Scripting for Cybersecurity

Lab 8

Port Scanning

Introduction

Port scanning allows an attacker to gain information about the services running on a host. Port numbers are used as addressing at the transport layer. A port is open when an application is listening on that port. This lab will explore port scanning with Scapy, as well as Python switch statements and Sockets.

Task 1 – It's a SYN

Create a script "port_scan.py" containing the following code. This script will be a tool we can use to scan ports of a target host. The tool will take in an interface name, a target IP address, and one or more ports to scan. Port numbers should be provided with a comma separating them. We'll use it like ./port_scan.py -i enp0s3 -t 10.0.0.4 -p 22,23,25,80

```
#!/usr/bin/python3

from scapy.all import *
from time import sleep
import sys

def scan_port(iface, target_ip, target_port):
  pass

def help():
  print("Example usage:")
  print("./port_scan.py -i eth0 -t 10.0.0.1 -p 23")
  print("./port_scan.py -i enp0s3 -t 10.0.0.1 -p 22,80,555")

def main():
  args = sys.argv
  if(("-i" in args) and ("-t" in args) and ("-p" in args) and (len(args) == 7)):
    try:
      iface = args[args.index("-i")+1]
      target_ip = args[args.index("-t")+1]
      target_ports = args[args.index("-p")+1]
    except:
      help()
      exit()
  else:
    help()
    exit()
main()
```

The above code

1) Reads the command line arguments into a variable *args*
2) Checks that the required parameters (-i, -t, and -p) have been provided as well as whether it looks like values have been provided with them (args should have 7 items -> Script name, 7 "-i", iface, "-t", target ip, "-p", target ports.) If the required values are not provided then we call the help function and exit
3) Tries to read the values from the args list. Wrapped in a try except in case any issues arise. If something goes wrong we call the help function and exit
4) A help function is defined with some example usage for the script
5) A function has been defined for the port scanning which we'll use to perform a SYN scan. In this function we'll send a TCP SYN and return the flags of the response (e.g. SYN-ACK or RST).

The script will take in one or more port numbers to scan. As seen in the example usage, the provided port numbers should be separated using a comma. Lets add some code to take the port numbers and separate them into a list.

```
  else:
    help()
    exit()

  ports = target_ports.split(",")
  print(ports)


main()
```

The targets ports variable is a string and we know that each port is separated with a comma, so we should be able to call the .split() function on the string to create a list of port numbers. Running the script with the required command line arguments should now print out a list containing one or more ports to scan.

```
ubuntu@ubuntu-VirtualBox:~/my_scripts$ ./port_scan.py -i enp0s3 -t 10.0.0.1 -p 22,23,80,8000
['22', '23', '80', '8000']
```

This will also work fine for us if we only provide a single port. If there's no commas to split on the .split() function will just return a list containing nothing but the original string.

```
ubuntu@ubuntu-VirtualBox:~/my_scripts$ ./port_scan.py -i enp0s3 -t 10.0.0.1 -p 22
['22']
```

Next, as we want to scan each of these ports we'll add in a for loop containing a call to the scan_port function. The loop will iterate through each of the items in our ports list and pass in the appropriate values to the function call.

```
ports = target_ports.split(",")

for p in ports:
  resp = scan_port(iface, target_ip, int(p))
  if("R" in resp):
    print(p + "\tclosed")
  elif("SA" in resp):
    print(p + "\topen")
  else:
    print(p + "\tno resp.")
```

In the above code, we replace the print() function call with the loop. The scan_port function will scan the target port we provide it with. The scan_port function will send a TCP SYN and return the flags within the result. As this is a SYN scan, if the target host returns a message with the RA flags (RST & ACK) set then the port is closed. If the response contains the SA flags (SYN & ACK) then the port is open. The loop contains the code to check if the RST flag is in the response, and if it is then we print that the port is closed, otherwise if it contains the SYN-ACK flags then the port is open. The "\t" in the print statement will have the terminal insert a tab before the strings "closed" and "open". If no response comes back then we'll print "no resp.". Also note that we cast the string *p* to an int as Scapy will expect the ort number to be an int.

Next, we'll add code to the scan_port function. In this function we need to

- Create the IP header containing the target destination address
- Create the TCP header containing the target port and SYN flags
- Build and send the packet
- Receive the response and return the flags within the response
- If no response then return an empty string

The code below implements the above functionality:

```
def scan_port(iface, target_ip, target_port):
  ip_hdr = IP(dst=target_ip)
  tcp_hdr = TCP(dport=target_port, flags="S")
  pkt = ip_hdr/tcp_hdr
  resp, unans = sr(pkt, iface=iface, timeout=2)
  if(len(resp) > 0):
    return str(resp[0][1][TCP].flags)
  else:
    return ""
```

We set a timeout in the sr() function to limit the amount of time Scapy will wait for a response.

We'll also add the following line to the script to supress output from Scapy when sending packets:

```
import sys

conf.verb = 0

def scan_port(iface
```

At this point the script is a functional port scanner. Navigate to the mininet directory from lab 7 and start up Mininet using the following command:

sudo mn --topo single,4 --test pingall --post cmds.txt

In the Attacker terminal window, navigate to the my_scripts directory and use the tool to scan the port 22, 23, 80, 8080, 8000 on the Server host.

```
root@ubuntu-VirtualBox:/home/ubuntu/my_scripts# ./port_scan.py -i h1-eth0 -t 10.0.0.4 -p 22,23,80,8080,8000
22      closed
23      open
80      closed
8080    closed
8000    closed
```

Run the following command in the Server terminal window to start the Python HTTP server. This server will be listening on port 8000.

```
                                        Server
root@ubuntu-VirtualBox:/home/ubuntu/mininet# python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Scan the ports from the Attacker terminal again. Port 8000 should not appear as “open”.

Task 2 – Switch doctor

Nmap, a well-known port scanner, will output the service that it detects running on a port. We can implement some similar functionality in our script.

We’ll create a function to do this. This function will take a port number and map it to the service that port number is known for. We’ll create a mapping in our function using a switch statement. The current full list of registered port number to service mappings can be found here:

https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

While there is no requirement for services to run on any specific port, the standardized mapping above means that we can assume the service based off the port number (e.g. If port 23 is open then it’s most likely Telnet, etc).

Add the following function to the script between the help() and the scan_port() function:

```
def get_port_service(port):
  try:
    service = {
      22 : "SSH",
      23 : "Telnet",
      80 : "HTTP",
      443: "HTTPS",
      8080 : "HTTP alt",
      8000 : "HTTP alt"
    }[port]
    return service
  except KeyError:
    return "?"
```

Modify the loop in the main function to be the following:

```
for p in ports:
  resp = scan_port(iface, target_ip, int(p))
  service = get_port_service(int(p))
  if("R" in resp):
    print(p + "\tclosed\t" + service)
  elif("SA" in resp):
    print(p + "\topen\t" + service)
  else:
    print(p + "\tno resp.\t" + service)
```

Run the script in the Attacker terminal window again and now you should receive some more helpful output from the script:

```
root@ubuntu-VirtualBox:/home/ubuntu/my_scripts# ./port_scan.py -i h1-eth0 -t 10.0.0.4 -p 22,23,80,8080,8000
22      closed  SSH
23      open    Telnet
80      closed  HTTP
8080    closed  HTTP alt
8000    open    HTTP alt
```

Task 3 – PSH it

The current script can only perform a TCP SYN scan. We can extend it to include other scans, such as the FIN, NULL, and XMAS scans. Including these other types of scans is a straight forward process as it only involves modifying the TCP flags in the message sent to the target.

Modify the main function to accept the following command line arguments

- -S -> SYN scan
- -F -> FIN scan
- -X -> XMAS scan
- -N -> NULL scan

As we're including additional parameters for our script, we'll first have to change the following line:

```
if(("-i" in args) and ("-t" in args) and ("-p" in args) and (len(args) >= 7)):
```

In the above (len(args)) == 7 has been changed to (len(args) >=7.

Add the following if else statements to the main function and modify the call to the scan_port function as shown in the image:

```
flags = "S"
if("-F" in args):
  print("[+] Performing FIN scan")
  flags = "F"
elif("-N" in args):
  print("[+] Performing NULL scan")
  flags = ""
elif("-X" in args):
  print("[+] Performing XMAS scan")
  flags = "UPF"
else:
  print("[+] Performing SYN scan")

ports = target_ports.split(",")

for p in ports:
  resp = scan_port(iface, target_ip, int(p), flags)
  service = get_port_service(int(p))
```

In the above, a new variable *flags* is added. This variable is set depending on the type of scan selected by the user. The flags variable is passed to the scan_port function when it is called in the loop.

The scan_port function will not have to be modified to accept the flags variable and use this when building the TCP header. The necessary modifications can be seen below:

```
def scan_port(iface, target_ip, target_port, flags):
  ip_hdr = IP(dst=target_ip)
  tcp_hdr = TCP(dport=target_port, flags=flags)
```

Before we can use the new scanning methods, we first need to modify the code which tells us whether a port is open or closed. At the moment this code will only report an open port for a SYN scan, as t checks for a returning SYN-ACK message. With a FIN scan, a NULL scan, and an XMAS scan we are expecting to receive no returning message if the port is open. The following code takes this into account and will correctly report an open or closed port.

```
if("R" in resp):
  print(p + "\tclosed\t" + service)
elif(("SA" in resp) or (resp == "" and flags != "S")):
  print(p + "\topen\t" + service)
else:
  print(p + "\tno resp.\t" + service)
```

The above modification will report an open port if

- A SYN-ACK is received or
- No response is received and the flags are not “S”

In other words, if we get a SYN-ACK or we get no response and we are not using a SYN scan.

Try scanning the ports on the server with each of the added scanning techniques. For the FIN, XMAS, and NULL scans you’ll notice a delay before the open port is reported. This is our 2 second timeout while waiting for a response. For faster results you could reduce the timeout.

```
root@ubuntu-VirtualBox:/home/ubuntu/my_scripts# ./port_scan.py -i h1-eth0
 -t 10.0.0.4 -p 22,23,80,8080,8000 -X
[+] Performing XMAS scan
22      closed  SSH
23      open    Telnet
80      closed  HTTP
8080    closed  HTTP alt
8000    open    HTTP alt
```

Task 4 – Socket man

Next, we'll add another function to perform a connect scan.

A connect scan establishes a full TCP connection with the target port. This type of scan can be useful as you can use it to perform *banner grabbing***.** Banner grabbing involves gaining additional information about the services running on a port. It can be a more accurate way of determining the services running on a host.

The best way for us to perform a connect scan in Python is to use Sockets. We can create a socket connection, and use this connection to send and receive data.

```
>>> import socket
>>> sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>>> sock.connect(("10.0.0.4", 23))
>>> sock.close()
>>> sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>>> sock.connect(("10.0.0.4", 666))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ConnectionRefusedError: [Errno 111] Connection refused
>>>
```

The image above shows sockets being used in the Python interactive shell. First, the socket library is imported. Next, a socket object is created and is made accessible through the *sock* variable. A connection is attempted and established to 10.0.0.4 on port 23. We know this connection is successful as no errors occurred. The socket is then closed.

After the first connection, a new socket object is created. Another connection is attempted, this time to 10.0.0.4 on port 666. This connection is not successful as the port is closed, and we receive a ConnectionRefusedError.

We can use code similar to the above to implement the connect scan. If the connection is refused then the port is closed, otherwise the port is open. Import the socket library at the top of your script and then add the following function blow the scan_port function:

```
def connect_scan(target_ip, target_port):
  data = ""
  try:
    sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    sock.settimeout(1)
    sock.connect((target_ip, target_port))
    data = sock.recv(1024)
    sock.close()
  except ConnectionRefusedError:
    return ("R", data)
  except:
    pass
  return ("", data)
```

This function has some additional code to the code seen in the previous image. Firstly, we're calling sock.recv(1024) to receive 1024 Bytes of data after we connect to the target IP and port. The

sock.settimeout(1) line will set a timeout for this to 1 second, so we aren't waiting for data that will never arrive. Reading the data after we make a connection allows us to grab the banner from the service. This works well with some services, such as SSH, and not so well with others which may not return any data without an initial request.

Modify your code so that a user can pass in "-C" when unning the program to indicate that they want to use a connect scan. Your program should also print out any data received during the connect scan. The output should look like this: