Required(*value*)
Bases: securecrt_tools.textfsm.OptionBase

The Value must be non-empty for the row to be recorded.

OnSaveRecord()
Called just prior to a record being committed.

classmethod ValidOptions()
Returns a list of valid option names.

class securecrt_tools.textfsm.**TextFSMRule**(*line*, *line_num=-1*, *var_map=None*)
Bases: object

A rule in each FSM state.

A value has syntax like:

^<regexp> -> Next.Record State2

Where '<regexp>' is a regular expression. 'Next' is a Line operator. 'Record' is a Record operator. 'State2' is the next State.

Attributes: match: Regex to match this rule. regex: match after template substitution. line_op: Operator on input line on match. record_op: Operator on output record on match. new_state: Label to jump to on action regex_obj: Compiled regex for which the rule matches. line_num: Integer row number of Value.

ACTION2_RE = <_sre.SRE_Pattern object>

ACTION3_RE = <_sre.SRE_Pattern object>

ACTION_RE = <_sre.SRE_Pattern object>

LINE_OP = ('Continue', 'Next', 'Error')

LINE_OP_RE = '(?P<ln_op>Continue|Next|Error)'

MATCH_ACTION = <_sre.SRE_Pattern object>

NEWSTATE_RE = '(?P<new_state>\\w+|\\".*\\")'

OPERATOR_RE = '((?P<ln_op>Continue|Next|Error)(\\.(?P<rec_op>Clear|Clearall|Record|NoR

RECORD_OP = ('Clear', 'Clearall', 'Record', 'NoRecord')

RECORD_OP_RE = '(?P<rec_op>Clear|Clearall|Record|NoRecord)'

exception securecrt_tools.textfsm.**TextFSMTemplateError**
Bases: *securecrt_tools.textfsm.Error*

Errors while parsing templates.

class securecrt_tools.textfsm.**TextFSMValue**(*fsm=None*, *max_name_len=48*, *options_class=None*)
Bases: object

A TextFSM value.

A value has syntax like:

'Value Filldown,Required helloworld (.*)'

Where 'Value' is a keyword. 'Filldown' and 'Required' are options. 'helloworld' is the value name. '(.*) is the regular expression to match in the input data.

Attributes: max_name_len: (int), maximum character length os a variable name. name: (str), Name of the value. options: (list), A list of current Value Options. regex: (str), Regex which the value is matched on. template: (str), regexp with named groups added. fsm: A TextFSMBase(), the containing FSM. value: (str), the current value.

AssignVar(*value*)
Assign a value to this Value.

ClearAllVar()
Clear this Value.

ClearVar()
Clear this Value.

Header()
Fetch the header name of this Value.

OnSaveRecord()
Called just prior to a record being committed.

OptionNames()
Returns a list of option names for this Value.

Parse(*value*)
Parse a 'Value' declaration.

Args: value: String line from a template file, must begin with 'Value '.

Raises: TextFSMTemplateError: Value declaration contains an error.

exception securecrt_tools.textfsm.**Usage**
Bases: exceptions.Exception

Error in command line execution.

securecrt_tools.textfsm.**main**(*argv=None*)
Validate text parsed with FSM or validate an FSM via command line.

9.3.2.2 securecrt_tools.ipaddress

A fast, lightweight IPv4/IPv6 manipulation library in Python.

This library is used to create/poke/manipulate IPv4 and IPv6 addresses and networks.

exception securecrt_tools.ipaddress.**AddressValueError**
Bases: exceptions.ValueError

A Value Error related to the address.

class securecrt_tools.ipaddress.**IPv4Address**(*address*)
Bases: securecrt_tools.ipaddress._BaseV4, securecrt_tools.ipaddress._BaseAddress

Represent and manipulate single IPv4 Addresses.

is_global

is_link_local
Test if the address is reserved for link-local.

Returns: A boolean, True if the address is link-local per RFC 3927.

is_loopback
Test if the address is a loopback address.

Returns: A boolean, True if the address is a loopback per RFC 3330.

is_multicast
Test if the address is reserved for multicast use.

Returns: A boolean, True if the address is multicast. See RFC 3171 for details.

is_private
Test if this address is allocated for private networks.

Returns: A boolean, True if the address is reserved per iana-ipv4-special-registry.

is_reserved
Test if the address is otherwise IETF reserved.

Returns: A boolean, True if the address is within the reserved IPv4 Network range.

is_unspecified
Test if the address is unspecified.

Returns: A boolean, True if this is the unspecified address as defined in RFC 5735 3.

packed
The binary representation of this address.

class securecrt_tools.ipaddress.**IPv4Interface**(*address*)
Bases: *securecrt_tools.ipaddress.IPv4Address*

ip

with_hostmask

with_netmask

with_prefixlen

class securecrt_tools.ipaddress.**IPv4Network**(*address*, *strict=True*)
Bases: securecrt_tools.ipaddress._BaseV4, securecrt_tools.ipaddress._BaseNetwork

This class represents and manipulates 32-bit IPv4 network + addresses..

Attributes: [examples for IPv4Network('192.0.2.0/27')] .network_address: IPv4Address('192.0.2.0') .hostmask: IPv4Address('0.0.0.31') .broadcast_address: IPv4Address('192.0.2.32') .netmask: IPv4Address('255.255.255.224') .prefixlen: 27

is_global
Test if this address is allocated for public networks.

Returns: A boolean, True if the address is not reserved per iana-ipv4-special-registry.

class securecrt_tools.ipaddress.**IPv6Address**(*address*)
Bases: securecrt_tools.ipaddress._BaseV6, securecrt_tools.ipaddress._BaseAddress

Represent and manipulate single IPv6 Addresses.

ipv4_mapped
Return the IPv4 mapped address.

Returns: If the IPv6 address is a v4 mapped address, return the IPv4 mapped address. Return None otherwise.

is_global
Test if this address is allocated for public networks.

Returns: A boolean, true if the address is not reserved per iana-ipv6-special-registry.

is_link_local
Test if the address is reserved for link-local.

Returns: A boolean, True if the address is reserved per RFC 4291.

is_loopback
Test if the address is a loopback address.

Returns: A boolean, True if the address is a loopback address as defined in RFC 2373 2.5.3.

is_multicast
Test if the address is reserved for multicast use.

Returns: A boolean, True if the address is a multicast address. See RFC 2373 2.7 for details.

is_private
Test if this address is allocated for private networks.

Returns: A boolean, True if the address is reserved per iana-ipv6-special-registry.

is_reserved
Test if the address is otherwise IETF reserved.

Returns: A boolean, True if the address is within one of the reserved IPv6 Network ranges.

is_site_local
Test if the address is reserved for site-local.

Note that the site-local address space has been deprecated by RFC 3879. Use is_private to test if this address is in the space of unique local addresses as defined by RFC 4193.

Returns: A boolean, True if the address is reserved per RFC 3513 2.5.6.

is_unspecified
Test if the address is unspecified.

Returns: A boolean, True if this is the unspecified address as defined in RFC 2373 2.5.2.

packed
The binary representation of this address.

sixtofour
Return the IPv4 6to4 embedded address.

Returns: The IPv4 6to4-embedded address if present or None if the address doesn't appear to contain a 6to4 embedded address.

teredo
Tuple of embedded teredo IPs.

Returns: Tuple of the (server, client) IPs or None if the address doesn't appear to be a teredo address (doesn't start with 2001::/32)

class securecrt_tools.ipaddress.**IPv6Interface**(*address*)
Bases: *securecrt_tools.ipaddress.IPv6Address*

ip

is_loopback

is_unspecified

with_hostmask

with_netmask

with_prefixlen

class securecrt_tools.ipaddress.**IPv6Network**(*address*, *strict=True*)
Bases: securecrt_tools.ipaddress._BaseV6, securecrt_tools.ipaddress._BaseNetwork

This class represents and manipulates 128-bit IPv6 networks.

Attributes: [examples for IPv6('2001:db8::1000/124')] .network_address: IPv6Address('2001:db8::1000') .hostmask: IPv6Address('::f') .broadcast_address: IPv6Address('2001:db8::100f') .netmask: IPv6Address('ffff:ffff:ffff:ffff:ffff:ffff:ffff:fff0') .prefixlen: 124

hosts()
Generate Iterator over usable hosts in a network.

This is like __iter__ except it doesn't return the Subnet-Router anycast address.

is_site_local
Test if the address is reserved for site-local.

Note that the site-local address space has been deprecated by RFC 3879. Use is_private to test if this address is in the space of unique local addresses as defined by RFC 4193.

Returns: A boolean, True if the address is reserved per RFC 3513 2.5.6.

exception securecrt_tools.ipaddress.**NetmaskValueError**
Bases: exceptions.ValueError

A Value Error related to the netmask.

securecrt_tools.ipaddress.**collapse_addresses**(*addresses*)
Collapse a list of IP objects.

Example:

collapse_addresses([IPv4Network('192.0.2.0/25'), IPv4Network('192.0.2.128/25')]) -> [IPv4Network('192.0.2.0/24')]

Args: addresses: An iterator of IPv4Network or IPv6Network objects.

Returns: An iterator of the collapsed IPv(4|6)Network objects.

Raises: TypeError: If passed a list of mixed version objects.

securecrt_tools.ipaddress.**get_mixed_type_key**(*obj*)
Return a key suitable for sorting between networks and addresses.

Address and Network objects are not sortable by default; they're fundamentally different so the expression

IPv4Address('192.0.2.0') <= IPv4Network('192.0.2.0/24')

doesn't make any sense. There are some times however, where you may wish to have ipaddress sort these for you anyway. If you need to do this, you can use this function as the key= argument to sorted().

Args: obj: either a Network or Address object.

Returns: appropriate key.

securecrt_tools.ipaddress.**ip_address**(*address*)
Take an IP string/int and return an object of the correct type.

Args:

address: A string or integer, the IP address. Either IPv4 or IPv6 addresses may be supplied; integers less than 2**32 will be considered to be IPv4 by default.

Returns: An IPv4Address or IPv6Address object.

Raises:

ValueError: if the *address* passed isn't either a v4 or a v6 address

securecrt_tools.ipaddress.**ip_interface**(*address*)
Take an IP string/int and return an object of the correct type.

Args:

address: A string or integer, the IP address. Either IPv4 or IPv6 addresses may be supplied; integers less than 2**32 will be considered to be IPv4 by default.

Returns: An IPv4Interface or IPv6Interface object.

Raises:

ValueError: if the string passed isn't either a v4 or a v6 address.

Notes: The IPv?Interface classes describe an Address on a particular Network, so they're basically a combination of both the Address and Network classes.

securecrt_tools.ipaddress.**ip_network**(*address*, *strict=True*)
Take an IP string/int and return an object of the correct type.

Args:

address: A string or integer, the IP network. Either IPv4 or IPv6 networks may be supplied; integers less than 2**32 will be considered to be IPv4 by default.

Returns: An IPv4Network or IPv6Network object.

Raises:

ValueError: if the string passed isn't either a v4 or a v6 address. Or if the network has host bits set.

securecrt_tools.ipaddress.**summarize_address_range**(*first*, *last*)
Summarize a network range given the first and last IP addresses.

Example:

```
>>> list(summarize_address_range(IPv4Address('192.0.2.0'),
...                              IPv4Address('192.0.2.130')))
...                                #doctest: +NORMALIZE_WHITESPACE
[IPv4Network('192.0.2.0/25'), IPv4Network('192.0.2.128/31'),
 IPv4Network('192.0.2.130/32')]
```

Args: first: the first IPv4Address or IPv6Address in the range. last: the last IPv4Address or IPv6Address in the range.

Returns: An iterator of the summarized IPv(4|6) network objects.

Raise:

TypeError: If the first and last objects are not IP addresses. If the first and last objects are not the same version.

ValueError: If the last object is not greater than the first. If the version of the first address is not 4 or 6.

securecrt_tools.ipaddress.**v4_int_to_packed**(*address*)

Represent an address as 4 packed bytes in network (big-endian) order.

Args: address: An integer representation of an IPv4 IP address.

Returns: The integer address packed as 4 bytes in network (big-endian) order.

Raises:

ValueError: If the integer is negative or too large to be an IPv4 IP address.

securecrt_tools.ipaddress.**v6_int_to_packed**(*address*)

Represent an address as 16 packed bytes in network (big-endian) order.

Args: address: An integer representation of an IPv6 IP address.

Returns: The integer address packed as 16 bytes in network (big-endian) order.

9.3.2.3 securecrt_tools.manuf

Parser library for Wireshark's OUI database.

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

See README.md.

class securecrt_tools.manuf.**MacParser**(*manuf_name='manuf'*, *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

Args: 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=m

get_all(*mac*)

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

Args: mac (str): MAC address in standard format.

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

Raises: ValueError: If the MAC could not be parsed.

get_comment(*mac*)
Returns comment from a MAC address.

Args: mac (str): MAC address in standard format.

Returns: string: String containing comment, or None if not found.

Raises: ValueError: If the MAC could not be parsed.

get_manuf(*mac*)
Returns manufacturer from a MAC address.

Args: mac (str): MAC address in standard format.

Returns: string: String containing manufacturer, or None if not found.

Raises: ValueError: If the MAC could not be parsed.

refresh(*manuf_name=None*)
Refresh/reload manuf database. Call this when manuf file is updated.

Args:

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.

Args: 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.

Args:

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 securecrt_tools.manuf.**Vendor**(*manuf*, *comment*)
Bases: tuple

comment
Alias for field number 1

manuf
Alias for field number 0

securecrt_tools.manuf.**main**()
Simple command line wrapping for MacParser.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W