Network Research | Project: Network Remote Control

Kenneth Wong s32
CFC190324
Trainer: Samson

Table of Contents

Introduction

This report outlines a comprehensive approach to enhancing security and anonymity in network reconnaissance activities. It begins by emphasizing the importance of anonymity in protecting individuals' privacy, especially in the context of network reconnaissance. The report then introduces Nipe, a tool designed to route network traffic through the Tor network, thus providing bidirectional anonymous connections.

The script discussed in the report leverages Nipe to establish an anonymous connection to an SSH server, from which WHOIS and Nmap scans are performed on a targeted domain. This approach ensures that the reconnaissance activities are conducted anonymously, making them resistant to eavesdropping and traffic analysis.

The report delves into the comparison between SSH and Telnet, highlighting the significance of SSH as a secure protocol for remote administration compared to the unsecured Telnet protocol. It analyzes how each protocol contributes to the CIA triad, emphasizing confidentiality, integrity, and availability.

Furthermore, the report provides a detailed breakdown of the script's methodology, discussing its purpose, execution, and recommendations for usage. It also includes an analysis of the script's performance and effectiveness in achieving its objectives.

Overall, the report offers valuable insights into enhancing security and anonymity in network reconnaissance activities, emphasizing the importance of using secure protocols like SSH and tools like Nipe to protect privacy and explains how it plays a part in the CIA triad.

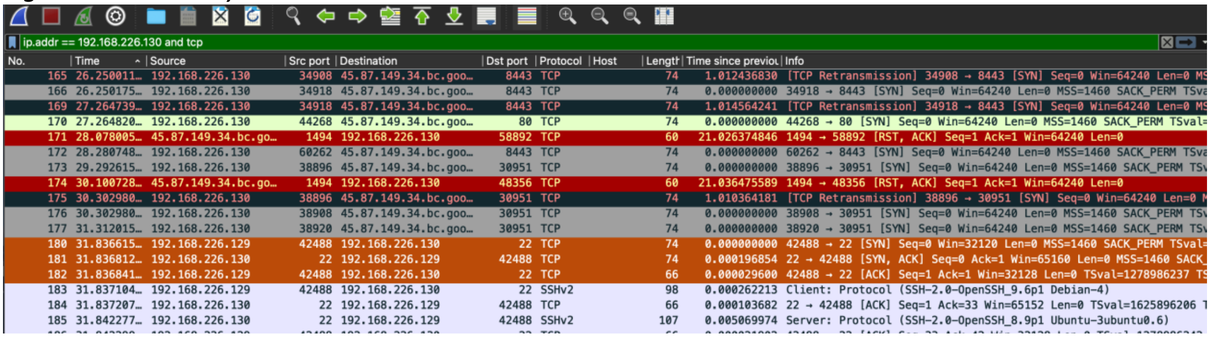
Wireshark analysis

This section delves into the examination of network traffic captured during the execution of the script. The script, crafted to enhance security and anonymity in network reconnaissance activities, leverages tools such as Nipe to establish anonymous connections via the Tor network. By routing traffic through Tor, the script aims to shield the identity and activities of the user, thereby fortifying privacy protections.

This analysis focuses on scrutinizing the network traffic generated by the script, with a particular emphasis on identifying and examining traffic related to SSH connections, Nipe routing, WHOIS queries, and Nmap scans. By isolating and analyzing these traffic patterns, we aim to ascertain the effectiveness of the script's methodology in preserving anonymity and ensuring secure reconnaissance activities.

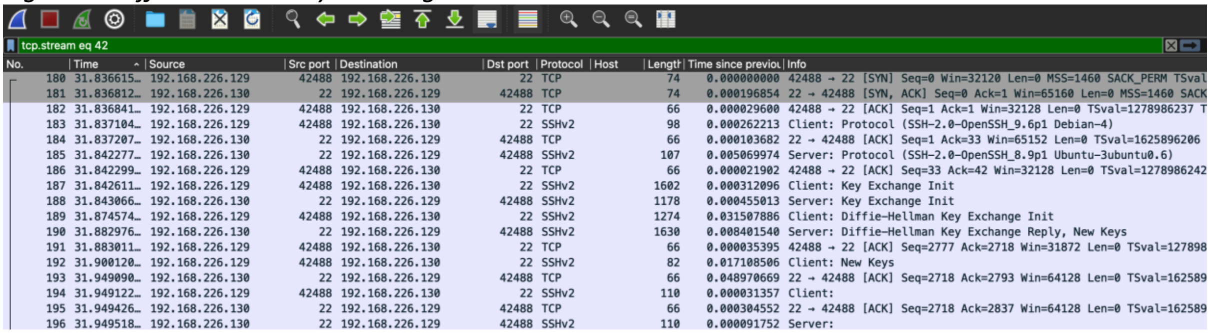
SSH

Figure 1: Three-way handshake with SSH server as seen in Wireshark.



After the user enters their credentials, the SSH client initiates a connection to the SSH server. During this process, Wireshark captures the standard TCP three-way handshake *(Figure 1)*, affirming the establishment of a TCP connection between the client and the SSH server. This handshake protocol, fundamental to TCP communication, signifies the synchronization of sequence numbers and acknowledgment of successful packet exchange, ensuring secure communication.

Figure 2: Diffie-Hellman key exchange as seen on Wireshark



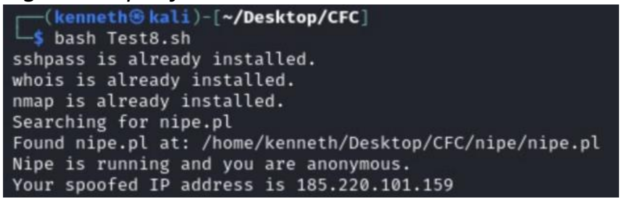
The Diffie-Hellman (DH) public key exchange cryptography algorithm *(Figure 2)* plays a pivotal role in the establishment of secure connections with SSH servers. As a key-exchange protocol, DH facilitates the establishment of a shared secret between two parties communicating over a public channel, without the need for transmitting the key over the Internet. This mutual key, generated through DH, empowers the communicating parties to utilize public keys for encryption and decryption, employing symmetric cryptography to safeguard their conversation or data. This sophisticated cryptographic mechanism ensures the confidentiality and integrity of the exchanged information, thus fortifying the security posture of SSH connections.

One significant advantage of Diffie-Hellman key exchange is its provision of Perfect Forward Secrecy (PFS). PFS guarantees the ongoing security of past session keys, even in scenarios where long-term private keys are compromised in the future. This prevents any retroactive decryption of previously encrypted communications. Additionally, the encryption algorithm employed by Diffie-Hellman makes it exceedingly challenging for eavesdroppers to discern the chosen encryption key.

Nipe

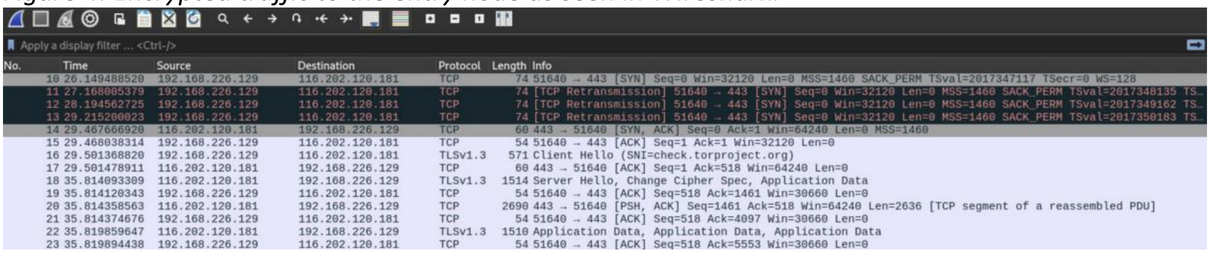
Wireshark will capture packets as usual, but the content of the packets will be encrypted if Nipe is successfully anonymizing the traffic through the Tor network. This means that while we can still see the metadata (such as packet headers, source and destination IP addresses, ports, etc.), the payload of the packets will be unreadable.

Figure 3: Spoofed IP address as seen on Kali



```
┌──(kenneth㉿kali)-[~/Desktop/CFC]
└─$ bash Test8.sh
sshpass is already installed.
whois is already installed.
nmap is already installed.
Searching for nipe.pl
Found nipe.pl at: /home/kenneth/Desktop/CFC/nipe/nipe.pl
Nipe is running and you are anonymous.
Your spoofed IP address is 185.220.101.159
```

Figure 4: Encrypted traffic to the entry node as seen in Wireshark.



No.	Time	Source	Destination	Protocol	Length	Info
10	26.149488520	192.168.226.129	116.202.120.181	TCP	74	51640 → 443 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=2017347117 TSecr=0 WS=128
11	27.168005379	192.168.226.129	116.202.120.181	TCP	74	[TCP Retransmission] 51640 → 443 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=2017348135 TS…
12	28.194562725	192.168.226.129	116.202.120.181	TCP	74	[TCP Retransmission] 51640 → 443 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=2017349162 TS…
13	29.215200023	192.168.226.129	116.202.120.181	TCP	74	[TCP Retransmission] 51640 → 443 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=2017350183 TS…
14	29.467666920	116.202.120.181	192.168.226.129	TCP	60	443 → 51640 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
15	29.468038314	192.168.226.129	116.202.120.181	TCP	54	51640 → 443 [ACK] Seq=1 Ack=1 Win=32120 Len=0
16	29.501368820	192.168.226.129	116.202.120.181	TLSv1.3	571	Client Hello (SNI=check.torproject.org)
17	29.501478911	116.202.120.181	192.168.226.129	TCP	60	443 → 51640 [ACK] Seq=1 Ack=518 Win=64240 Len=0
18	35.814093309	116.202.120.181	192.168.226.129	TLSv1.3	1514	Server Hello, Change Cipher Spec, Application Data
19	35.814120343	192.168.226.129	116.202.120.181	TCP	54	51640 → 443 [ACK] Seq=518 Ack=1461 Win=30660 Len=0
20	35.814358563	116.202.120.181	192.168.226.129	TCP	2690	443 → 51640 [PSH, ACK] Seq=1461 Ack=518 Win=64240 Len=2636 [TCP segment of a reassembled PDU]
21	35.814374676	192.168.226.129	116.202.120.181	TCP	54	51640 → 443 [ACK] Seq=518 Ack=4097 Win=30660 Len=0
22	35.819859647	116.202.120.181	192.168.226.129	TLSv1.3	1510	Application Data, Application Data, Application Data
23	35.819894438	192.168.226.129	116.202.120.181	TCP	54	51640 → 443 [ACK] Seq=518 Ack=5553 Win=30660 Len=0

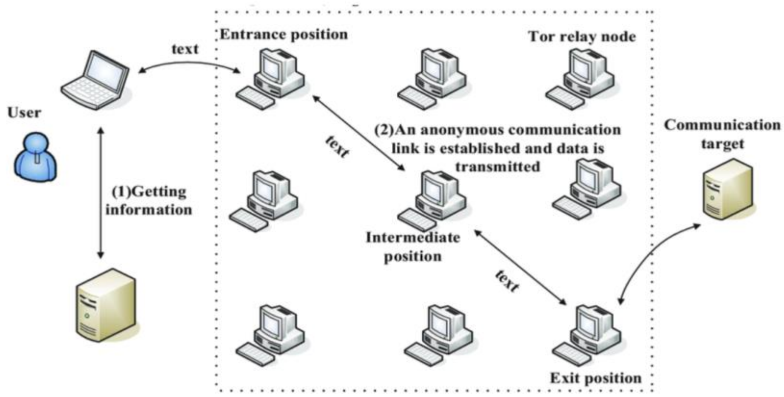
The IP address being spoofed, "185.220.101.159" as seen in Figure 3, represents the exit node in the Tor network. In a typical Tor network structure, there are three types of nodes: entry, middle, and exit nodes. The entry node, with IP address "116.202.120.181" in this case, is the initial relay where traffic enters the Tor network. Middle nodes serve as intermediate relays between the entry and exit nodes. Finally, the exit node is the last relay that communicates with the destination server on behalf of the user.

Wireshark can capture and log traffic related to the entry node on our Network Interface Card (NIC), but it cannot directly capture traffic from the exit node. This is because the exit node is part of the Tor network and handles communication with the destination servers independently of our NIC. Consequently, the traffic between the exit node and the destination server does not pass through our NIC, making it impossible for Wireshark to capture it.

In the accompanying image *(Figure 4)*, we can observe only the encrypted traffic to the entry node. Due to this encryption and the nature of the Tor network, Wireshark cannot display the multiple hops that Nipe makes within the Tor network.

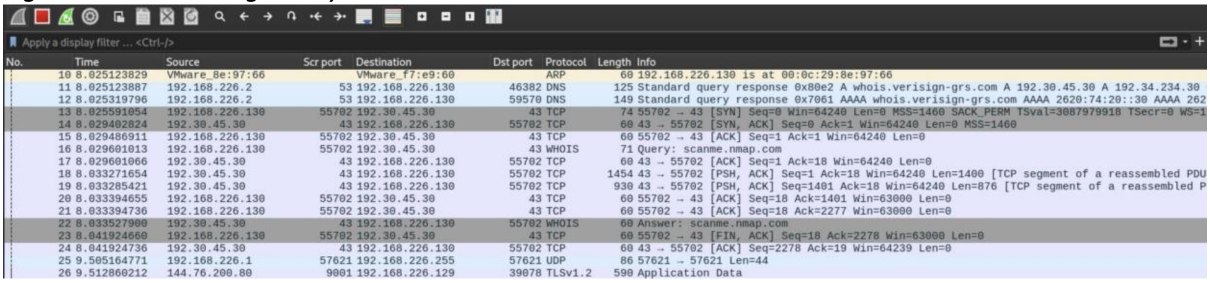
The diagram *(Figure 5)* illustrates the architecture of the Tor network, which is designed to enhance privacy and anonymity for users on the internet. In the Tor network, traffic is routed through a series of relays, including entry nodes, middle nodes, and exit nodes.

Figure 5: Diagram of the Tor network showing entry, middle, and exit nodes.



WHOIS and Nmap

Figure 6: Whois being run by the SSH server



Apply a display filter ... <Ctrl-/>

No.	Time	Source	Scr port	Destination	Dst port	Protocol	Length	Info
10	8.025123829	VMware_8e:97:66		VMware_f7:e9:60		ARP	60	192.168.226.130 is at 00:0c:29:8e:97:66
11	8.025123887	192.168.226.2	53	192.168.226.130	46382	DNS	125	Standard query response 0x80e2 A whois.verisign-grs.com A 192.30.45.30 A 192.34.234.30
12	8.025319796	192.168.226.2	53	192.168.226.130	59570	DNS	149	Standard query response 0x7061 AAAA whois.verisign-grs.com AAAA 2620:74:20::30 AAAA 262
13	8.025591054	192.168.226.130	55702	192.30.45.30	43	TCP	74	55702 → 43 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=3087979918 TSecr=0 WS=1
14	8.029402824	192.30.45.30	43	192.168.226.130	55702	TCP	60	43 → 55702 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
15	8.029486911	192.168.226.130	55702	192.30.45.30	43	TCP	60	55702 → 43 [ACK] Seq=1 Ack=1 Win=64240 Len=0
16	8.029601013	192.168.226.130	55702	192.30.45.30	43	WHOIS	71	Query: scanme.nmap.com
17	8.029601066	192.30.45.30	43	192.168.226.130	55702	TCP	60	43 → 55702 [ACK] Seq=1 Ack=18 Win=64240 Len=0
18	8.033271654	192.30.45.30	43	192.168.226.130	55702	TCP	1454	43 → 55702 [PSH, ACK] Seq=1 Ack=18 Win=64240 Len=1400 [TCP segment of a reassembled PDU
19	8.033285421	192.30.45.30	43	192.168.226.130	55702	TCP	930	43 → 55702 [PSH, ACK] Seq=1401 Ack=18 Win=64240 Len=876 [TCP segment of a reassembled P
20	8.033394655	192.168.226.130	55702	192.30.45.30	43	TCP	60	55702 → 43 [ACK] Seq=18 Ack=1401 Win=63000 Len=0
21	8.033394736	192.168.226.130	55702	192.30.45.30	43	TCP	60	55702 → 43 [ACK] Seq=18 Ack=2277 Win=63000 Len=0
22	8.033527900	192.30.45.30	43	192.168.226.130	55702	WHOIS	60	Answer: scanme.nmap.com
23	8.041924660	192.168.226.130	55702	192.30.45.30	43	TCP	60	55702 → 43 [FIN, ACK] Seq=18 Ack=2278 Win=63000 Len=0
24	8.041924736	192.30.45.30	43	192.168.226.130	55702	TCP	60	43 → 55702 [ACK] Seq=2278 Ack=19 Win=64239 Len=0
25	9.505164771	192.168.226.1	57621	192.168.226.255	57621	UDP	86	57621 → 57621 Len=44
26	9.512860212	144.76.200.80	9001	192.168.226.129	39078	TLSv1.2	590	Application Data

This analysis explores the behaviour of network traffic when WHOIS and Nmap commands are executed from an SSH server *(Figure 6)*. The SSH server, with the IP address 192.168.226.130, serves as the source of these requests. The goal is to understand how traffic is routed through the SSH connection and the implications for network monitoring and logging.

When connected to an SSH server and executing commands, the traffic for these commands is routed through the SSH connection. This setup means that the source of the traffic for these commands is the SSH server rather than the local machine from which the SSH session was initiated.

The WHOIS requests and Nmap scans are initiated directly from the SSH server (IP: 192.168.226.130).
Wireshark captures will show the SSH server's IP address as the source of these requests. This is expected behaviour since the commands are executed on the server.

In this scenario, the SSH server effectively acts as a proxy. All network traffic generated by WHOIS and Nmap commands will appear to originate from the server. The IP address 192.168.226.130 will be visible in Wireshark as the source IP for both WHOIS and Nmap traffic, reflecting the server's role in initiating these requests.

Since we used a secure connection like SSH, the traffic between the local machine and the SSH server is encrypted. This encryption ensures that intermediate devices and network monitors cannot easily access or interpret the data being transmitted. Due to this encryption, it is not possible to record and export the log files of the commands executed on the SSH server from the local machine using Wireshark.

If an unsecured protocol like FTP were used instead of SSH, network traffic, including log files and command outputs, would be transmitted in plaintext. In such cases, it would be possible to capture and export these files using Wireshark.

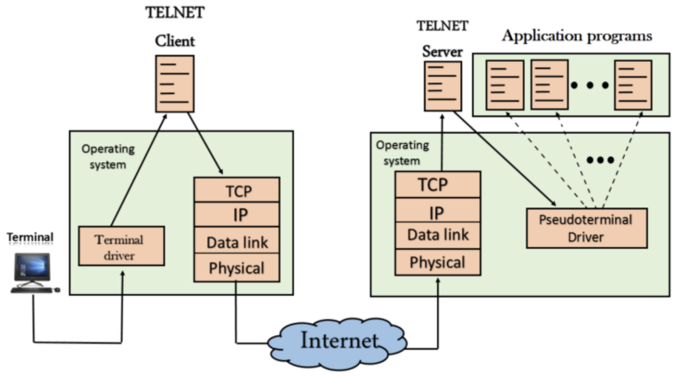
Telnet

Telnet is a network protocol that allows a user to remotely access and control another computer over the Internet or a local area network (LAN). It enables a user to establish a connection to a remote system and perform tasks as if they were sitting in front of that computer. It begins with the client initiating a connection to the Telnet server to start a remote session, typically on port 23.

Telnet does not have a distinct header structure like some other protocols. Instead, it operates by sending sequences of control characters and data across the TCP connection such as "DO," "DON'T," "WILL," and "WON'T." This negotiation process ensures that both the client and server agree on parameters such as terminal type and local echoing.

Once the connection is established and options are negotiated, the client can send commands to the server as if typing them directly on the server's terminal. The server executes these commands and sends back the results. This interaction is text-based, supporting the transmission of commands, messages, and data streams in a clear text format.

Figure 6: Telnet remote login



The history of Telnet dates back to the early days of computer networking. It was originally developed in the late 1960s as a way to allow users of one computer to connect to another computer and use its resources remotely. The name “Telnet” comes from “Terminal Network,” as the protocol was designed to allow users to access remote computers using a terminal or command-line interface.

Despite its declining popularity in recent years, Telnet still maintains relevance in certain applications, particularly within legacy systems and environments that necessitate remote access via a command-line interface. However, since SSH is vastly more secure than Telnet, there are some cases when we choose to use Telnet over SSH; when working on trusted networks (such as LANs) that are not connected to the Internet, and when working with devices that do not support SSH.

Nevertheless, Telnet continues to be used today due to its simplicity and widespread support, especially in network devices like routers, switches, and firewalls, where it serves as the primary method for remote management. Additionally, its utility extends to testing and troubleshooting network connections and services.

For instances where Telnet has to be used, below are some practices which we can use to enhance the security of our communications:

- VPN: Running Telnet over a Virtual Private Network (VPN) can secure the connection by encrypting all traffic between your computer and the VPN exit node, protecting your traffic from eavesdropping.
- IP Whitelisting: Restrict access to the Telnet server by allowing only specific IP addresses to connect. This prevents unauthorized access but does not protect against interception of the data being transmitted.
- Telnet Over TLS/SSL: Implementing Telnet over TLS or SSL can provide encryption for Telnet sessions. This requires configuring the Telnet server to support SSL and using a Telnet client that supports SSL connections.

Telnet operates on a client-server model, establishing connections where clients initiate interactions with Telnet servers on remote hosts, facilitating bidirectional communication. Portability and compatibility are intrinsic to Telnet, ensuring its accessibility across diverse operating systems and hardware platforms.

Administratively, Telnet facilitates remote management and administration of network devices, routers, servers, and switches, enabling centralized configuration, monitoring, and troubleshooting. Moreover, Telnet incorporates option negotiation mechanisms for terminal type and settings, ensuring compatibility and optimal display configurations between client and server. With error handling procedures in place, Telnet communications remain robust and reliable, contributing to its widespread adoption for remote system access and control.

To further enhance its functionality, Telnet benefits from the guidelines provided in RFC 5198, "Unicode Format for Network Interchange." RFC 5198 specifies the use of Unicode to support a wide range of characters and scripts, which is essential for internationalization. It recommends using UTF-8 as the encoding form for Unicode due to its efficiency, backward compatibility with ASCII, and wide support. By adhering to these standards, Telnet can handle international characters effectively, ensuring commands and responses are correctly interpreted across different systems.

RFC 5198 also defines how text should be represented to maintain consistency and avoid ambiguities, including the use of specific code points and sequences for characters and control codes. This is crucial for Telnet as it ensures that text commands and responses are accurately transmitted and interpreted. It also emphasizes text normalization using Normalization Form C (NFC), which combines characters into a single composite form, preventing discrepancies from different representations of the same character sequence.

Additionally, RFC 5198 specifies the use of the newline character (U+000A) to represent line endings in text transmitted over networks, ensuring that line breaks are consistently interpreted across different systems and platforms. By standardizing the use of Unicode and specifying consistent text representation rules, RFC 5198 aims to ensure compatibility between different systems and applications. For Telnet, this means that text data sent and received during a session will be accurately exchanged and displayed, promoting a seamless user experience across diverse environments.

SSH

The Secure Shell (SSH) protocol is a method for securely sending commands to a computer over the network. SSH uses cryptography to authenticate and encrypt connections between devices. SSH also allows for tunneling, or port forwarding, which is when data packets are able to cross networks that they would not otherwise be able to cross. SSH is often used for controlling servers remotely, transferring files, tunneling and port forwarding.

SSH is a critical component in today's industry, providing important security features and safeguarding sensitive information.

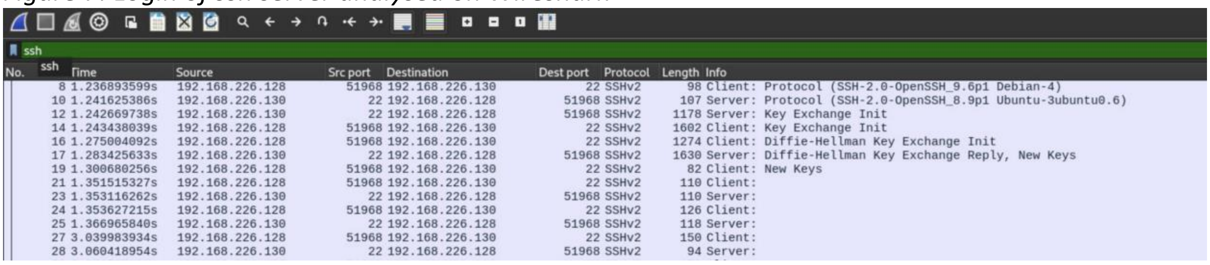
CIA Triad – Telnet and SSH

One of the key benefits in Telnet lies in its availability as it is universally supported. Additionally, it offers accessibility which provides convenient remote access which greatly benefits productivity.

Telnet's architecture lacks robust measures for ensuring data integrity, leaving transmitted data vulnerable to tampering and compromising confidentiality. Its reliance on basic username and password authentication exposes it to security risks such as brute-force attacks.

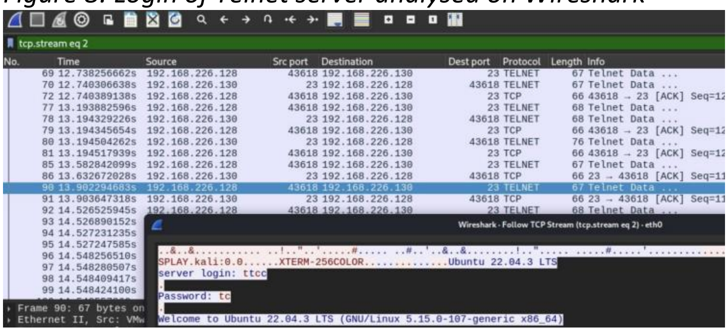
Additionally, limited logging capabilities hinder effective event monitoring, compromising confidentiality further. These weaknesses highlight the need for alternative secure protocols to mitigate risks and protect sensitive information effectively.

Figure 7: Login of ssh server analysed on Wireshark



In Figure 7, the SSH login process demonstrates the encryption applied to all data exchanged during authentication. This encryption ensures that sensitive information, such as usernames and passwords, remains obscured and secure from potential eavesdropping or interception which plays a role in confidentiality. Conversely, in Figure 8, depicting a Telnet login, the lack of encryption exposes login credentials in plaintext format during transmission. This stark contrast highlights SSH's robust security measures, offering an encrypted channel for authentication and data exchange, thereby safeguarding sensitive information from unauthorized access or malicious exploitation.

Figure 8: Login of Telnet server analysed on Wireshark



In addition, the Diffie-Hellman key exchange which SSH uses, plays a crucial role in upholding the principles of the CIA triad.

It ensures confidentiality by establishing a shared secret key between the client and server without directly transmitting the key over the network. This shared secret key is used to encrypt and decrypt data exchanged during the SSH session, protecting the confidentiality of sensitive information transmitted over the network.

Furthermore, it contributes to integrity by enabling the derivation of session keys that are used to protect the integrity of data exchanged during the SSH session. Through cryptographic mechanisms such as message authentication codes (MACs) or digital signatures, SSH ensures that data transmitted between the client and server remains unchanged and unaltered by malicious actors.

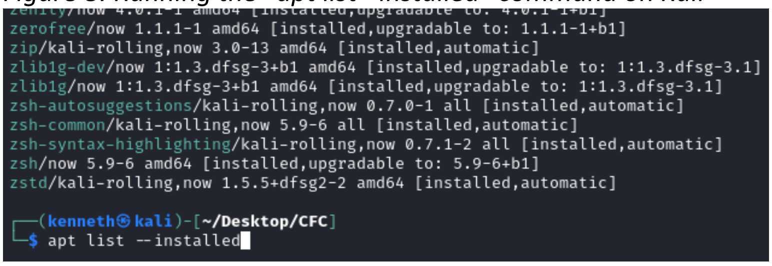
Even though Diffie-Hellman key exchange primarily focuses on confidentiality and integrity, its proper implementation also indirectly contributes to availability. By establishing secure and reliable communication channels between clients and servers, SSH ensures that authorized users can access network resources and services without disruption, thereby upholding the availability of critical systems and data.

Methodologies

The goal of this script is the defined with the aim of providing anonymous network scans in a secure and efficient way. Below is the explanation of the methodologies and thought process behind the selection and usage of key commands.

To ensure the script had all necessary packages, a function called "check_and_install_package" was developed. Initially, two methods were considered for package checking: using "apt list --installed" and "dpkg -l". While both methods could suffice, "dpkg -l" was chosen for its comprehensive and detailed package information. Nonetheless, either method could be employed for this purpose.

Figure 8: Running the "apt list –installed" command on Kali



Locating and integrating Nipe into the script proved to be the most challenging aspect. Despite employing various methods to verify its installation status, it was discovered that Nipe does not adhere to conventional Debian package installation as it was not installed using "apt-get". Initially, efforts were made to execute Nipe by providing a definitive path, yet this approach proved ineffective. Consequently, the decision was made to navigate into the Nipe folder for execution, facilitated by the 'find' function. Upon locating the folder, the presence of "nipe.pl" within was verified. In instances where it was absent, the script automatically initiated the installation for it.

Figure 9: Changing the directory to where "nipe.pl" is located

```
#~ Check if nipe path is not empty
if [ -n "$nipe_path" ]; then
    #~ Extract the directory containing nipe.pl
    nipe_dir=$(dirname "$nipe_path")

    #~ Change to the directory containing nipe.pl
    #~ If fail, print message and exit the script
    cd "$nipe_dir" || { echo "Failed to change directory to $nipe_dir."; exit 1; }
```

In the scripting process for logging scan details, a function was developed to accept two parameters: "domain" and "scan_type". In this function, local variables were created to store the parameter values, and a timestamp was generated using the date command. The timestamp, along with the scan type and domain information, was then concatenated to form a log entry. This entry was subsequently piped to the tee command for dual functionality: appending to the log file and suppressing standard output using "/dev/null". The initial execution of the function creates a new log file, while subsequent executions append data to the existing file.

Figure 10: Function to create a log of the scan

```
#~ Define log file
log_file="/var/log/domain_scan.log"

#~ Function to log scan details
log_scan(){
    #~ Define a local variable and assign it the first and second argument passed to the function
    local domain=$1
    local scan_type=$2
    #~ Assigning 'log_time' to current date and time formatted as "Day Month Date Time Year"
    local log_time=$(date "+%a %b %d %H:%M:%S %Y")
    #~ 'tee -a' to append to the log file and redirect standard output of tee to /dev/null
    echo "$log_time - $scan_type data collected for: $domain" | sudo tee -a $log_file > /dev/null
}
```

Figure 11: Log file created on host machine

```
┌──(kenneth㉿kali)-[/var/log]
└─$ cat domain_scan.log
Sat May 25 12:05:27 2024 - whois and nmap data collected for: cncworks.co.nz
Sat May 25 12:09:24 2024 - whois and nmap data collected for: cncworks.co.nz
Sat May 25 12:11:08 2024 - whois and nmap data collected for: scanme.nmap.com
Sun May 26 09:19:55 2024 - whois and nmap data collected for: scanme.nmap.com
Sun May 26 09:26:45 2024 - whois and nmap data collected for: scanme.nmap.com
```

To assess the status of Nipe and ascertain the user's anonymity status, the script employed a series of commands in with the output of "sudo perl nipe.pl status". Initially, the status output was captured and stored in the variable "nipe_status".

Subsequently, the "nipe_running" variable was extracted by parsing the relevant line from "nipe_status". This line indicates whether Nipe is currently running if it reflects "true", signifying that Nipe is operational, and the user's online activities are being anonymized through the Tor network.

Upon detecting that Nipe is running, the script outputs a message confirming that it is running and the user is anonymous. A brief pause of one second is introduced for readability before proceeding with further script execution.

Figure 12: Verifying that Nipe is running

```
#~ Check the status of Nipe and extract IP address and country
nipe_status=$(sudo perl nipe.pl status)
nipe_running=$(echo "$nipe_status" | head -n2 | tail -n1 | awk '{print $3}')
nipe_ip=$(echo "$nipe_status" | head -n3 | tail -n1 | awk '{print $3}')
ip_country=$(whois "$nipe_ip" | grep -i country | awk '{print $2}')

#~ Check if nipe is running
if [ "$nipe_running" = "true" ]; then
    echo "Nipe is running and you are anonymous."
    sleep 1
    echo "Your spoofed IP address is $nipe_ip"
    sleep 1
    echo "The country you are connected to is $ip_country"
    sleep 1
```

In the log in to SSH process where the user would have to provide their input to facilitate the process, the "-p" flag instructs the system to provide the password to the SSH server on behalf of the user.

To maintain confidentiality and security, the "-s" flag is employed with the read command for the SSH password input. This hides the user's input, ensuring that sensitive information, such as the password, remains hidden. Following this, the "echo" command is used to prompt the user to input other necessary details. Without echo, the user would not receive any prompt or indication to input the required information due to the "-s" flag.

Figure 12: Reading user input

```
#~ Prompt user to enter credentials to ssh into a server
read -p "Enter SSH username: " ssh_user
read -s -p "Enter SSH password: " ssh_password
echo
read -p "Enter SSH server address: " ssh_server
#~ Prompt user to enter the domain to scan and store the input in the variable target_domain
read -p "Enter domain or URL to scan:" target_domain
```

__Discussion__

```
#~ Function to check and install a package if not installed
check_and_install_package() {
    #~ Declares a local variable and assigns it the value of the first argument passed to the function
    local package_name=$1
    #~ Checks if the package by checking debian package (dpkg -l)
    #~ If the package is not found, then execute the code inside the 'if' block
    if ! dpkg -l | grep -q "$package_name"; then
        echo "$package_name is not installed. Installing $package_name..."
        #~ Update the package list to ensure we have the latest information about available packages
        sudo apt-get update
        #~ Install package with '-y' flag to automatically answer 'yes' to prompts
        sudo apt-get install -y "$package_name"
        echo "$package_name installed successfully."
    else
        #~ If package is already installed, print this
        echo "$package_name is already installed."
        #~ Pause for 1 second before continuing
        sleep 1
    fi
}
```

1. ! dpkg -l

This command is used to show which packages are not installed. If it is not, then execute then execute the code inside the 'if' block

2. sudo apt-get install -y "$package_name"

The "-y" flag is used to answer "yes" to prompts during installation for a more seamless user experience.

3. sleep 1

Pause the script for one second as it would be easier for the user to read the output.

```
#~ Function to log scan details
log_scan(){
    #~ Define a local variable and assign it the first and second argument passed to the function
    local domain=$1
    local scan_type=$2
    #~ Assigning 'log_time' to current date and time formatted as "Day Month Date Time Year"
    local log_time=$(date "+%a %b %d %H:%M:%S %Y")
    #~ 'tee -a' to append to the log file and redirect standard output of tee to /dev/null
    echo "$log_time - $scan_type data collected for: $domain" | sudo tee -a $log_file > /dev/null
}
```

4. local domain=$1

This keyword is used to declare local variables within a function. Variables declared with local are only accessible within the function in which they are defined. Here, local is used to declare "domain", "scan_type", and "log_time" as local variables.

5. echo "$log_time - $scan_type data collected for: $domain" | sudo tee -a $log_file > /dev/null

The "tee" command reads from standard input and writes to both standard output. The "-a" flag stands for append. When used with the "tee" command, it appends the output to the "/dev/null" file which is a special file that discards all data written to it. We use it when we do not want the output to be displayed on the terminal.

```
#~ Change to the directory containing nipe.pl
#~ If fail, print message and exit the script
cd "$nipe_dir" || { echo "Failed to change directory to $nipe_dir."; exit 1; }
```

6. cd "$nipe_dir" || { echo "Failed to change directory to $nipe_dir."; exit 1; }

In this line, the code will attempt to cd into "nipe_dir" which has been defined earlier in the script. If unable to do so, the command in the curly braces will be executed. "||" being the logical OR operator.

```
        #~ Connect to SSH server and run commands, saving output to files
        echo "Connecting to SSH server..."
        #~ Use sshpass to pass the SSH password and connect to the SSH server and run the following commands
        sshpass -p "$ssh_password" ssh -o StrictHostKeyChecking=no "$ssh_user@$ssh_server" << EOF
            echo "Remote SSH server IP address:" > $remote_whois_output
            ifconfig | grep 'inet ' | grep -v '127.0.0.1' | awk '{print \$2}' >> $remote_whois_output
            echo "Running whois on $target:" >> $remote_whois_output
            whois $target_domain >> $remote_whois_output
            echo "Running nmap scan on $target:" > $remote_nmap_output
            nmap $target_domain >> $remote_nmap_output
EOF
```

7. sshpass -p "$ssh_password" ssh -o StrictHostKeyChecking=no "$ssh_user@$ssh_server" << EOF

This line uses “sshpass” to provide the SSH password non-interactively. The “-o StrictHostKeyChecking=no” flag disables host key checking which prevents the SSH client from prompting the user to accept the host key, which prevents interruptions during the automated process.

Everything between the “<<EOF” and “EOF” is treated as input to the SSH session, which allows the commands to be executed on the SSH server.

```
#~ Verify the copied files
echo "Verifying the copied files..."
if [[ -f "whois_output.txt" ]]; then
    echo "whois output successfully copied to whois_output.txt"
else
    echo "Failed to copy whois output"
fi
if [[ -f "nmap_output.txt" ]]; then
    echo "nmap output successfully copied to nmap_output.txt"
else
    echo "Failed to copy nmap output"
fi
```

8. if [[-f “whois_output.txt”]]; then

The double square brackets are used to define a conditional expression for checking if there is the existence of a file named “whois_output.txt”. If true, the block of code would be executed.

Conclusion

Creating this script was a valuable learning experience, highlighting the significance of thoroughly documenting the purpose and functionality of each line. With this, it not only aids our understanding of the script if we were to revisit it in future, but also anyone’s understanding of it when they look at it.

Throughout the process, we learnt the importance of error handling and automating repetitive tasks to ensure a seamless user experience.

Overall, the experience highlighted to me, the importance of secure and anonymous network practices especially in environments where privacy is a non-negotiable. This script could serve as a convenient and efficient tool for pen-testers conducting anonymous reconnaissance and scanning activities, enhancing the ability to identify and assess potential security risks in target systems and networks.

Recommendation

While working on this script, we encountered significant challenges in debugging and troubleshooting. The behaviour of the script was inconsistent and sometimes it functioned correctly, despite no changes being made. To address these issues and enhance the script's reliability and maintainability, it is recommended to modularize the script.

Modularization involves breaking the script into smaller, reusable functions. This approach offers several advantages. First, it improves readability. When each function is dedicated to a specific task, the script becomes easier to read and understand. This is also good practice for anyone who may need to work on the script in the future. Second, it enables easier maintenance. Modifications can be made to individual functions without affecting the rest of the script. This reduces the risk of introducing new errors and simplifies the update process.

Furthermore, modularization allows for easier debugging. By testing and debugging functions independently, issues can be identified and resolved more efficiently. This leads to more consistent and reliable performance. Additionally, enhanced reusability is another significant benefit. Functions can be reused in other scripts or contexts, saving time and effort in future projects.

For instance, the package installation process can be separated into its own function. By doing so, the installation logic is encapsulated within a single, easily maintainable block of code.

In addition to modularization, automating the process of restarting Nipe until it successfully connects to the Tor network can significantly improve the script's usability. Currently, users need to bash the script multiple times before it starts working correctly. This manual intervention is both time-consuming and frustrating. Automating this process would ensure that Nipe continues to attempt to connect until it is successful, without requiring constant user input.

To implement this automation, a loop can be added to the script that checks the status of Nipe and restarts it if it is not running. This loop would continue until Nipe is successfully connected to the Tor network.

Another potential improvement could be made by using the “-Pn” flag with Nmap. Currently, the script runs Nmap without this option, which means it first pings the target to see if it is up before scanning it. However, in many cases, firewalls or security settings might block ping requests, causing Nmap to incorrectly assume that the host is down and skip the scan. With “-Pn” option, the ping check is skipped and proceeds directly to scanning the domain.

References

OpenAI. (2024). *ChatGPT* (GPT-4). Retrieved from https://chatgpt.com/

Anonymous Connection. In *ScienceDirect*. Retrieved May 27, 2024, from https://www.sciencedirect.com/topics/computer-science/anonymous-connection#:~:text=The%20program%20passes%20the%20information,to%20require%20usernames%20and%20passwords.

Diffie-Hellman key exchange. In Wikipedia. Retrieved May 27, 2024, from https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

HYPR. Diffie-Hellman algorithm. Retrieved May 27, 2024, from https://www.hypr.com/security-encyclopedia/diffie-hellman-algorithim#:~:text=The%20Diffie%E2%80%93Hellman%20(DH)%20Algorithm%20is%20a%20key%2D,being%20transmitted%20over%20the%20Internet.

PhoenixNAP. Diffie-Hellman key exchange. Retrieved May 28, 2024, from https://phoenixnap.com/blog/diffie-hellman-key-exchange#:~:text=The%20main%20benefit%20of%20the,it%20over%20an%20untrusted%20channel

ClouDNS. Telnet explained: What is it and how it works. Retrieved May 28, 2024, from https://www.cloudns.net/blog/telnet-explained-what-is-it-and-how-it-works/

PhoenixNAP. Telnet vs SSH. Retrieved May 28, 2024, from https://phoenixnap.com/kb/telnet-vs-ssh

Cloudflare. What is SSH? Retrieved May 28, 2024, from https://www.cloudflare.com/en-gb/learning/access-management/what-is-ssh/

Cloudflare. What is tunneling? Retrieved May 28, 2024, from https://www.cloudflare.com/en-gb/learning/network-layer/what-is-tunneling/

Coinmonks. (2018). Tor nodes explained. Retrieved May 28, 2024, from https://medium.com/coinmonks/tor-nodes-explained-580808c29e2d

ResearchGate. Tor network structure diagram. Retrieved May 28, 2024, from https://www.researchgate.net/figure/Tor-network-structure-diagram_fig3_319490703

Postel, J. B., & Reynolds, J. K. (1983). RFC 854: Telnet protocol specification. Retrieved May 29, 2024, from https://datatracker.ietf.org/doc/html/rfc854#:~:text=INTRODUCTION%20The%20purpose%20of%20the,oriented%20processes%20to%20each%20other

RFC Editor. (1983). RFC 8540. Retrieved May 29, 2024, from https://www.rfc-editor.org/rfc/rfc8540

Klensin, J. (2008). RFC 5198: Unicode format for network interchange. Retrieved May 30, 2024, from https://datatracker.ietf.org/doc/html/rfc5198

Javatpoint. Introduction to Telnet. Retrieved May 29, 2024, from https://www.javatpoint.com/computer-network-telnet

ClouDNS. Telnet explained: What is it and how it works. Retrieved May 29, 2024, from https://www.cloudns.net/blog/telnet-explained-what-is-it-and-how-it-works/

GeeksforGeeks. Introduction to Telnet. Retrieved May 29, 2024, from https://www.geeksforgeeks.org/introduction-to-telnet/

MakeComputerScienceGreatAgain. (2018). Understanding Telnet: A deeper dive into remote communication. Retrieved May 29, 2024, from https://medium.com/@MakeComputerScienceGreatAgain/understanding-telnet-a-deeper-dive-into-remote-communication-0998fa93dc40

Stevens, W. R. (2012). TCP/IP Illustrated, Volume 1: The Protocols (p. 602). Addison-Wesley.

Howtonetwork. CIA Triad. Retrieved May 29, 2024, from https://www.howtonetwork.com/free-ccna-study-guide-ccna-book/cia-triad/

NIST. (2006). Guide to General Server Security. (NIST Special Publication 800-123). Retrieved May 30, 2024, from https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-123.pdf

Tenable. UNIX patch auditing over Telnet. Retrieved May 30, 2024, from https://pt-br.tenable.com/blog/unix-patch-auditing-over-telnet

Baeldung. List installed packages in Linux. Retrieved May 30, 2024, from https://www.baeldung.com/linux/list-installed-packages#:~:text=Another%20way%20is%20to%20use,tool%20queries%20the%20dpkg%20database.&text=The%20%2Dl%20option%20lists%20all%20the%20packages%20installed%20on%20our%20system.

Stack Overflow. How can I set the current working directory to the directory of the script in Bash? Retrieved May 30, 2024, from https://stackoverflow.com/questions/3349105/how-can-i-set-the-current-working-directory-to-the-directory-of-the-script-in-ba

Stack Overflow. How would I create a text file and fill it with sudo permission in Bash? Retrieved May 30, 2024, from https://stackoverflow.com/questions/42810888/how-would-i-create-a-text-file-and-fill-it-with-sudo-permission-in-bash