Brooklyn Dressel
Assignment 6

Abstract

This assignment sets up a honeypot server in a virtual machine by disabling the firewall. We then proceed to attack this from a separate virtual machine over a network using five different network services. These were HTTP, FTP, SSH, SMTP, and Telnet traffic. All communications and attackers were recorded in logs by the honeypot server. In addition to this we wrote two python scripts to create two more, more sophisticated imitations of network services, HTTP on port 8080 and an echo server on port 9009. From these we can tell the breadth and depth of information a honey pot server is able to record, exposing attackers ip addresses and methods in plaintext.

Introduction

This assignment will use an Ubuntu virtual machine with a disabled firewall to play the part of the honeypot. A Kali Linux virtual machine will play the role of the attacker. Netcat listeners will be set up on Ubuntu and log all activities on five separate types of traffic (ports). WireShark will run on the Kali machine and capture all network traffic to be analyzed after the attacks. Kali will connect to the Ubuntu machine using a Firefox web browser to generate HTTP traffic (port 80), FTP traffic (port 21), a SSH connection (port 22), SMTP traffic (port 25), and finally Telnet traffic (port 23). Additionally, two python scripts were used to simulate A list of all commands used during this assignment is attached separately.

Summary of Results

The initial step in this assignment was to boot up a Kali and an Ubuntu virtual machine, configured to a Bridged Network to allow for communication between the virtual machines. Upon startup, `command 1` to find the respective ip addresses of these virtual machines, I then ran `commands 2 and 3` and to confirm the two could communicate. I confirmed the pings were successful and terminated the pings.

```
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP gr
oup default qlen 1000
    link/ether 08:00:27:c6:4c:93 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.228/24 brd 10.0.0.255 scope global dynamic noprefixroute eth0
       valid_lft 172779sec preferred_lft 172779sec
    inet6 2601:642:4f7f:c880::dcf4/128 scope global dynamic noprefixroute
       valid_lft 7180sec preferred_lft 7180sec
    inet6 2601:642:4f7f:c880:39a5:2c95:ba8e:10ee/64 scope global dynamic nopre
fixroute
       valid_lft 298sec preferred_lft 298sec
    inet6 fe80::b769:c00:3272:481f/64 scope link noprefixroute
       valid_lft forever preferred_lft forever

┌──(kali㉿kali)-[~]
└─$ ping 10.0.0.71
PING 10.0.0.71 (10.0.0.71) 56(84) bytes of data.
64 bytes from 10.0.0.71: icmp_seq=1 ttl=64 time=0.491 ms
64 bytes from 10.0.0.71: icmp_seq=2 ttl=64 time=0.650 ms
64 bytes from 10.0.0.71: icmp_seq=3 ttl=64 time=0.864 ms
64 bytes from 10.0.0.71: icmp_seq=4 ttl=64 time=0.740 ms
64 bytes from 10.0.0.71: icmp_seq=5 ttl=64 time=0.441 ms
64 bytes from 10.0.0.71: icmp_seq=6 ttl=64 time=0.372 ms
64 bytes from 10.0.0.71: icmp_seq=7 ttl=64 time=0.540 ms
64 bytes from 10.0.0.71: icmp_seq=8 ttl=64 time=0.609 ms
64 bytes from 10.0.0.71: icmp_seq=9 ttl=64 time=0.841 ms
64 bytes from 10.0.0.71: icmp_seq=10 ttl=64 time=0.741 ms
64 bytes from 10.0.0.71: icmp_seq=11 ttl=64 time=0.472 ms
64 bytes from 10.0.0.71: icmp_seq=12 ttl=64 time=0.643 ms
64 bytes from 10.0.0.71: icmp_seq=13 ttl=64 time=1.25 ms
^C
```

```
brooklyn@brooklyn-VirtualBox:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fa
st state UP group default qlen 1000
    link/ether 08:00:27:23:b8:55 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.71/24 brd 10.0.0.255 scope global dynamic noprefixro
ute enp0s3
       valid_lft 172776sec preferred_lft 172776sec
    inet6 2601:642:4f7f:c880:dc7d:b407:68bb:f739/64 scope global tem
porary dynamic
       valid_lft 300sec preferred_lft 300sec
    inet6 2601:642:4f7f:c880:a00:27ff:fe23:b855/64 scope global dyna
mic mngtmpaddr
       valid_lft 300sec preferred_lft 300sec
    inet6 fe80::a00:27ff:fe23:b855/64 scope link
       valid_lft forever preferred_lft forever
brooklyn@brooklyn-VirtualBox:~$ ping 10.0.0.228
PING 10.0.0.228 (10.0.0.228) 56(84) bytes of data.
64 bytes from 10.0.0.228: icmp_seq=1 ttl=64 time=0.856 ms
64 bytes from 10.0.0.228: icmp_seq=2 ttl=64 time=0.330 ms
64 bytes from 10.0.0.228: icmp_seq=3 ttl=64 time=0.345 ms
64 bytes from 10.0.0.228: icmp_seq=4 ttl=64 time=0.729 ms
64 bytes from 10.0.0.228: icmp_seq=5 ttl=64 time=0.956 ms
64 bytes from 10.0.0.228: icmp_seq=6 ttl=64 time=0.982 ms
```

In Ubuntu, I ran `command 4` to stop any previous running ssh services, to ensure a clean slate. I had to run `commands 5 and 6` to close the socket service as well, and confirmed that all services were terminated using `command 7`. Finally, to make this a true honeypot I ran `command 8` in order to disable my firewall and make the Ubuntu machine vulnerable to attacks and foreign connections. I confirmed this using `command 9`.

```
brooklyn@brooklyn-VirtualBox:~$ sudo systemctl stop ssh
[sudo] password for brooklyn:
Stopping 'ssh.service', but its triggering units are still active:
ssh.socket
brooklyn@brooklyn-VirtualBox:~$ sudo systemctl stop ssh.socket
brooklyn@brooklyn-VirtualBox:~$ sudo systemctl disable ssh.socket
Removed "/etc/systemd/system/sockets.target.wants/ssh.socket".
Removed "/etc/systemd/system/ssh.service.requires/ssh.socket".
brooklyn@brooklyn-VirtualBox:~$ sudo lsof -i :22
brooklyn@brooklyn-VirtualBox:~$ sudo ufw disable
Firewall stopped and disabled on system startup
```

The next part of the assignment required Netcat listeners on Ubuntu to catch all traffic and log it. At first I ran into many problems getting the listener set up correctly. The first problem I encountered was needing to run the command as root, using sudo. Then after a small typo, and a few adjustments to ensure traffic from the attacker as well as from the honeypot were logged. Finally settled on `commands 10-14`, which set up the listeners for HTTP, FTP, SSH, SMTP, and Telnet traffic respectively. These commands also sent fake replies to the attacker, intended to look (relatively) real so as to not raise the attacker's suspicion.

```
nc: Permission denied
nc: Permission denied
nc: Permission denied
^C

brooklyn@brooklyn-VirtualBox:~$ sudo bash -c 'while true; echo -c "H
TTP/1.1 200 OK\n\n $(date)" | nc -l -p 80 -q 1 >> http.log; done'
bash: -c: line 1: syntax error near unexpected token `done'
bash: -c: line 1: `while true; echo -c "HTTP/1.1 200 OK\n\n $(date)"
 | nc -l -p 80 -q 1 >> http.log; done'
brooklyn@brooklyn-VirtualBox:~$ sudo bash -c 'while true; do echo -c
 "HTTP/1.1 200 OK\n\n $(date)" | nc -l -p 80 -q 1 >> http.log; done'
```

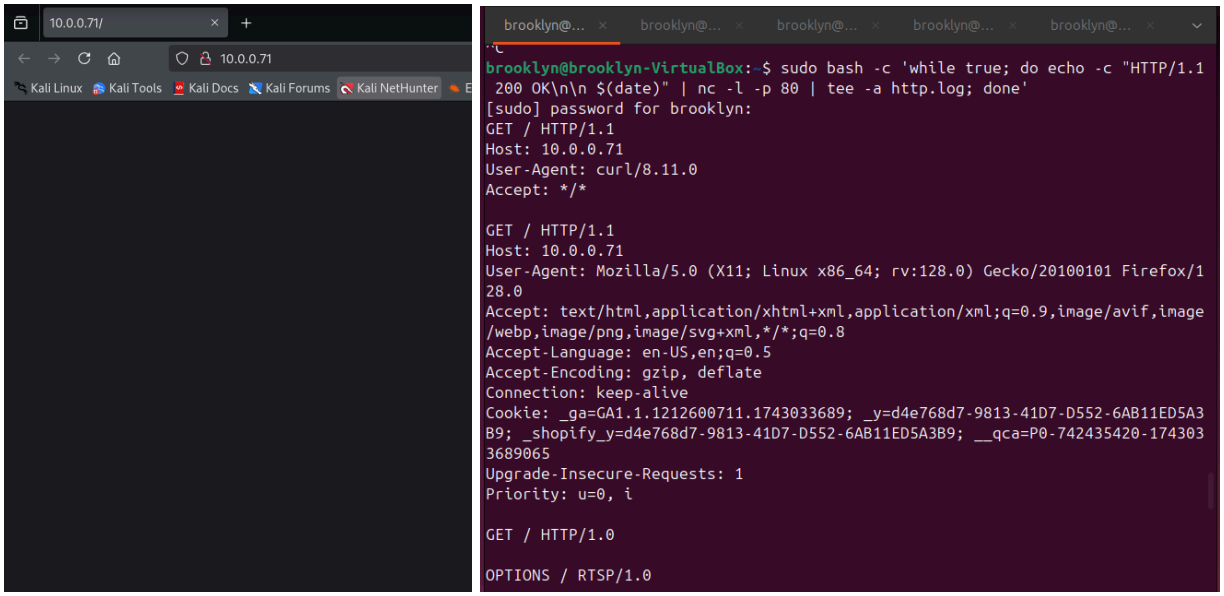
```
brooklyn@brooklyn-VirtualBox:~$ sudo bash -c 'while true; do echo -e
to the Telnet server" | nc -l -p 23 | tee -a telnet.log; done'
[sudo] password for brooklyn:
help
ls
uname -a
]
◆◆◆◆

◆◆% NTLMSSP◆◆◆SO?G◆◆◆,◆◆`~◆◆◆{◆$◆◆w◆◆◆◆<=◆o◆n(
fedcba`◆(4◆◆R◆◆
```

Now that all of the listeners and Ubuntu side of these assignments, it was time to begin the 'attack' on the Kali side of things. In order to connect to the five services being hosted on the Ubuntu machine, I used `commands 16-21` to achieve this, once connected to the services I filled the connection with random traffic to discover it in the logs as well as WireShark. As an additional attack I ran a scan of the Ubuntu machine using `command 22`. As shown in the image below on the left, this scan revealed the five open ports running on the Ubuntu machine, posts 22, 23, 25, 21, and 80.

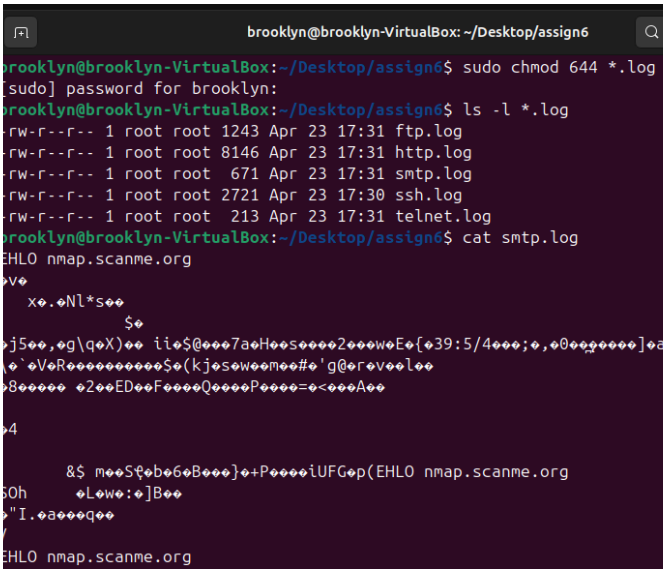
```
┌──(kali㉿kali)-[~]
└─$ curl http://10.0.0.71:80
curl: (1) Received HTTP/0.9 when not allowed

┌──(kali㉿kali)-[~]
└─$ firefox http://10.0.0.71:80
^CExiting due to channel error.
Exiting due to channel error.
Exiting due to channel error.
Exiting due to channel error.
Exiting due to channel error.
Exiting due to channel error.
Exiting due to channel error.

┌──(kali㉿kali)-[~]
└─$ nc 10.0.0.71 21
220 Welcome to fake FTP server
USER root
PASS letmein
^C

┌──(kali㉿kali)-[~]
└─$ ssh root@10.0.0.71
hello
!
^C

┌──(kali㉿kali)-[~]
└─$ telnet 10.0.0.71 23
Trying 10.0.0.71...
Connected to 10.0.0.71.
Escape character is '^]'.
Welcome to the Telnet server
help
```

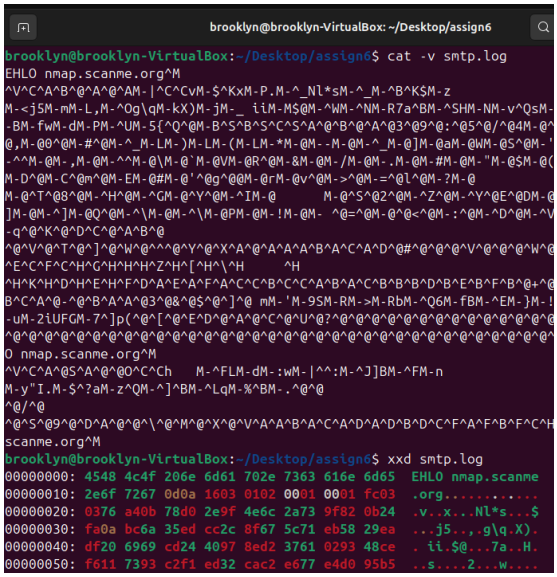
```
?               print help information
telnet> close
Connection closed.

┌──(kali㉿kali)-[~]
└─$ nc 10.0.0.71 25
220 fake.smtp.server ESMTP Postfix
wow
so cool
sick
^C

┌──(kali㉿kali)-[~]
└─$ nmap -p 1-65535 -T4 -A -v 10.0.0.71
Starting Nmap 7.95 ( https://nmap.org ) at 2025-04-23 20:28 EDT
NSE: Loaded 157 scripts for scanning.
NSE: Script Pre-scanning.
Initiating NSE at 20:28
Completed NSE at 20:28, 0.00s elapsed
Initiating NSE at 20:28
Completed NSE at 20:28, 0.00s elapsed
Initiating NSE at 20:28
Completed NSE at 20:28, 0.00s elapsed
Initiating ARP Ping Scan at 20:28
Scanning 10.0.0.71 [1 port]
Completed ARP Ping Scan at 20:28, 0.07s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 20:28
Completed Parallel DNS resolution of 1 host. at 20:29, 11.05s elapsed
Initiating SYN Stealth Scan at 20:29
Scanning 10.0.0.71 [65535 ports]
Discovered open port 22/tcp on 10.0.0.71
Discovered open port 23/tcp on 10.0.0.71
Discovered open port 25/tcp on 10.0.0.71
Discovered open port 21/tcp on 10.0.0.71
Discovered open port 80/tcp on 10.0.0.71
```

The images below show the feedback received from running `commands 16 and 17` respectively. The left image shows the connection to the locally hosted web page from the Kali side, with a little more details there would be no red flags showing Kali that this service was being logged. However, the image on the right shows the feedback from the connection in the Ubuntu terminal.



Next, I checked the logs produced by the various connections. Firstly, since the listener commands were run using `sudo` that make the permissions root, so I had to run `command 23` to change the permissions of the file before I could open it. To ensure that the command ran properly, I ran `command 24` to check if the permissions had updated. Once that was done I opened the smtp (port 23) log using `command 25`. The results of this, shown below on the left, revealed one plain text command I had typed in when connected, followed by many characters that could not be converted to ascii. These characters are most likely partial inputs that are the result of the nmap scan. In order to get a better understanding of these seemingly random values in the log, I ran `commands 26 and 27` to view the results of the log in a different manner. The results of these are shown below on the right.





The images below show the WireShark results on ports 21, 80, and 22, respectively going left to right and top to bottom. From this we can see all traffic sent between the two machines. Most of which is available to read in plain text and is unencrypted. Blue scripts are from the Ubuntu, honeypot server, with red from the Kali, attacker server.

```
Wireshark ·

220 Welcome to fake FTP server

USER root
PASS letmein
```

```
GET / HTTP/1.1
Host: 10.0.0.71
User-Agent: curl/8.11.0
Accept: */*


-c HTTP/1.1 200 OK\n\n Wed Apr 23 05:02:00 PM PDT 2025
```

```
SSH-2.0-OpenSSH_7.4

SSH-2.0-Nmap-SSH2-Hostkey
......k....%..C..?]
j(....diffie-hellman-group1-sha1,diffie-hellman-group14-sha1,diffie-hellman-group14-sha256
,diffie-hellman-group16-sha512,diffie-hellman-group-exchange-sha1,diffie-hellman-group-exc
hange-sha256....ecdsa-sha2-nistp384...Waes128-cbc,3des-cbc,blowfish-cbc,aes192-cbc,aes256-
cbc,aes128-ctr,aes192-ctr,aes256-ctr...Waes128-cbc,3des-cbc,blowfish-cbc,aes192-cbc,aes256
-cbc,aes128-ctr,aes192-ctr,aes256-ctr...!hmac-md5,hmac-sha1,hmac-ripemd160...!hmac-md5,hma
c-sha1,hmac-ripemd160....none....none...............o..z<.
```

For further analysis, I attempted to initialize my honey pot server to make it more realistic and dynamic. In order to do this, I created two python scripts, `http_server.py` and `echo_server.py`. These scripts would simulate an http web server on port 8080 and an “echo” server set up on port 9090. Both of these scripts involved a segment to log incoming and outgoing traffic, making sure to note the time, the sender and receiver of the traffic, as well as the message or traffic itself.

Open http_server.py
~/

```python
from http.server import BaseHTTPRequestHandler, HTTPServer
from datetime import datetime

class MyHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
        self.send_header('Content-type', 'text/plain')
        self.end_headers()
        response = f"HTTP Service\nCurrent time: {datetime.now(
        self.wfile.write(response.encode())

    def log_message(self, format, *args):
        with open("http.log","a") as log_file:
            log_file.write("%s - - [%s] %s\n" % (
                self.client_address[0],
                self.log_date_time_string(),
                format % args
            ))

if __name__ == '__main__':
    print("Serving HTTP on port 8080...")
    server_address = ('',8080)
    httpd = HTTPServer(server_address,MyHandler)
    httpd.serve_forever()
```

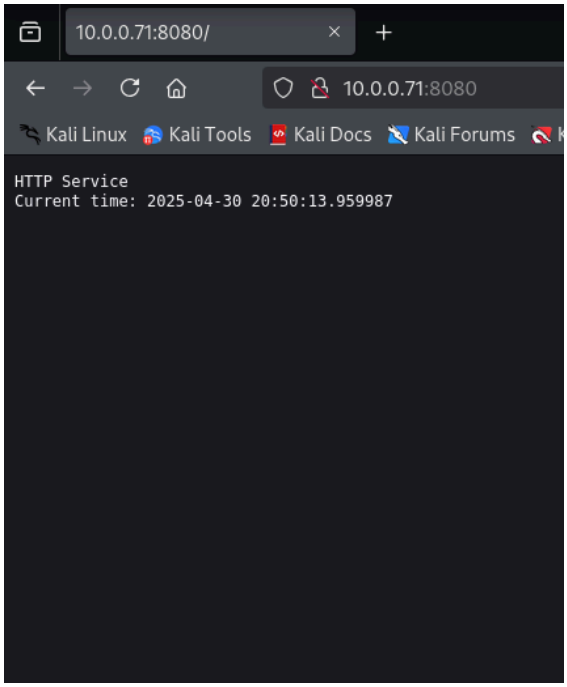
```python
import socket
from datetime import datetime

LOG_FILE = "echo.log"

def log(message):
  timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
  entry = f"[{timestamp}] {message}\n"
  print(entry,end="")
  with open(LOG_FILE,"a") as f:
    f.write(entry)

HOST = '0.0.0.0'
PORT = 9090

with socket.socket(socket.AF_INET,socket.SOCK_STREAM) as server_socket:
  server_socket.bind(HOST,PORT))
  server_socket.listen(1)
  log(f"Echo service running on port {PORT}...")

    while TRUE:
      client_socket, client_address = server_socket.accept()
      with client_socket:
        log(f"Connected by {client_address}")

        greeting = "Welcome to the Echo Server!\n"
        client_socket.sendall(greeting.encode())
        log(f"Sent greeting to {client_address}")

        while True:
          data = client_socket.recv(1024)
          if not data:
            break
          decoded = data.decode().strip()
          log(f"Received from {client_address}: {decoded}")
          client_socket.sendall(data)

        log("Connection closed for {client_address}")
```

Now that the scripts had been written, I used `commands 28 and 29` to run the script and thus start up the services. This involved me debugging the scripts in order to get them to connect, as shown in the left image below, I got a few errors before I was able to get the scripts running well. But eventually received the feedback that the server was up and running, “Serving HTTP on port 8080…” Still I ran `command 30` to ensure that the server was able to be connected to, as opposed to just the output message being sent. Once I received the correct feedback I knew it was ready to attack from the Kali side.

```
brooklyn@brooklyn-VirtualBox:~$ python3 http_server.py
  File "/home/brooklyn/http_server.py", line 22
    server_address = ('',8080)
TabError: inconsistent use of tabs and spaces in indentation
brooklyn@brooklyn-VirtualBox:~$ python3 http_server.py
  File "/home/brooklyn/http_server.py", line 23
    httpd = HTTPServer(server_address,MyHandler)
TabError: inconsistent use of tabs and spaces in indentation
brooklyn@brooklyn-VirtualBox:~$ python3 http_server.py
Serving HTTP on port 8080...
```

```
brooklyn@brooklyn-VirtualBox:~$ curl http://localhost:8080
curl: (7) Failed to connect to localhost port 8080 after 0 ms: Cou
to server
brooklyn@brooklyn-VirtualBox:~$ curl http://localhost:8080
curl: (7) Failed to connect to localhost port 8080 after 0 ms: Cou
to server
brooklyn@brooklyn-VirtualBox:~$ curl http://localhost:8080
HTTP Service
Current time: 2025-04-30 20:16:05.026728
brooklyn@brooklyn-VirtualBox:~$
```

For this part I attempted to connect in more natural ways, so I opened the Firefox browser in Kali and entered the url at `command 31`. I received the correct feedback and moved on to check this http server from the terminal, and so I ran `command 32`. This displayed the web traffic information in the form of html script. Next, I moved on to connect to the echo server running on port 9090, so I ran `command 33`. After working out a few bugs, I was able to connect and all messages I sent into the chat were echoed back at me.




```
└─$ telnet 10.0.0.71 8080
Trying 10.0.0.71 ...
Connected to 10.0.0.71.
Escape character is '^]'.
GET / HTTP1.1
<!DOCTYPE HTML>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Error response</title>
    </head>
    <body>
        <h1>Error response</h1>
        <p>Error code: 400</p>
        <p>Message: Bad request version ('HTTP1.1').</p>
        <p>Error code explanation: 400 - Bad request syntax or
hod.</p>
    </body>
</html>
Connection closed by foreign host.

┌──(kali㉿kali)-[~]
└─$

┌──(kali㉿kali)-[~]
└─$ telnet 10.0.0.71 9090
Trying 10.0.0.71 ...
telnet: Unable to connect to remote host: Connection refused

┌──(kali㉿kali)-[~]
└─$ nc 10.0.0.71 9090
(UNKNOWN) [10.0.0.71] 9090 (?) : Connection refused
```

The below images show the WireShark packet captured on port 8080, on the left, and port 9090, on the right. From this we can see that none of the traffic was encrypted and it is all readable in plain text.

```
GET / HTTP/1.1
Host: 10.0.0.71:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128
.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/png,image/svg+xml,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Cookie: _ga=GA1.1.1212600711.1743033689; _y=d4e768d7-9813-41D7-D552-6AB11ED5A3B9
; _shopify_y=d4e768d7-9813-41D7-D552-6AB11ED5A3B9; __qca=P0-742435420-1743033689
065
Upgrade-Insecure-Requests: 1
Priority: u=0, i


HTTP/1.0 200 OK
Server: BaseHTTP/0.6 Python/3.12.3
Date: Thu, 01 May 2025 03:50:13 GMT
Content-type: text/plain

HTTP Service
Current time: 2025-04-30 20:50:13.959987
```

```
Wireshark · Follow TC

Welcome to the Echo Server!

sick

sick

echoooo

echoooo

wowza

wowza

.....
```

The images below are the final results of the logs files, with the http(1).log on the left and echo.log on the right. These logs tell us much more about the interaction between the attacker and the honey pot than the previous simple command run services. In these we can see the ip address of the attacker and the honeypot, which machine is sending which packets, as well as the exact time of the transfers.

```
New Text Document.txt    http(1).log
File   Edit   View

127.0.0.1 - - [30/Apr/2025 20:16:05] "GET / HTTP/1.1" 200 -
10.0.0.228 - - [30/Apr/2025 20:50:13] "GET / HTTP/1.1" 200 -
10.0.0.228 - - [30/Apr/2025 20:50:14] "GET /favicon.ico HTTP/1.1" 200 -
10.0.0.228 - - [30/Apr/2025 20:50:57] code 400, message Bad request version
('HTTP1.1')
10.0.0.228 - - [30/Apr/2025 20:50:57] "GET / HTTP1.1" 400 -
```

```
http(1).log    echo.log
File   Edit   View

[2025-04-30 20:57:49] Echo service running on port 9090...
[2025-04-30 20:58:12] Echo service running on port 9090...
[2025-04-30 21:04:07] Connected by ('10.0.0.228', 47510)
[2025-04-30 21:04:07] Sent greeting to ('10.0.0.228', 47510)
[2025-04-30 21:04:09] Received from ('10.0.0.228', 47510): hi
[2025-04-30 21:04:10] Received from ('10.0.0.228', 47510): wow
[2025-04-30 21:04:15] Received from ('10.0.0.228', 47510): this is legit
[2025-04-30 21:04:17] Connection closed for {client_address}
[2025-04-30 21:04:35] Connected by ('10.0.0.228', 46078)
[2025-04-30 21:04:35] Sent greeting to ('10.0.0.228', 46078)
[2025-04-30 21:04:35] Received from ('10.0.0.228', 46078): hello
[2025-04-30 21:04:45] Connection closed for {client_address}
[2025-04-30 21:04:57] Connected by ('10.0.0.228', 58874)
[2025-04-30 21:04:57] Sent greeting to ('10.0.0.228', 58874)
[2025-04-30 21:04:59] Received from ('10.0.0.228', 58874): sick
[2025-04-30 21:05:01] Received from ('10.0.0.228', 58874): echoooo
[2025-04-30 21:05:04] Received from ('10.0.0.228', 58874): wowza
```

Conclusion

From this assignment we have seen the advantages and limitations of setting up a honeypot server. While there are simple ways to start up a server on a specific port, it becomes much more challenging and time consuming to create hyper realistic imitations of these services. However, it is immediately apparent if the advantages of having a honeypot server set up. Through this server, with careful logging, you can determine other users, possibly malicious, you can determine their ip address as well as keep records of messages and data being sent.

In the case of attackers, this can be incredibly helpful to determine the what, where, when, and how of the attackers methods. I do not include the who as it is possible the attacker is using a proxy and in that case this information can be somewhat telling but is not a surefire way.

However, there is value in being able to determine which port the attacker is seeking as well as their method of attack, since all messages are recorded.

This can help to improve security as it can help to determine the mere presence of an attacker and what they are targeting. This helps ordinary users and victims to not only prepare for an attack but also isolate any corrupted software and control or remove it. In cases of worms it can also help to determine which users are unknowingly spreading this malware and so there are multiple ways to check and determine which users are corrupted. If a honeypot were to be combined with a firewall, we could almost immediately isolate or kick malicious users from our networks. This helping to mitigate any damage they might be causing.