C.R.A.SSH - Cisco Remote Automation via SSH Documentation

Release 02

Nick Bettison

February 25, 2017

Contents

C.R.A.SSH (*crassh*) stands for *Cisco Remote Automation via SSH*, it is a python script for automating commands on Cisco devices.

Crassh can be used by network administrators to quickly run the same command(s) on multiple devices, or it can be imported as a module by developers as part of a wider Cisco/Python project.

CHAPTER 1

Why crassh?

I've called the tool Cisco Remote Automation via SSH, or C.R.A.SSH for short. The name is in homage to S.H.I.E.L.D because I really wanted the name to sound like "crash" as a way of reminding users that if you are not careful this script is a car-crash-waiting-to-happen!

Disclaimer

The word *Cisco* is used as a description because this script should work with any Cisco IOS device. Cisco is a registered trademark of Cisco Systems Inc; this script is not associated, endorsed, supported or affiliated in any way with Cisco and none of these are implied.

Contents:

Installing C.R.A.SSH

Crassh can be installed in two ways, either as a standalone script for users or via *pip* for developers.

Standalone installations are intended for Network Administrators

Developer installations are intended for those who want crassh imported into their own python scripts or wish the script to fall under package (*version*) management.

Standalone Installation

You'll need both python and Paramiko, once you have both of those just download crassh.py direct from github and save it somewhere (*like* `$HOME/bin`), e.g:

```
curl -k -o crassh https://raw.githubusercontent.com/linickx/crassh/master/crassh.py
chmod +x crassh
```

Developer (PIP) Installation

Crassh has been published on PyPi: https://pypi.python.org/pypi/CraSSH

If your system supports pip (*with Internet access*) then crassh can be installed with:

```
pip install crassh
```

The PIP installation will solve the dependencies and will make the command `crassh` available in your *$PATH* and make crassh available for `import` within your own python scripts.

Dependencies

If you are not using `pip` then Paramiko will need to be manually installed:

Paramiko on Linux

For debian/ubuntu boxes:

```
sudo apt-get install python-paramiko
```

For redhat/fedora boxes:

```
sudo yum install python-paramiko
```

Paramiko on OS X

For apples, get homebrew setup and then:

```
brew install python
pip install paramiko
```

Paramiko on Windows

For windohz boxes, it's a bit more complicated.

- Download and install Visual Studio C++ 2008 Express Edition (*do not install SQL*)
- Install Python 2.7.8 – Select the correct MSI for your architecture
- Download get-pip.py (*Don't use Internet Explorer it will mangle the file; _use Firefox_ to download.*)
- Open an **Administrator** command prompt and run:

```
c:\Python27\python.exe get-pip.py
```

- From the same admin prompt, run:

```
C:\Program Files\Microsoft Visual Studio 9.0\Common7\Tools\vsvars32.bat
```

- that's for 32bit machines... or for 64bit machines, run:

```
C:\Program Files (x86)\Microsoft Visual Studio 9.0\Common7\Tools\vsvars64.bat
```

- From the same admin prompt, run:

```
c:\Python27\Scripts\pip install paramiko
```

User Guide

The following documentation is intended for Network Administrators. Crassh provides a way of automating commands on Cisco IOS devcies, that being either lots of commands on one device, one command on lots or devices, or a combination of both.

No Python, programming or scripting knowledge is required to run crassh, it is simply a command line tool that you run on your local PC/Laptop

My personal blog contains a tutorial here on how to use crassh in standalone mode which is a subset of the documentation found here.

Assuming that you have performed a standalone installation, the script would be run from the current directory and is quite straight forward, `./crassh`.

If you have installed crassh via pip, the crassh command should be available without the `./`

Crassh has a version specific built in help with `-h`, e.g

```
linickx:crassh nick$ ./crassh -h


Nick's Cisco Remote Automation via Secure Shell - Script, or C.R.A.SSH for short!

Usage: ./crassh -s switches.txt -c commands.txt -p -w -t 45 -e
   -s supply a text file of switch hostnames or IP addresses [optional]"
   -c supply a text file of commands to run on switches [optional]"
   -w write the output to a file [optional | Default: True]"
   -p print the output to the screen [optional | Default: False]"
   -pw is supported, will print the output to screen and write the output to file! [optional]"
   -t set a command timeout in seconds [optional | Default: 60]"
   -T set a connection timeout in seconds [optional | Default: 10]
   -X disable \"do no harm\" [optional]"
   -Q disable \"quit on failure\" [optional]"
   -e set an enable password [optional]"
   -d set a delay between commands [optional]"
   -A set an Authentication file for SSH credentials [optional]
   -U set a Username for SSH Authentication [optional]
   -P set a Password for SSH Authentication [optional]
   -B set a BACKUP Username for SSH Authentication [optional]
   -b set a BACKUP Password for SSH Authentication [optional]
   -E set a BACKUP ENABLE Password [optional]

Version: 2.6

linickx:crassh nick$
```

Input files

The `-s` option allows you to feed in a *switch* file, i.e. a list of devices to connect to, the format is a simple plain text file (`*.txt`), one device per line, (*either IP addresses or resolvable names is fine*) eg:

```
192.168.1.72
coreswitch.domain.local
accessswitch1.domain.local
```

The `-c` option allows you to run multiple commands; same format as before, a simple plain text file (`*.txt`), one command per line. For example:

```
show ver
show log
```

You can even make config changes:

```
conf t
interface GigabitEthernet1/9
description *** UNUSED ***
```

If you want to mix *config* commands with *show* commands then you need to include **exits** , e.g:

```
show run int g1/9
conf t
interface GigabitEthernet1/9
description *** UNUSED ***
exit
exit
show run int g1/9
```

Authentication

By default crassh will prompt for username and password credentials; `-U` can be used to supply a username as a CLI option, `-P` can be used to supply a password. **Please take note that ‘‘-P‘‘ may expose your password in the command line history**

crassh will look for and read a `~/.crasshrc` file; currently the file supports two colon separated variables `username` and `password`:

```
username: nick
password: mysecretpass
```

STORING YOUR PASSWORD IN PLAIN TEXT IN ‘‘~/.crasshrc‘‘ IS A SECURITY RISK Please appropriately secure your system; crassh will perform a basic file permission check.

The `-A` option can be used to specify different authentication files, for example `-A /var/secrets/router_credentials.txt`

Backup Credentials

If the TACACS (ACS) server does not respond or the environment has a mixture of central & local credentials the `-B` option can be used to supply a backup username. `-b` can be used to supply a backup password and `-E` used for a backup enable password.

Do no Harm

crassh has a very basic safe mode, i.e. to stop users reloading all their switches on the network at once; if that is something you really *really* want to do then `-X` is what you need!

Print Vs Write

By default, crassh will write it’s output to a file, in the format hostname-YearMonthDate-HourMinuteSecond. If you suppy the `-p` option, crassh will output to screen instead. If you want to Print and Write, use `-pw`

Quit on Failure

crassh by default will stop in it's tracks (quit/exit) if there is a connectivity failure to a device, this is to stop invalid credentials hammering a list of devices and potentially locking out TACACS accounts. **BUT** this also means that if there is network error (*i.e. TCP/IP connectivity issue*) then crassh will also stop, the `-Q` option can be used to disable *Quit on Failure*

Execution Timeout

Let's say you run a command that take a long time, say a million pings, crassh will wait for 60 seconds for the command to complete and then bail and move on to the next command - this should be fine for most commands. If you do actually want to send a million pings, then use the `-t` option to extend the timeout (*i.e how long crassh will wait*)

Developer Guide

Crassh is supplied as a Python module which developers can include in their own scripts. Crassh is a Paramiko wrapper specifically designed for talking to Cisco IOS devices and routers.

Developers/Coders are reminded not to *reinvent the wheel*, crassh (*as a standalone script*) can already read commands from a file and execute them on either one device or many devices (*i.e. read list of devices from a file*), tasks such as backing up the network estate do not require any additional scripts/development.

Where crassh *as a module* is valuable is doing *something* other than executing commands and printing/storing the result.

An example of *doing something* is writing an auditing script; the following example is taken from my personal blog where crassh can be used in a script to look for the *insecure* SNMP community `public`.

```
#!/usr/bin/env python
# coding=utf-8

import crassh

# Variables
routers = ["10.159.83.135", "10.159.83.136"]
username = "nick"
password = "nick"

# Loop
for device in routers:

    try:
        hostname = crassh.connect(device, username, password)

        output = crassh.send_command("show run | inc snmp-server community", hostname)
        crassh.disconnect()

        # Split the output by spaces so we can search the response
        words = output.split()

        # Look for "public" in the output
        for x in words:
            if x == "public":
                print("DANGER: Public SNMP Community set on %s [%s]" % (hostname, device))
```

```
    except:
        pass # If connect fails, move onto next router in the list.
```

C.R.A.SSH (crassh) autodoc

The *autodoc* automagically documents all of the functions from the source code.

Python script to automate running commands on switches. Cisco Remote Automation via Secure Shell... or C.R.A.SSH for short!

`crassh.`**`connect`**(*device='127.0.0.1'*, *username='cisco'*, *password='cisco'*, *enable=False*, *enable_password='cisco'*, *sysexit=False*, *timeout=10*)

Connect and get Hostname of Cisco Device

This function wraps up `paramiko` and returns the hostname of the **Cisco** device. The function creates two global variables `remote_conn_pre` and `remote_conn` which are the paramiko objects for direct manipulation if necessary.

Args: device (str): IP Address or Fully Qualifed Domain Name of Device

username (str): Username for SSH Authentication

password (str): Password for SSH Authentication

enable (bool): Is enable going to be needed?

enable_password (str): The enable password

sysexit (bool): Should the connecton exit the script on failure?

Returns: str. The hostname of the device

Example:

```
    >>> hostname = connect("10.10.10.10", "nick", "cisco")
    >>> print(hostname)
    r1
```

REF:

- https://pynet.twb-tech.com/blog/python/paramiko-ssh-part1.html
- http://yenonn.blogspot.co.uk/2013/10/python-in-action-paramiko-handling-ssh.html

`crassh.`**`disconnect`**()

Disconnect an SSH Session

Crassh wrapper for paramiko disconnect

No Argumanets, disconnects the current global variable `remote_conn_pre`

`crassh.`**`do_no_harm`**(*command*)

Check Commands for dangerous things

Args: command (str): The Command you wish to run on the device.

Returns: Nothing

This function will `sys.exit()` if an *evil* command is found

```
    >>> crassh.do_no_harm("show ver")
    >>>
```

So, good commands just pass through with no response... maybe I should oneday make it a True/False kind of thing.

crassh.**isgroupreadable**(*filepath*)
Checks if a file is *Group* readable

Args: filepath (str): Full path to file

Returns: bool. True/False

Example:

```
>>> print(str(isgroupreadable("file.txt")))
True
```

REF: http://stackoverflow.com/questions/1861836/checking-file-permissions-in-linux-with-python

crassh.**isotherreadable**(*filepath*)
Checks if a file is *Other* readable

Args: filepath (str): Full path to file

Returns: bool. True/False

Example:

```
>>> print(str(isotherreadable("file.txt")))
True
```

crassh.**main**()
Main Code Block

This is the main script that Network Administrators will run.

No Argumanets. Input is used for missing CLI Switches.

crassh.**print_help**(*exitcode=0*)
Prints the Help for the CLI tool

Args: exit (int): Exit Code

Returns: None

When called this function will `sys.exit()`

crassh.**readauthfile**(*filepath*)
Read C.R.A.SSH Authentication File

The file format is a simple, one entry per line, colon separated affair:

```
username: nick
password: cisco
```

Args: filepath (str): Full path to file

Returns: tuple. `username` and `password`

Example:

```
>>> username, password = readauthfile("~/.crasshrc")
>>> print(username)
nick
>>> print(password)
cisco
```

crassh.**readtxtfile**(*filepath*)

Read lines of a text file into an array Each line is stripped of whitepace.

Args: filepath (str): Full path to file

Returns: array. Contents of file

Example:

```
>>> print(readtxtfile("./routers.txt"))
1.1.1.1
1.1.1.2
1.1.1.3
```

crassh.**send_command**(*command='show ver'*, *hostname='Switch'*, *bail_timeout=60*)

Sending commands to a switch, router, device, whatever!

Args: command (str): The Command you wish to run on the device.

hostname (str): The hostname of the device (*expected in the* `prompt`).

bail_timeout (int): How long to wait for `command` to finish before giving up.

Returns: str. A text blob from the device, including line breaks.

REF: http://blog.timmattison.com/archives/2014/06/25/automating-cisco-switch-interactions/

C

C

D

I

M

P

R

S