Network Research Project
Shaun Sng (S17)
Mar 2024

Table of Contents

1. Introduction

Summary - Part one of this project entails preparing a script with multiple stages:

a. To check for and install required applications such as *sshpass* and *nipe*.
b. To make an anonymous connection via nipe.
c. To ask the user for a domain or IP address for scanning. Namely, the target or victim.
d. To access a remote server and scan the provided domain/IP from that location, and to log the results.
e. Lastly, to retrieve the log to the local machine.

Aim - This project simulates the initial phase of vulnerability assessment (blue team) or penetration testing/cyber attack (red team). Anonymising the user's location twice over by Nipe/tor and by scanning the victim/target from a remote server underscores the importance of stealth and how actual attacker's would use similar approaches when carrying out such reconnaissance activity.

This project required students to have fundamental knowledge of networking, and also further develops/tests our scripting competencies.

2. Methodologies

2.1 Application check and installation

apt-mark showinstall retrieves a list of applications that are installed and/or marked for installation. A for loop checks three required applications (geoip-bin, tor and sshpass) against this list, and to install them if not present. *Grep* with flag *-x* is stricter matching by whole line, in order to extract only the application name (e.g. “tor”). This facilitates the exact string matching required in the IF condition.

The directory from where the script is run is saved as variable *start* to facilitate commands and navigation later on in the script. To facilitate the automatic installation of applications the “kali” password is echoed to the sudo command to avoid prompting for password.

```
11  # Retrieve record of installed apps and output to text file for checking later in script.
12  apt-mark showinstall > installed.txt
13
14  # Saving directory where script is run as variable, to facilitate navigation later on.
15  start=$(pwd)
16
17  # Run loop to check if first three apps are installed, and install them if not.
18  apps=('geoip-bin' 'tor' 'sshpass')
19  for app in "${apps[@]}"
20
21  do
22     # Use string match to check if app names are listed in apt-mark.
23     result=$(cat installed.txt | grep -x $app)
24     if [ $result == $app ]
25     then
26      echo "[+] $app is already installed"
27     else
28      echo "[-] $app was not found. Installing now."
29      echo "kali" | sudo -S apt-get install $app
30      sleep 1
31     fi
32
33  done
```

```
┌──(kali㉿kali)-[~/network/project]
└─$ bash NR_project.sh
[+] geoip-bin is already installed
[+] tor is already installed
[+] sshpass is already installed
```

Nipe is processed separately from other applications as checking for the application and installation is different. As Nipe does not appear on apt-mark (or apt list) even when installed, *locate* is used to search for the **nipe.pl file**, to determine if it is available. Prior to this, *updatedb* is run to refresh the database.

The path of the nipe.pl file is saved to a variable nipefile, which is first used in the IF condition. If there is no nipe.pl, this variable will be empty, initiating the “then” path and executing the commands to install nipe. If nipe.pl is found, *dirname* strips the filename from its path, providing us the file’s parent folder. We cd into this directory to prepare for the next stage.

```
35  # As installing nipe requires several steps, we process this separately from other apps.
36  # We search for nipe.pl file to check if it is installed, as it does not appear on apt-list or apt-mark records eve
37
38  echo "kali" | sudo -S updatedb
39  nipefile=$(locate nipe.pl)
40
41  if [ -z "$nipefile" ]
42
43  then
44      # Installation steps if nipe not found.
45      echo "[-] Nipe was not found. Installing now."
46      git clone https://github.com/htrgouvea/nipe && cd $start/nipe
47      sudo apt-get install cpanminus
48      cpanm --installdeps .
49      sudo cpan install Switch JSON LWP::UserAgent Config::Simple
50      sudo perl nipe.pl install
51      sleep 2
52
53  else
54      echo "[+] Nipe is already installed."
55      # Earlier saved location of search is also useful to help us enter directory where nipe is already installed.
56      cd $(dirname $nipefile)
57
58  fi
```

```
[sudo] password for kali: [+] Nipe is already installed.
```

2.2 Establish anonymous connection

Throughout this process, the user is informed of the progress and status e.g. “Attempting to connect…” Additional *echo* is included to provide spacing in the screen output for readability, and *sleep* to pause execution, and give time for the user to process the information.

sudo perl nipe.pl start and *sudo perl nipe.pl status* are run to make an anonymous connection and check on its status. The result is saved to a nipestatus.txt file and a simple *grep* match for “true” to determine if the attempt was successful.

If successful, the spoofed IP address and Country are retrieved from the status file using *grep*, *awk* and *geoiplookup* before being output to screen.

```
60  #~ 2. Connect through Nipe and once connected(anonymous), display the spoofed IP and Country.
61  #~ a. If we are unable to connect, let the user know then exit the script right away.
62
63  echo
64  echo "[➔] All apps ready. Attempting to connect anonymously..."
65  echo
66  sleep 2
67
68  # We attempt to connect to nipe, save status and check if connection successful using keyword "true".
69
70  sudo perl nipe.pl start
71  sudo perl nipe.pl status > nipestatus.txt
72  nipetrue=$( cat nipestatus.txt | grep true)
73
74  if [ ! -z "nipetrue" ]
75  then
76      echo "[☺] Anonymous connection successful!"
77
78      # Retrieving IP and country info to show user.
79      IPadd=$(cat nipestatus.txt | grep Ip | awk '{print$3}')
80      Country=$(geoiplookup $IPadd | awk -F: '{print$2}')
81      echo "[➔] Your spoofed IP address is: $IPadd"
82      echo "[➔] Your spoofed country is: $Country"
83
84  else
85      echo "[☹] Failed to make anonymous connection. This script will now exit. Please try again later."
86  fi
```

```
[➔] All apps ready. Attempting to connect anonymously ...

[☺] Anonymous connection successful!
[➔] Your spoofed IP address is: 51.195.91.124
[➔] Your spoofed country is:  FR, France
```

2.3 Prompting user for domain/IP

A straightforward *read* is used to ask the user for the target domain/IP address and save it as a variable. The Ubuntu remote server IP address is saved as a variable in preparation for the next stage.

```
88  #~ 3. Get the user's input for the domain/url to scan
89
90  # Straightforward query and saving domain/IP as variable.
91
92  echo
93  echo "[?] Specify a domain or IP address to scan."
94  read IPcheck
95
96  #~ 4. Connect to the remote server via ssh.(`sshpass`)
97
98  serverIP='192.168.133.128'
99  echo "[→] Connecting to server:"
100
```

```
[?] Specify a domain or IP address to scan.
cnn.com
[→] Connecting to server:
```

2.4 Access remote server and scan target

sshpass is used to “pass” the known password to the *ssh* command (with the known user and server), eliminating the password prompt and facilitating automated access. An EOF marker is used to organise the series of commands to be executed. *Uptime* retrieves how long the server has been running, and *geoiplookup* the country of the server’s IP.

Whois retrieves domain registration information which could include the name and contact information of the registrant. This is run on the target (variable $IPcheck). in the second batch of commands. *Date* retrieves date and time, to function as a timestamp. These results are saved to “scan.log” file and processes briefly echoed on screen to update user of actions.

```
104  # Access remote server with known credentials. Using sshpass so no password prompt needed.
105  # EOF delimiter to organise multiple commands to be executed on server.
106
107  sshpass -p "tc" ssh tc@$serverIP <<EOF
108      echo
109      echo "Successfully logged on to remote server."
110      echo
111      sleep 2
112      echo "Server uptime: $(uptime)"
113      echo "IP address: $serverIP"
114      echo "Country: $(geoiplookup $serverIP | awk -F: '{print$2}')"
115      echo
116      echo "Whois check running on specified domain/address..."
117      TZ='Singapore' date >> scan.log
118      echo $IPcheck >> scan.log
119      whois $IPcheck >> scan.log
120      echo >> scan.log
121      echo "$(TZ='Singapore' date) - whois data for: $IPcheck collected in current directory in scan.log"
122
123  EOF
```

```
Successfully logged on to remote server.

Server uptime:  21:30:21 up  5:32,  1 user,  load average: 0.19, 0.17, 0.18
IP address: 192.168.133.128
Country:  IP Address not found

Whois check running on specified domain/address ...
Thu Mar 21 01:30:21 AM Singapore 2024 - whois data for: cnn.com collected in current directory in scan.log
```

2.5 Retrieve log file

sshpass is used again along with *scp* (secure copy - which copies a file from a remote server via ssh) is used to retrieve the scan.log file to the local machine.

rm is used to delete two text files created in the script. While removing these traces may not be necessary assuming the Kali machine is a known or internal machine, potentially the script could also be run in an external user's machine.

```
125 #~ 5. We will want to have a log file to record the domain/URL scanned.
126 # Remember to include the day, date and time. And of course the domain/URL that was scanned.
127
128 # Running sshpass + securecopy to retrieve the log from server to local machine.
129
130 sshpass -p "tc" scp tc@$serverIP:/home/tc/scan.log $start
131
132 # Removing the temporary files we created to reduce trace of activity.
133 rm $start/nipe/nipestatus.txt
134 rm $start/installed.txt
135
136 echo "This script has completed. Have a nice day."
137
```

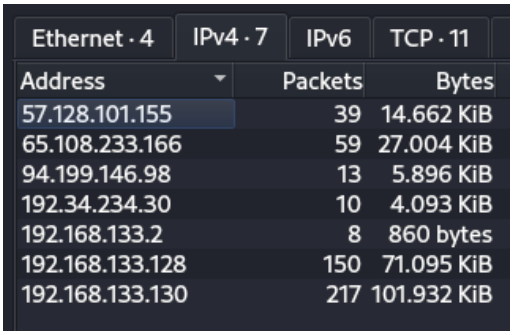
2.6 Tcpdump and Wireshark

tcpdump -i eth0 -w NRscript.pcap was run to capture packets from Host Kali eth0 interface. The -w flag saves results in a pcap file for the analysis in Wireshark.

250 packets in 26 seconds were captured.

Resolved Endpoints - Four public IPs and three (familiar) internal IPs are captured in the PCAP.

- 192.168.133.2 - Default Gateway
- 192.168.133.128 - Ubuntu server
- 192.168.133.130 - Host Kali



Ethernet · 4 | IPv4 · 7 | IPv6 | TCP · 11

Address	Packets	Bytes
57.128.101.155	39	14.662 KiB
65.108.233.166	59	27.004 KiB
94.199.146.98	13	5.896 KiB
192.34.234.30	10	4.093 KiB
192.168.133.2	8	860 bytes
192.168.133.128	150	71.095 KiB
192.168.133.130	217	101.932 KiB

Where are the Public IPs from - Using bulk IP lookup, the four public IPs originate from Finland, UK, US and France. Noting earlier results from the script, the French IP is likely the last node where our Nipe is tunnelling out from. One obvious point to still mention is that the Spoofed IP returned in nipe status - 51.195.91.124 - is not the actual IP shown in the connected French machine - 57.128.101.155.

IP	Domain	Country	Region	City	ISP
57.128.101.155		France			OVH SAS
65.108.233.166	static.166.233.108.65.clients.your-server.de	Finland	Uusimaa	Helsinki	Hetzner Online GmbH
94.199.146.98	rdap.comlaude.com	United Kingdom			Redcentric Solutions Ltd
192.34.234.30	whois-2.verisign-grs.com	United States			VERISIGNGRS

4 out of 4 hosts successfully processed. Have a nice day!

Anonymous connection Initial packets in the PCAP show host machine 192.168.133.130 communicating with two IPs - 65.108.233.166 in Finland and 57.128.101.155 in France, mostly by Transport Layer Security (TLS) v1.2 - more familiar as the protocol that facilitates encryption when we browse using HTTPS.

This traffic shows the process of Nipe attempting to make an anonymous connection via the Tor network.

	Time	Source	Destination	Protocol	Length	Info
1	0.000000	57.128.101.155	192.168.133.130	TLSv1.2	590	Application Data
2	0.000049	192.168.133.130	57.128.101.155	TCP	54	58682 → 443 [ACK]
3	4.975859	65.108.233.166	192.168.133.130	TLSv1.2	590	Application Data
4	4.975892	192.168.133.130	65.108.233.166	TCP	54	49342 → 9001 [ACK
5	5.731993	57.128.101.155	192.168.133.130	TLSv1.2	590	Application Data
6	5.732027	192.168.133.130	57.128.101.155	TCP	54	58682 → 443 [ACK]
7	7.895458	192.168.133.130	65.108.233.166	TLSv1.2	1104	Application Data
8	7.895687	65.108.233.166	192.168.133.130	TCP	60	9001 → 49342 [ACK
9	8.234612	57.128.101.155	192.168.133.130	TLSv1.2	590	Application Data
10	8.234642	192.168.133.130	57.128.101.155	TCP	54	58682 → 443 [ACK]
11	8.326728	192.168.133.130	65.108.233.166	TLSv1.2	590	Application Data
12	8.326826	192.168.133.130	57.128.101.155	TLSv1.2	1104	Application Data

Connecting to remote server Subsequent packets show host Kali 192.168.133.130 making the SSH connection to Ubuntu server at 192.168.133.128. SSH is shown explicitly in the Protocol Column with port 22 open on the server. Frames 103 and 104 interestingly mention the *Diffie-Hellman Key Exchange* - which entails the exchange of cryptographic keys and points to the background details /processes of how SSH works.

92	17.185669	192.168.133.130	57.128.101.155	TCP	54	58682 → 443 [ACK] Seq=5875 Ack=6433 Win=65535 Len=0
93	19.496717	192.168.133.130	192.168.133.128	TCP	74	57508 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_P
94	19.496987	192.168.133.128	192.168.133.130	TCP	74	22 → 57508 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=
95	19.497029	192.168.133.130	192.168.133.128	TCP	66	57508 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=181
96	19.497290	192.168.133.130	192.168.133.128	SSHv2	98	Client: Protocol (SSH-2.0-OpenSSH_9.6p1 Debian-3)
97	19.497469	192.168.133.128	192.168.133.130	TCP	66	22 → 57508 [ACK] Seq=1 Ack=33 Win=65152 Len=0 TSval=38
98	19.503015	192.168.133.128	192.168.133.130	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu
99	19.503039	192.168.133.130	192.168.133.128	TCP	66	57508 → 22 [ACK] Seq=33 Ack=42 Win=64256 Len=0 TSval=1
100	19.503282	192.168.133.130	192.168.133.128	SSHv2	1602	Client: Key Exchange Init
101	19.503948	192.168.133.128	192.168.133.130	SSHv2	1178	Server: Key Exchange Init
102	19.551667	192.168.133.130	192.168.133.128	TCP	66	57508 → 22 [ACK] Seq=1569 Ack=1154 Win=64128 Len=0 TSv
103	19.567384	192.168.133.130	192.168.133.128	SSHv2	1274	Client: Diffie-Hellman Key Exchange Init
104	19.580040	192.168.133.128	192.168.133.130	SSHv2	1630	Server: Diffie-Hellman Key Exchange Reply, New Keys
105	19.580074	192.168.133.130	192.168.133.128	TCP	66	57508 → 22 [ACK] Seq=2777 Ack=2718 Win=64000 Len=0 TSv
106	19.603213	192.168.133.130	192.168.133.128	SSHv2	82	Client: New Keys
107	19.644170	192.168.133.128	192.168.133.130	TCP	66	22 → 57508 [ACK] Seq=2718 Ack=2793 Win=64128 Len=0 TSv
108	19.644187	192.168.133.130	192.168.133.128	SSHv2	110	Client:
109	19.644344	192.168.133.128	192.168.133.130	TCP	66	22 → 57508 [ACK] Seq=2718 Ack=2837 Win=64128 Len=0 TSv

Whois to multiple servers These frames show the process of the remote server executing the Whois command. Frames 133-134 are DNS queries to retrieve the IP address of the verisign.whois servers, before the actual query and response for cnn.com in frames 142 and 144.

132	21.956597	192.168.133.130	192.168.133.128	TCP	66	57508 → 22 [ACK] Seq=3797 Ack=5022 Win=64128 Len=0 TSval=1819986441 TSecr=3649165774
133	21.958765	192.168.133.128	192.168.133.2	DNS	93	Standard query 0x60c1 A whois.verisign-grs.com OPT
134	21.958865	192.168.133.128	192.168.133.2	DNS	93	Standard query 0x3d88 AAAA whois.verisign-grs.com OPT
135	21.966849	00:50:56:ed:fd:3d	Broadcast	ARP	60	Who has 192.168.133.128? Tell 192.168.133.2
136	21.967015	00:0c:29:e6:ff:34	00:50:56:ed:fd:3d	ARP	60	192.168.133.128 is at 00:0c:29:e6:ff:34
137	21.967016	192.168.133.2	192.168.133.128	DNS	125	Standard query response 0x60c1 A whois.verisign-grs.com A 192.34.234.30 A 192.30.45.30 OPT
138	21.967031	192.168.133.2	192.168.133.128	DNS	149	Standard query response 0x3d88 AAAA whois.verisign-grs.com AAAA 2620:74:21::30 AAAA 2620:74:20:
139	21.967398	192.168.133.128	192.34.234.30	TCP	74	49046 → 43 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=129711118 TSecr=0 WS=128
140	22.133720	192.34.234.30	192.168.133.128	TCP	60	43 → 49046 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
141	22.133901	192.168.133.128	192.34.234.30	TCP	60	49046 → 43 [ACK] Seq=1 Ack=1 Win=64240 Len=0
142	22.134055	192.168.133.128	192.34.234.30	WHOIS	70	Query: domain cnn.com
143	22.134067	192.34.234.30	192.168.133.128	TCP	60	43 → 49046 [ACK] Seq=1 Ack=17 Win=64240 Len=0
144	22.380744	192.34.234.30	192.168.133.128	WHOIS	3627	Answer: domain cnn.com

Of note, similar queries was also sent to another whois server at 94.199.146.98 at domain comlaude.com.

149	22.301668	192.168.133.128	192.168.133.2	DNS	89	Standard query 0x4beb A whois.comlaude.com OPT
150	22.301763	192.168.133.128	192.168.133.2	DNS	89	Standard query 0x5493 AAAA whois.comlaude.com OPT
151	22.313800	192.168.133.2	192.168.133.128	DNS	105	Standard query response 0x4beb A whois.comlaude.com A 94.
152	22.313801	192.168.133.2	192.168.133.128	DNS	117	Standard query response 0x5493 AAAA whois.comlaude.com AA
153	22.314211	192.168.133.128	94.199.146.98	TCP	74	59262 → 43 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM
154	22.627863	94.199.146.98	192.168.133.128	TCP	60	43 → 59262 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=146
155	22.628040	192.168.133.128	94.199.146.98	TCP	60	59262 → 43 [ACK] Seq=1 Ack=1 Win=64240 Len=0
156	22.628206	192.168.133.128	94.199.146.98	WHOIS	63	Query: cnn.com
157	22.628289	94.199.146.98	192.168.133.128	TCP	60	43 → 59262 [ACK] Seq=1 Ack=10 Win=64240 Len=0
158	23.137742	94.199.146.98	192.168.133.128	TCP	5301	43 → 59262 [PSH, ACK] Seq=1 Ack=10 Win=64240 Len=5247 [TC
159	23.137833	192.168.133.128	94.199.146.98	TCP	60	59262 → 43 [ACK] Seq=10 Ack=1461 Win=62780 Len=0
160	23.137833	192.168.133.128	94.199.146.98	TCP	60	59262 → 43 [ACK] Seq=10 Ack=2921 Win=62780 Len=0
161	23.137902	192.168.133.128	94.199.146.98	TCP	60	59262 → 43 [ACK] Seq=10 Ack=4381 Win=61320 Len=0
162	23.137903	192.168.133.128	94.199.146.98	TCP	60	59262 → 43 [ACK] Seq=10 Ack=5248 Win=61320 Len=0
163	23.141448	94.199.146.98	192.168.133.128	WHOIS	60	Answer: cnn.com

Retrieve log file The last batch of packets revert to SSH communications between the Host Kali and Ubuntu server, which should include transferring the scan.log file to Host Kali. However the Export-Objects dialog in Wireshark does not list any files for easy extraction - which is one of the main benefits of using secure protocol.

223	23.481817	192.168.133.128	192.168.133.130	SSHv2	1514	Server:
224	23.481948	192.168.133.128	192.168.133.130	SSHv2	1514	Server:
225	23.481958	192.168.133.128	192.168.133.130	SSHv2	1514	Server:
226	23.482002	192.168.133.128	192.168.133.130	SSHv2	4410	Server:
227	23.482079	192.168.133.130	192.168.133.128	TCP	66	58674 → 2
228	23.482252	192.168.133.128	192.168.133.130	SSHv2	4410	Server:
229	23.482285	192.168.133.128	192.168.133.130	SSHv2	2962	Server:
230	23.482295	192.168.133.128	192.168.133.130	SSHv2	1514	Server:
231	23.482304	192.168.133.128	192.168.133.130	SSHv2	1514	Server:
232	23.482376	192.168.133.128	192.168.133.130	SSHv2	4410	Server:
233	23.482445	192.168.133.130	192.168.133.128	TCP	66	58674 → 2
234	23.482570	192.168.133.128	192.168.133.130	SSHv2	250	Server:
235	23.482736	192.168.133.130	192.168.133.128	SSHv2	134	Client:
236	23.482981	192.168.133.128	192.168.133.130	SSHv2	134	Server:
237	23.483139	192.168.133.130	192.168.133.128	SSHv2	118	Client:
238	23.483404	192.168.133.128	192.168.133.130	SSHv2	134	Server:
239	23.483552	192.168.133.130	192.168.133.128	SSHv2	102	Client:
240	23.483993	192.168.133.128	192.168.133.130	SSHv2	190	Server:
241	23.484058	192.168.133.130	192.168.133.128	SSHv2	102	Client:
242	23.484097	192.168.133.130	192.168.133.128	SSHv2	126	Client:
243	23.484130	192.168.133.130	192.168.133.128	TCP	66	58674 → 2

3. Discussion

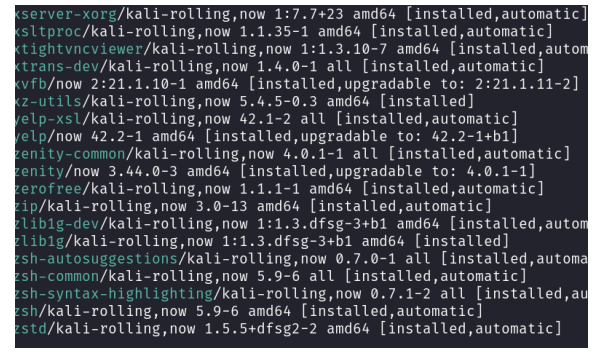
3.1 Script design and limitations

This section elaborates on choices made in scripting and discusses some of the implications and limitations as a result.

Checking for installed applications - Initially *apt list -installed* was explored to retrieve the list of installed applications. However, as the output is comprehensive, more text manipulation is needed in order to extract only the app name for an exact string matching. Grep with -w flag (match whole word) using this retrieved two results. Apart from tor itself, there is also a “tor-geoipdb.”

```
┌──(kali㉿kali)-[~/network/project]
└─$ apt list --installed | grep -w tor

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

tor-geoipdb/kali-rolling,now 0.4.8.10-1 all [installed,automatic]
tor/kali-rolling,now 0.4.8.10-1 amd64 [installed]

┌──(kali㉿kali)-[~/network/project]
└─$ apt list --installed | grep -x tor

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.
```

As *apt-mark showinstall* retrieves only the app name (see comparison below), less text manipulation is needed to facilitate the string matching. However, one limitation is that applications marked for installation would be this list but potentially not yet installed. This was assessed not to be a critical issue for this project scope and operating environment. However if the task involved checking for applications in an enterprise environment where software updates are rolled out centrally and on a particular schedule, this would be more of a concern. In that environment it may be preferred to use the more up to date apt list installed, and spend the effort on text manipulation.

Apt list -installed output	Apt-mark showinstall output
	

Sudo & security Given the nature of the project, it was assessed that the script by and large should avoid having to require user input, except for the target domain/IP explicitly stated in the scope. However, sudo commands will be needed as installation of applications is entailed.

To avoid prompting the user for password, the known password “kali” is echoed and piped into initial sudo commands. However, this is not an ideal practice from a security standpoint. A brief online search suggests more secure alternatives include saving the password in a read-only file and passing it to the sudo command, and/or manipulating file user permissions. For the purpose of this project, no further exploration was done into these more complex methods.

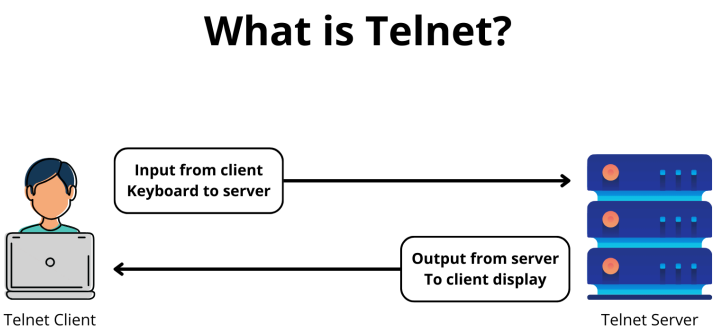
Stealth/Traces This script created two temporary text files, which are removed at the end. While removing these traces may not be necessary if the linux machine it is run from is an “internal” machine. However, potentially the script could also be run in an external user’s machine where it would be preferred to have as little trace of activity as possible. From a stealth standpoint it may be ideal if the script does not create any of such “temporary files,” although from a processing/speed standpoint at time it may be still be preferred to do so rather than run particular command multiple times.

```
131
132  # Removing the temporary files we created to reduce trace of activity.
133  rm $start/nipe/nipestatus.txt
134  rm $start/installed.txt
135
```

Keeping the user informed Noting trainer's previous project feedback, script was drafted to keep the user informed of its actions. This includes simple additions like improving readability of the script by echoing empty lines, and using *sleep* to pause the output at times. One feature was adopted from trainer's demonstration was to include "bullet points" in script output e.g. [#]. This improves the alignment of the output while adding subtle meaning for some lines (e.g. using "?" when asking for user input.

3.2 Non-secure Protocol - Telnet

Telnet is a network protocol/service that provides an 8-bit, text-based, bidirectional command line interface for communication with a remote device or server. Simply, it allows a user to remotely access and control another computer.



Telnet uses connection based Transmission Control Protocol (TCP) as its underlying transport protocol (default port 23). Development began in 1969 with RFC 15 before eventual standardisation as Internet Engineering Task Force (IETF) Internet Standard STD 8 (RFC 854) in 1983.

How to use Telnet - A user can use a command line Telnet client with the following simple syntax to access a Telnet: *telnet hostname <port number>.* The user then executes commands on the server by using specific Telnet commands into the Telnet prompt. To end a session and log off, the user ends a Telnet command with Telnet.

Telnet by Wireshark - To observe actual Telnet traffic, a windows client was used to connect to *telehack.com* and speak with the AI psychologist "Eliza." The communication is sent in plain text and can be easily extracted by just following the TCP stream in Wireshark.

4 0.021619	192.168.133.2	192.168.133.131	DNS	88 Standard query response 0x6345 A telehack.com A 64.13.139.230
5 0.021920	192.168.133.131	64.13.139.230	TCP	66 49803 → 23 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
6 0.207213	64.13.139.230	192.168.133.131	TCP	60 23 → 49803 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
7 0.207302	192.168.133.131	64.13.139.230	TCP	54 49803 → 23 [ACK] Seq=1 Ack=1 Win=64240 Len=0
8 0.248871	64.13.139.230	192.168.133.131	TCP	60 23 → 49802 [RST, ACK] Seq=1 Ack=2 Win=64239 Len=0
9 0.413927	64.13.139.230	192.168.133.131	TELNET	60 Telnet Data ...
10 0.414109	192.168.133.131	64.13.139.230	TELNET	57 Telnet Data ...

```
Connected to TELEHACK port 135
.............V.(....$..'...
It is 5:46 am on Thursday, March 21, 2024 in Mountain View, Cali
There are 135 local users. There are 26648 hosts on the network.

May the command line live forever.

Command, one of the following:
  2048         ?            a2           advent       aquarium
  c8           cal          calc         callsign     cat
  cowsay       ddate        delta        diff         dir
  exit         factor       figlet       file         finger
  geoip        gif          help         ipaddr       joke
  login        minesweeper  more         morse        netstat
  pig          ping         pong         primes       privacy
  recover      rig          rockets      roll         run
  sleep        sudoku       tail         traceroute   typespeed
  usenet       uupath       uuplot       when         zc

More commands available after login. Type HELP for a detailed co
Type NEWUSER to create an account. Press control-C to interrupt
.........'.........ANSI....'...eleilziza

HELLO, I AM ELIZA.

HHeellllloo  whwoh o .....[K...[K..[K..[KII a mam  SShahuanun


HI. WHAT SEEMS TO BE YOUR PROBLEM?

II  eaeta t totoomoumchu ch chicchkeincken


BEING TERSE IS ONE WAY TO MAKE ME NERVOUS.
```

One unusual feature compared to other protocols is that by default Telnet sends data one character at a time - contributing to the large number of packets in the PCAP.

237	39.095640	64.13.139.230	192.168.133.131	TCP	60 23 → 49803 [A
238	39.132827	64.13.139.230	192.168.133.131	TELNET	60 Telnet Data .
239	39.199223	192.168.133.131	64.13.139.230	TCP	54 49803 → 23 [A
240	39.255252	192.168.133.131	64.13.139.230	TELNET	55 Telnet Data .
241	39.256012	64.13.139.230	192.168.133.131	TCP	60 23 → 49803 [A
242	39.331602	64.13.139.230	192.168.133.131	TELNET	60 Telnet Data .
243	39.371373	192.168.133.131	64.13.139.230	TCP	54 49803 → 23 [A
244	39.561634	64.13.139.230	192.168.133.131	TELNET	60 Telnet Data .
245	39.570457	192.168.133.131	64.13.139.230	TELNET	55 Telnet Data .
246	39.571739	64.13.139.230	192.168.133.131	TCP	60 23 → 49803 [A

```
Frame 242: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface \Device
Ethernet II, Src: VMware_ed:fd:3d (00:50:56:ed:fd:3d), Dst: VMware_b2:ce:e2 (00:0c:29:b2:
Internet Protocol Version 4, Src: 64.13.139.230, Dst: 192.168.133.131
Transmission Control Protocol, Src Port: 23, Dst Port: 49803, Seq: 1293, Ack: 86, Len: 1
Telnet
   Data: t
```

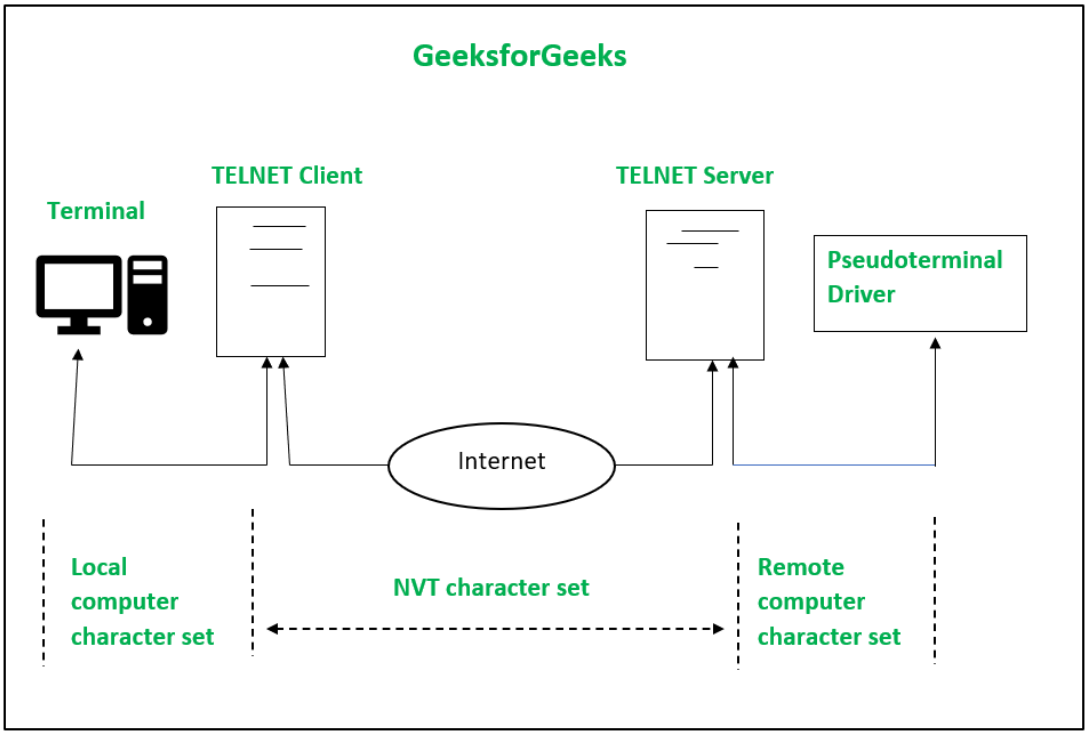
Insecure Protocol - As Telnet was developed before the mainstream adoption of the internet, by default it **does not use encryption**. As shown in the above wireshark communication, all data including sensitive information such as passwords would be

by default sent in plain text. This has a direct impact on **confidentiality** within the CIA triad - a malicious actor able to intercept Telnet traffic (perhaps via a MITM attack) would be able to view all information.

Knock on effects on **Availability** are also possible depending on the network environment and context in which Telnet was used. For instance, if Telnet was used to access a legacy Industrial Control System (ICS) controlling plant production, then availability of the service provided by the plant could be compromised assuming a malicious attacker obtains the necessary credentials to sabotage the plant remotely.

Generally, the insecure Telnet has been superseded by Secure Shell (SSH). Of note, the Telnet client is not enabled by default on Windows 10. However, there remain some uses for Telnet including as a troubleshooting and diagnostic tool (to check for open ports), with older legacy systems, and nostalgia driven entertainment such as MUDs (multi-user dungeon games).

Additional Features - One concept bundled with Telnet is that of Network Virtual Terminal (NVT). The diagram illustrates how communication between client and server involves a process of translating local computer characters into the NVT character set - creating a common language/environment for communication between terminals with different operating systems. This pioneering feature of **cross-interoperability** is worth noting, although it has been arguably superseded by the modern Internet.



Some characteristics of Telnet/NVT worth noting include:

- The Network Virtual Terminal (NVT) primarily employs two sets of characters: one for data and another for control.
- The NVT is an **8-bit character set** for data, with the 7 lowest-order bits identical to ASCII and the highest bit set to 0.
- For sending data and control characters the TELNET makes use of the same connection by just inserting control characters into the data stream.
- Each control character is preceded by the **Special Control character**, which is popularly known as Interpret as Control, for separating the data characters from the control characters (IAC).

The following is an excerpt of Telnet commands to show its own lexicon.

Telnet command section

Telnet commands are distinguished by various character set. The data stream portion of Telnet ensures the incorpo network users. Commands are always introduced via the Interpret as command (IAC) character.

Defined Telnet commands consist of:

WILL - 251 - Offer or accept to enable

WON'T - 252 - Accept of Offer to disable

DO - 253 - Request or Approve to enable operation

DON'T - 254 - Disapprove enable or Request to disable

SE - 240 - End of subnegotiation parameters

NOP - 241 - No operation

3.3 Secure Shell

Secure Shell (SSH) is a method for securely sending commands to a computer over an unsecured network, using cryptography to authenticate and encrypt connections between devices. SSH is notably used for remote login and command-line execution - features that have been covered in this course.

SSH was first designed in 1995 by Finnish computer scientist Tatu Ylönen to replace the Telnet network protocol, choosing port 22 as default because it was between the FTP and Telnet ports .

> *"I designed SSH to replace both telnet (port 23) and ftp (port 21). Port 22 was free. It was conveniently between the ports for telnet and ftp. I figured having that port number might be one of those small things that would give some aura of credibility.[1]" - Tatu Ylonen*

Approach and Process SSH runs on top of the TCP/IP protocol suite and incorporates encryption and authentication via public key cryptography. Public key cryptography is a way to encrypt data, or sign data, with two different keys. In an SSH connection, both sides have a public/private key pair, and each side authenticates the other using these keys. While public key cryptography authenticates the connected devices in SSH, a properly secured computer will still require authentication from the person using SSH. Often this takes the form of entering a username and password. Once authentication is complete, the person can execute commands on the remote machine as if they were doing so on their own local machine.

Improvement on Telnet This encryption **addresses vulnerability** of earlier protocols such as Telnet, while maintaining the same core function of remote access and command line execution. Certainly security risks remain - with some attackers involving stealing of SSH keys in order to access private computers and servers.

[1] https://www.ssh.com/academy/ssh/port

4. Conclusion & Recommendations

The script development portion of the project serves to remind that there is a wide range of tools, commands and approaches to achieve a particular outcome. While engaging, the process can be labour intensive. It is also important to consider “simple things” such as what the user is viewing, apart from having the specific knowledge to produce a functioning script.

The use of anonymous connection and SSH in the script component dovetails with scope 2 of the project with a more explicit focus on the importance of security or secured protocols. This component highlights the wide range of tools and protocols with different impacts on security. The CIA triad remains a broad but still relevant high level perspective we can consider these implications. Students of this course who are new/would-be cybersecurity practitioners would be well placed to stay open and increase our breadth knowledge of these tools and protocols over time.

5. References

Various websites that provided guidance on this project are listed below.

1. https://www.cyberciti.biz/faq/apt-get-list-packages-are-installed-on-ubuntu-linux/
2. https://www.computerhope.com/unix/apt-mark.htm
3. https://ioflood.com/blog/bash-loop-through-array
4. https://superuser.com/questions/67765/sudo-with-password-in-one-command-line
5. https://unix.stackexchange.com/questions/459923/multiple-commands-in-ssh pass
6. https://superuser.com/questions/1358108/how-to-send-sudo-password-in-a-shell-script
7. https://howhttps.works/https-ssl-tls-differences/
8. https://www.infobyip.com/ipbulklookup.php
9. https://unix.stackexchange.com/questions/582933/when-connecting-via-ssh-does-the-diffie-hellman-key-exchange-take-place-over-an
10. https://www.techtarget.com/searchnetworking/tutorial/How-to-capture-and-analyze-traffic-with-tcpdump
11. https://datatracker.ietf.org/doc/html/rfc854
12. https://datatracker.ietf.org/doc/html/rfc15
13. https://www.geeksforgeeks.org/what-is-network-virtual-terminal-in-telnet/
14. https://www.britannica.com/technology/Telnet
15. https://www.cloudns.net/blog/telnet-explained-what-is-it-and-how-it-works/
16. https://www.acronis.com/en-sg/blog/posts/telnet/
17. https://www.cloudflare.com/en-gb/learning/access-management/what-is-ssh
18. https://www.ssh.com/academy/ssh/port