Module Eleven:

Port Scanner & SSH Worm

Grading Criteria

Total Points: 45 points

☐ **[5 points]** Your submission is labeled as "cyb404_module11_lab00_[nau_id]_[lastname]_[firstname].zip." For example, if I were to submit a file, it would be labeled as cyb404_module11_lab00_mv668_vigil-hayes_morgan.zip. **FAILURE TO COMPLY WITH THIS STEP CAN LEAD TO A ZERO GRADE.**

☐ **[5 points]** Your submission files contained in the ZIP are in the correct formats as specified.

- ☐ [1 points] The port scanner is all contained in a file port_scanner.py
- ☐ [1.5 points] There is a comment for every functional unit of code that details the purpose of the port scanner code.
- ☐ [1 points] SSH worm is all contained in a file ssh_worm_brute.py
- ☐ [1.5 points] There is a comment for every functional unit of code that details the purpose of the SSH worm code.

☐ **[15 points]** Code correctly scans a list of ports and hostnames and uses threading to do so more efficiently

- ☐ [2.5 points] File runs using command: python3 port_scanner.py -H <target host> -p <target port>
- ☐ [2.5 points] There are no runtime errors
- ☐ [5 points] Screenshot of some of the banner TCP packets prompted by your scanner
- ☐ [5 points] Screenshot of terminal output of your scanner

☐ **[20 points]** Code correctly starts an SSH session with pxssh and tries to brute force the password.

- ☐ [2.5 points] File runs using command: python3 ssh_worm_brute.py
- ☐ [2.5 points] There are no runtime errors
- ☐ [5 points] Screenshot of the script taking control of the remote host with the telnet command
- ☐ [5 points] Screenshot of the terminal output of python3 ssh_worm_brute.py
- ☐ [5 points] Screenshot of the packets generated when running python3 ssh_worm_brute.py

One of the most common attack pathways is to scan different machines for open ports, identify which services are running on different ports, and then launch an attack against those services. In this

assignment, you will be writing a simple port scanner and then writing a simple SSH worm that can take control of a remote machine through SSH and brute force an SSH password.

In the Module Twelve assignment, you will be extending this code to attack weak SSH keys and launch a botnet.

Part 1: Creating a port scanner

Reconnaissance serves as the first step in any good cyber assault. An attacker must discover where the vulnerabilities are before selecting and choosing exploits for a target. In the following section, we will build a small reconnaissance script that scans a target host for open TCP ports. However, in order to interact with TCP ports, we will need to first construct TCP sockets.

Python, like most modern languages, provides access to the BSD socket inter- face. BSD sockets provide an application-programming interface that allows coders to write applications in order to perform network communications between hosts. Through a series of socket API functions, we can create, bind, listen, connect, or send traffic on TCP/IP sockets. At this point, a greater under- standing of TCP/IP and sockets are needed in order to help further develop our own attacks.

The majority of Internet accessible applications reside on the TCP. For example, in a target organization, the web server might reside on TCP port 80, the email server on TCP port 25, and the file transfer server on TCP port 21. To connect to any of these services in our target organization, an attacker must know both the Internet Protocol Address and the TCP port associated with the service. While someone familiar with our target organization would probably have access to this information, an attacker may not.

An attacker routinely performs a port scan in the opening salvo of any successful cyber assault. One type of port scan includes sending a TCP SYN packet to a series of common ports and waiting for a TCP ACK response that will result in signaling an open port. In contrast, a TCP Connect Scan uses the full three-way handshake to determine the availability of the service or port.

TCP Full Connect Scan

You will begin by writing your own TCP port scanner that utilizes a TCP full connect scan to identify hosts. To begin, you will import the Python implementation of BSD socket API (what was used to implement the basic web server in earlier modules). The socket API provides us with some functions that will be useful in implementing our TCP port scanner. Let's examine a couple before proceeding. For a deeper understanding, view the Python Standard Library Documentation at: http://docs.Python.org/library/socket.html.

```
socket.gethostbyname(hostname) – This function takes a hostname
such as www.syngress.com and returns an IPv4 address format such
as 69.163.177.2.
```

socket.gethostbyaddr(ip address) – This function takes an IPv4 address and returns a triple containing the hostname, alternative list of host names, and a list of IPv4/v6 addresses for the same interface on the host.

socket.socket([family[, type[, proto]]]) – This function creates an instance of a new socket given the family. Options for the socket family are AF_INET, AF_INET6, or AF_UNIX. Additionally, the socket can be specified as SOCK_STREAM for a TCP socket or SOCK_DGRAM for a UDP socket. Finally, the protocol number is usually zero and is omitted in most cases.

socket.create_connection(address[, timeout[, source_address]]) – This function takes a 2-tuple (host, port) and returns an instance of a network socket. Additionally, it has the option of taking a timeout and source address.

If you look at port_scanner_starter.py, you can see that the scanner is broken into five unique steps. You will write Python code for each of them.

1. First, you will input a hostname and a comma separated list of ports to scan.
2. Next, you will translate the hostname into an IPv4 Internet address.
3. For each port in the list, you will also connect to the target address and specific port.
4. To determine the specific service running on the port, you will send garbage data and
5. Read the banner results sent back by the specific application.

After testing the port scanner, you might note that it takes a few seconds to scan each connection.

To create a more efficient scanner, implement Python threading in the port scanner.

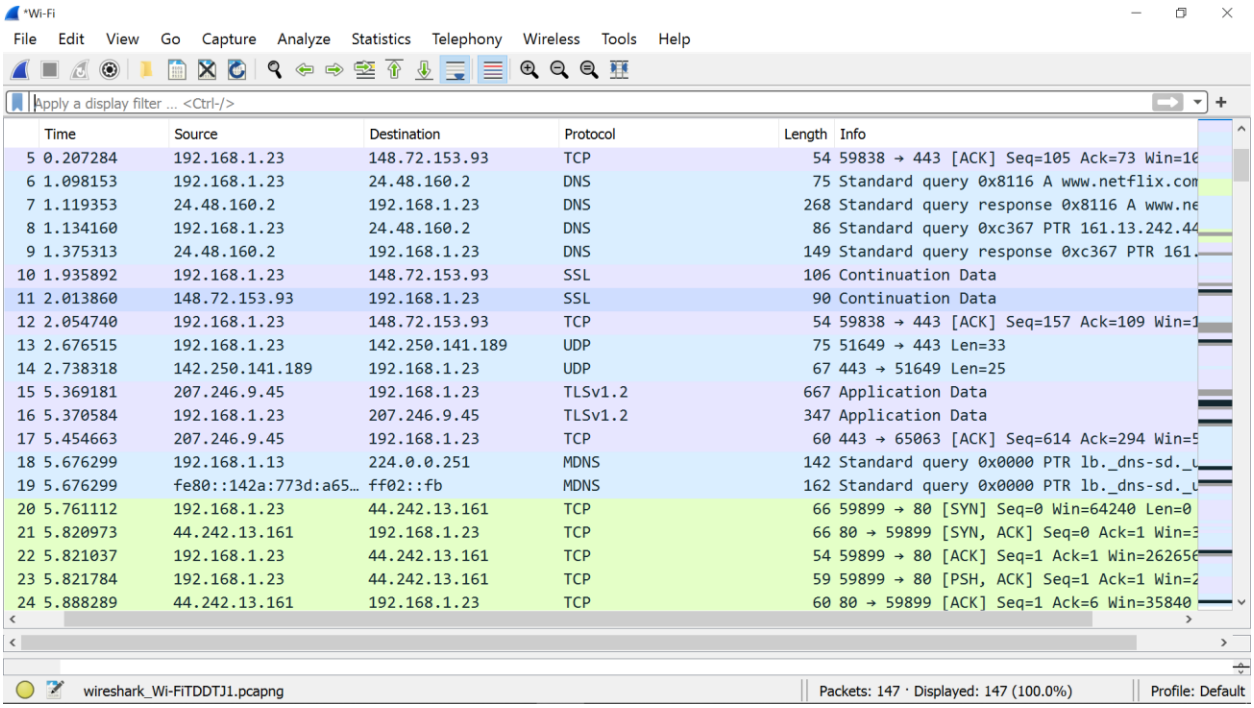
Threading provides a way to perform these kinds of executions simultaneously. To utilize this in our scan, we will modify the iteration loop in our portScan() function. Notice how we call the connScan function as a thread. Each thread created in the iteration will now appear to execute at the same time.

Take a screenshot of your terminal output that results from your port scan over a list of targets that includes at least 5 hostnames and 10 ports for each. You can reuse the port numbers for each target hostname.

```
PS C:\Users\km3248\documents\assignment11> python3 port_scanner_starter.py -H "www.amazon.com" -p "80, 67, 54, 84, 35, 59, 65, 72, 50, 21"

[+] Scan Results for: a23-55-249-131.deploy.static.akamaitechnologies.com
Scanning port 80
[+]80/tcp open
[-]80/tcp closed
Scanning port 67
[-]67/tcp closed
Scanning port 54
[-]54/tcp closed
Scanning port 84
[-]84/tcp closed
Scanning port 35
[-]35/tcp closed
Scanning port 59
[-]59/tcp closed
Scanning port 65
Scanning port 72
[-]72/tcp closed
Scanning port 50
[-]50/tcp closed
Scanning port 21
[-]21/tcp closed



PS C:\Users\km3248\documents\assignment11> python3 port_scanner_starter.py -H "www.gmail.com" -p "80, 67, 54, 84, 35, 59, 65, 72, 50, 21"

[+] Scan Results for: nuq04s39-in-f5.1e100.net
Scanning port 80
[+]80/tcp open
[-]80/tcp closed
Scanning port 67
[-]67/tcp closed
Scanning port 54
[-]54/tcp closed
Scanning port 84
[-]84/tcp closed
Scanning port 35
[-]35/tcp closed
Scanning port 59
[-]59/tcp closed
Scanning port 65
[-]65/tcp closed
Scanning port 72
[-]72/tcp closed
Scanning port 50
[-]50/tcp closed
Scanning port 21
[-]21/tcp closed

PS C:\Users\km3248\documents\assignment11> python3 port_scanner_starter.py -H "www.microsoft.com" -p "80, 67, 54, 84, 35, 59, 65, 72, 50, 21"

[+] Scan Results for: a23-55-249-185.deploy.static.akamaitechnologies.com
Scanning port 80
[+]80/tcp open
[-]80/tcp closed
Scanning port 67
[-]67/tcp closed
Scanning port 54
[-]54/tcp closed
Scanning port 84
[-]84/tcp closed
Scanning port 35
[-]35/tcp closed
Scanning port 59
[-]59/tcp closed
Scanning port 65
Scanning port 72
[-]72/tcp closed
Scanning port 50
[-]50/tcp closed
Scanning port 21
[-]21/tcp closed
```

```
PS C:\Users\km3248\documents\assignment11> python3 port_scanner_starter.py -H "www.netflix.com" -p "80, 67, 54, 84, 35, 59, 65, 72, 50, 21"

[+] Scan Results for: ec2-44-237-234-25.us-west-2.compute.amazonaws.com
Scanning port 80
[+]80/tcp open
[-]80/tcp closed
Scanning port 67
[-]67/tcp closed
Scanning port 54
[-]54/tcp closed
Scanning port 84
[-]84/tcp closed
Scanning port 35
[-]35/tcp closed
Scanning port 59
[-]59/tcp closed
Scanning port 65
[-]50/tcp closed
Scanning port 21
[-]21/tcp closed


PS C:\Users\km3248\documents\assignment11>

                                        python3 port_scanner_starter.py -H "www.yahoo.com" -p "80, 67, 54, 84, 35, 59, 65, 72, 50, 21"

[+] Scan Results for: media-router-fp74.prod.media.vip.ne1.yahoo.com
Scanning port 80
[+]80/tcp open
[-]80/tcp closed
Scanning port 67
[-]67/tcp closed
Scanning port 54
[-]54/tcp closed
Scanning port 84
[-]84/tcp closed
Scanning port 35
[-]35/tcp closed
Scanning port 59
[-]59/tcp closed
Scanning port 65
Scanning port 72
[-]72/tcp closed
Scanning port 50
[-]50/tcp closed
Scanning port 21
[-]21/tcp closed
```

While your port scanner is running, run Wireshark to capture the packets that are being sent back and forth. Take a screenshot that shows a deep dive into a packet that contains the banner information from one of the services you have scanned.



Part 2: Taking control of an SSH session

Now that we have constructed a port scanner to find targets we can begin the task of exploiting the vulnerabilities of each service. The Morris Worm includes forcing common usernames and passwords against the remote shell (RSH) service as one of its three attack vectors. In 1988, RSH provided an excellent (although not very secure) method for a system administrator to remotely connect to a machine and manage it by performing a series of terminal commands on the host. The Secure Shell (SSH) protocol has since replaced RSH by combining RSH with a public-key cryptographic scheme in order to secure the traffic. However, this does very little to stop the same attack vector by forcing out common user names and passwords. SSH Worms have proven to be very successful and common attack vectors.

You will implement your own automated SSH Worm that brute forces user credentials against a target. Because SSH clients require user interaction, our script must be able to wait and match for an expected output before sending further input commands. Consider the following scenario. In order to connect to our SSH machine at IP Address, 127.0.0.1, the application first asks us to confirm the RSA key fingerprint. In this case, we must answer, “yes” before continuing. Next, the application asks us to enter a password before granting us a command prompt. Finally, we execute our command uname –v to determine the kernel version running on our target.

```
attacker$ ssh root@127.0.0.1

The authenticity of host '127.0.0.1 (127.0.0.1)' can't be
established.
```

```
RSA key fingerprint is
5b:bd:af:d6:0c:af:98:1c:1a:82:5c:fc:5c:39:a3:68.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added '127.0.0.1' (RSA) to the list of known
hosts.

Password:**************

Last login: Mon Oct 17 23:56:26 2011 from localhost

attacker:~ uname -v

Darwin Kernel Version 11.2.0: Tue Aug 9 20:54:00 PDT 2011;
root:xnu-1699.24.8~1/RELEASE_X86_64
```

In order to automate this interactive console, we will make use of a third party Python module named Pexpect (available to download at http://pexpect. sourceforge.net). Pexpect has the ability to interact with programs, watch for expected outputs, and then respond based on expected outputs. This makes it an excellent tool of choice for automating the process of brute forcing SSH user credentials.

Open the provided starter code and examine the function connect(). This function takes a username, hostname, and password and returns an SSH connection resulting in an SSH spawned connection. Utilizing the pexpect library, it then waits for an expected output. Three possible expected outputs can occur—a timeout, a message indicating that the host has a new public key, or a password prompt. If a timeout occurs, then the session.expect() method returns to zero. The following selection statement notices this and prints an error message before returning. If the child.expect() method catches the ssh_newkey message, it returns a 1. This forces the function to send a message 'yes' to accept the new key. Following this, the function waits for the password prompt before sending the SSH password.

Running the script ssh_worm_starter.py, we see we can connect to an SSH server to remotely control a host. While we ran the simple command to display the hashed password for the root user from /etc/shadow file (see below), we could use the tool to something more devious like using wget to download a post exploitation toolkit.

You can start an SSH server on Kali by generating ssh-keys and then starting the SSH service. Try starting the SSH server and connecting to it with the script.

```
attacker# sshd-generate

Generating public/private rsa1 key pair.
```

```
<..SNIPPED..>

attacker# service ssh start

ssh start/running, process 4376

attacker# python3 ssh_worm_starter.py

cat /etc/shadow | grep root

root:$6$ms32yIGN$NyXj0YofkK14MpRwFHvXQW0yvUid.slJtgxHE2EuQqgD74S/
GaGGs5VCnqeC.bS0MzTf/EFS3uspQMNeepIAc.:15503:0:99999:7:::
```

It is possible to simplify the code used in ssh_worm_starter.py using a specialized Python library, pxssh. While the starter script help give you a deep understanding of the capabilities of pexpect, you can really simplify the previous script using pxssh. Pxssh is a specialized script included the pexpect library. It contains the ability to directly interact with SSH sessions with pre-defined methods for login(), logout(), prompt().

https://pexpect.readthedocs.io/en/stable/api/pxssh.html

For your next step in this assignment, you will modify the ssh_worm_starter.py script to use pxssh instead of pexpect.

Demonstrate your ability to take control of the SSH session by running:

```
telnet towel.blinkenlights.nl
```

Take a screenshot of your script being run from the commandline and the result of running the telnet command.

[SCREENSHOT GOES HERE]

Part 3: Brute forcing the password

Now that you have simplified the starter script, the worm is near complete. You will only need to make a few minor modifications to get the script to automate the task of brute forcing SSH credentials. Other than adding some option parsing to read in the hostname, username, and password file, the only thing you need to do is slightly modify the connect() function:

If the login() function succeeds without exception, you will print a message indicating that the password is found and update a global Boolean indicating so.

Otherwise, you will catch the exception. If the exception indicates that the password was 'refused', you know the password failed and you should just return.

However, if the exception indicates that the socket is 'read_non- blocking', then you should assume the SSH server is maxed out at the number of connections, and you should sleep for a few seconds before trying again with the same password.

Additionally, if the exception indicates that pxssh is having difficulty obtaining a command prompt, you will sleep for a second to allow it to do so.

You should include a Boolean release included in the connect() function arguments. Since connect() can recursively call another connect(), you only want the caller to be able to release your connection_lock semaphore.

```
connection_lock = BoundedSemaphore(value=maxConnections)

def connect(host, user, password, release):

      < …SNIPPED… >

      finally:

            if release: connection_lock.release()

< …SNIPPED… >

def main():

      < …SNIPPED… >

      connection_lock.acquire()

      < …SNIPPED… >
```

Now that you have the outline for the brute force version of the worm, copy ssh_worm_starter.py to ssh_worm_brute.py and make the modifications necessary to brute force a password attack.

When you have successfully written your script to launch the brute force login, take a screenshot of the terminal output when you run the script (this will require you insert some

print() statements detailing the progress of your script). Turn on Wireshark and capture packets that are involved in your connection to the SSH server. Include screenshots of the terminal output and captured packets here:

[SCREENSHOT GOES HERE]

[SCREENSHOT GOES HERE]