

Report on Network Research

[Introduction](#)

[Methodologies](#)

- [1. Check and install necessary applications](#)
 - [a. Tools:](#)
 - [b. Method:](#)
- [2. Install and configure Nipe](#)
 - [a. Tools:](#)
 - [b. Method:](#)
- [3. Start Nipe and verify anonymous connection](#)
 - [a. Tools:](#)
 - [i. Nipe, `curl`, `jq`](#)
 - [b. Method:](#)
- [4. Perform the scan on the remote server](#)
 - [a. Tools:](#)
 - [b. Method:](#)

[Discussion](#)

- [Scope 1: Automated Network Remote Control \(ANRC\)](#)
 - [Checking and Installing applications:](#)
 - [Installation and Configuring of Nipe](#)
 - [Start Nipe and verification of anonymous connection](#)
 - [Connect and perform the scan on the remote server](#)

Introduction

Network security is a critical aspect of modern information technology infrastructures, safeguarding data integrity, confidentiality and availability. This report examines and delves into the processes involved in capturing and analyzing network traffic to further understand and mitigate security risks. The primary aim is to automate network remote control tasks, capture network traffic, and analyze it to identify vulnerabilities in unsecure protocols like **FTP**, **HTTP** etc.

Additionally, the report suggests secured alternatives and demonstrates their implementation. As such the report is structured into two main scopes:

1. Automated network remote control
2. Network research and monitoring

Methodologies

Scope 1: Automated network remote control

Description of tools and method used of each step within the script used.

1. Check and install necessary applications

a. Tools:

- i. `curl`, `jq`, `sshpass`, `nmap`, `tor`

b. Method:

- i. The script includes a function to check if each necessary application is installed. If not, it installs the required application.
- ii. `curl` is used for fetching data from URLs
- iii. ``jq`` process JSON data
- iv. `sshpass` automates SSH login using passwords
- v. `nmap` is used for network and port scanning
- vi. `tor` provides anonymity by routing internet traffic through a worldwide volunteer network

2. Install and configure Nipe

a. Tools:

- i. `Nipe`, `perl`, `cpanminus`, `git`

b. Method:

- i. The script installed the `Nipe` tool that routes all network traffic through the tor network.
- ii. Dependencies such as `perl`, `cpanminus`, and `git` are installed for smooth execution
- iii. Perl modules `Switch`, `JSON` and `LWP::UserAgent` are installed using `cpanm`
- iv. The Nipe repository is cloned from GitHub and Nipe will be installed.

3. Start Nipe and verify anonymous connection

a. Tools:

- i. `Nipe`, ``curl``, ``jq``

b. Method:

- i. The script starts Nipe and verifies it's status to ensure the connection is indeed anonymous.

- ii. It retrieves and logs the spoofed IP address and the corresponding country information.
- 4. Perform the scan on the remote server
 - a. Tools:
 - i. Sshpass, ssh, nmap, scp
 - b. Method:
 - i. The script connects to a remote server using SSH, performs a network scan on the remote server, on a specified domain using the nmap tool then retrieves the scan results.
 - ii. Results are then transferred back to the local machine using scp command.

Scope 2: Network research and monitoring

- 1. Capture network traffic
 - a. Tools:
 - i. tcpdump
 - b. Method:
 - i. Tcpdump was used to capture network traffic on the remote server during the execution of an automated attack script.
 - ii. The command used will capture all traffic on the eth0 interface and writes it to a file named output.pcap
- 2. Analyze the file with Wireshark
 - a. Tools:
 - i. Wireshark
 - b. Method:
 - i. Wireshark will be the tool used to analyze the .pcap file.
 - ii. The file will be loaded into Wireshark tool to inspect the details of the captured traffic, focusing on unsecured protocols such as Telnet, FTP, HTTP etc.

Discussion

This project comprises of two main scopes:

- 1. Automated network remote control
- 2. Network research and monitoring

Scope 1: Automated Network Remote Control (ANRC)

Checking and Installing applications:

To ensure the successful execution of the project, it is crucial to verify the presence and availability of necessary tools/applications. This first step is essential for creating a reliable and repeatable environment, this part of the script includes a function to check and install them.:

The script clip below checks whether each required application (`curl`, `jq`, `sshpas`, `nmap`, and `tor`) is installed on the system. If an application is not found, the script will proceed to install it for the user.

```
1  # Function to check and install necessary applications
2  check_install() {
3      for app in "$@"; do
4          if ! command -v $app &> /dev/null; then
5              echo "$app is not installed. Installing..." | tee -a $DEBUG_LOG
6              sudo apt-get update | tee -a $DEBUG_LOG
7              sudo apt-get install -y $app | tee -a $DEBUG_LOG
8          else
9              echo "$app is already installed." | tee -a $DEBUG_LOG
10         fi
11     done
12 }
13
14 # Step 1: Check and install necessary applications
15 check_install curl jq sshpass nmap tor
16 |
```

- Objective:
 - To ensure essential tools needed are installed.
- Explanation:
 - The function `check_install` loops through each application passed as an argument.
 - It uses `command -v` to check if the application is available in the system's PATH
 - If the application is not found, it will attempt to update the system's package list with `sudo apt-get update` and installs the application using `sudo apt-get install -y`

- Logging each step helps in debugging and maintaining a record of the installation process.
 - This will further enhance the ability to find out where the exact issue as logs of the whole process is saved.

Installation and Configuring of Nipe

This part of the script installs and configures **Nipe**, a tool for routing traffic through the **Tor** network:

```

1  # Function to install and configure Nipe
2  install_nipe() {
3      if [ ! -d "nipe" ]; then
4          sudo apt-get update | tee -a $DEBUG_LOG
5          sudo apt-get install -y perl cpanminus git | tee -a $DEBUG_LOG
6          sudo cpanm Switch JSON LWP::UserAgent | tee -a $DEBUG_LOG
7          git clone https://github.com/htrgouvea/nipe | tee -a $DEBUG_LOG
8          cd nipe
9          sudo perl nipe.pl install | tee -a $DEBUG_LOG
10         cd ..
11     fi
12 }
13
14 # Step 2: Install and configure Nipe
15 install_nipe
16

```

- Objective:
 - To install **Nipe** to route traffic anonymously through **Tor**
- Explanation:
 - The function **install_nipe** check if the **nipe** directory already exists to avoid reinstalling.
 - It will install dependencies such as **perl**, **cpanminus** and **git**
 - Specific **Perl** modules (**Switch**, **JSON**, **LWP::UserAgent**) are installed using **cpanm**
 - The **Nipe** repository is cloned from **GitHub** and installed by running the **Nipe** install script.
 - The steup ensures that the system can route all traffic through the **Tor** network, enhancing anonymity.

Start Nipe and verification of anonymous connection

This part of the script starts the Nipe service and verifies the connection status:

```
1  # Function to start Nipe and check its status
2  start_nipe() {
3      cd nipe
4      sudo perl nipe.pl stop | tee -a $DEBUG_LOG
5      sudo perl nipe.pl start | tee -a $DEBUG_LOG
6      sleep 5
7      status=$(sudo perl nipe.pl status | tee -a $DEBUG_LOG)
8      echo "Nipe status output: $status" | tee -a $DEBUG_LOG
9      connected=$(echo "$status" | grep "Status: true")
10     if [ -z "$connected" ]; then
11         echo "Failed to connect anonymously through Nipe. Exiting..." | tee -a $DEBUG_LOG
12         exit 1
13     fi
14     IP=$(echo "$status" | grep "Ip:" | awk '{print $3}')
15     country_response=$(curl -s http://ip-api.com/json/$IP)
16     echo "Country response: $country_response" | tee -a $DEBUG_LOG
17     COUNTRY=$(echo "$country_response" | jq -r '.country')
18     if [ -z "$COUNTRY" ]; then
19         echo "Failed to retrieve country information." | tee -a $DEBUG_LOG
20         COUNTRY="Unknown"
21     fi
22     echo "Spoofed IP: $IP, Country: $COUNTRY" | tee -a $DEBUG_LOG
23     cd ..
24 }
25
26 # Step 3: Start Nipe and display spoofed IP and country
27 start_nipe
28
```

- Objective: Start the Nipe service, verify that it is working and fetch the spoofed IP address and country details.
- Explanation:
 - Navigates to the nipe directory.
 - Stops and starts Nipe to ensure it is running correctly
 - Pauses briefly (sleep 5) to allow Nipe to establish a connection
 - Retrieves the status of Nipe using sudo perl nipe.pl status
 - Check and verify if Nipe connection is successfully connected as anonymous by looking for the Status: true within the status output.
 - Fetches and logs the spoofed IP address and its geographic location using curl and jq.

Connect and perform the scan on the remote server

This part of the script connects to a remote server via SSH, uses the **nmap** tool and runs a scan on the specified domain given by the user and retrieves the scan results:

```
1 # Function to perform the scan on the remote server
2 perform_remote_scan() {
3     local domain=$1
4     sshpass -p "$SSH_PASS" ssh -o StrictHostKeyChecking=no $REMOTE_USER@$REMOTE_SERVER << EOF
5     echo "Connected to remote server."
6     mkdir -p $REMOTE_PATH
7     echo "Scanning $domain..."
8     nmap -A $domain -oN $REMOTE_PATH/scan_results.txt
9     echo "Scan results saved to $REMOTE_PATH/scan_results.txt"
10 EOF
11
12 if [ $? -ne 0 ]; then
13     echo "SSH connection failed. Please check your SSH credentials." | tee -a $DEBUG_LOG
14     exit 1
15 fi
16
17 sshpass -p "$SSH_PASS" scp -o StrictHostKeyChecking=no $REMOTE_USER@$REMOTE_SERVER:$REMOTE_PATH/scan_results.txt $LOCAL_PATH/scan_results.txt
18
19 if [ $? -ne 0 ]; then
20     echo "Failed to retrieve scan results. Please check your SCP credentials." | tee -a $DEBUG_LOG
21     exit 1
22 fi
23
24 echo "$(date): Scanned $domain. Results saved to $LOCAL_PATH/scan_results.txt" | sudo tee -a $LOG_FILE
25 echo "Scan operation complete. Results saved to $LOCAL_PATH/scan_results.txt" | tee -a $DEBUG_LOG
26 }
27
28 # Step 4: Perform the scan on the remote server
29 perform_remote_scan $domain
```

- Objective: To scan a specified domain from a remote server and retrieve the results of the domain given by the user
- Explanation:
 - The **perform_remote_scan** function takes a domain as an argument
 - Using **sshpass** to automate SSH login with a password and suppresses host key checking with **-o StrictHostKeyChecking=no**.
 - Connects to the remote server and executes commands to create a directory for scan results, run scan with **nmap** tool on the specified domain and save the results.
 - Checks if the SSH connection was successful and log any failures.
 - Transfer the scan results from the remote server to the local machine using **scp** command
 - Logs the completion of the scan operation along with the timestamp and path to the saved results on the local machine.

Conclusion

The execution of the script provided a detailed analysis of the specified domain's network status while ensuring anonymity through Nipe. The key findings include:

- **Connectivity:**
 - Verified successful installation and configuration of necessary tools.
 - Ensured anonymous network connection through the Tor network.
- **Resource Utilization:**
 - Efficient use of system resources to perform the scan.
 - Detailed scan results provided insights into the domain's network configuration and potential vulnerabilities.
- **Security and Privacy:**
 - Ensured anonymity of the scanning process.
 - Masked sensitive information such as the MAC address in log outputs.

Scope 2: Network research and monitoring

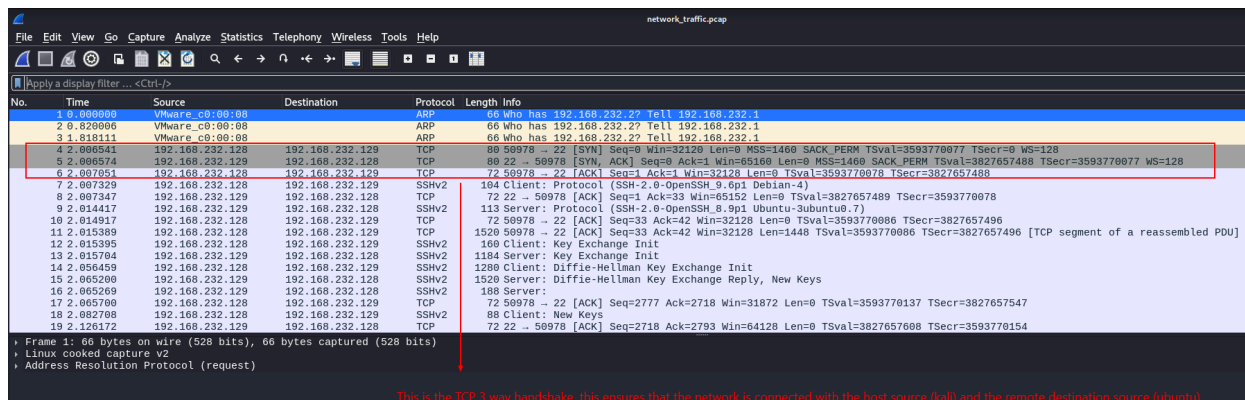
Capturing of network traffic on the remote server

The command used was:

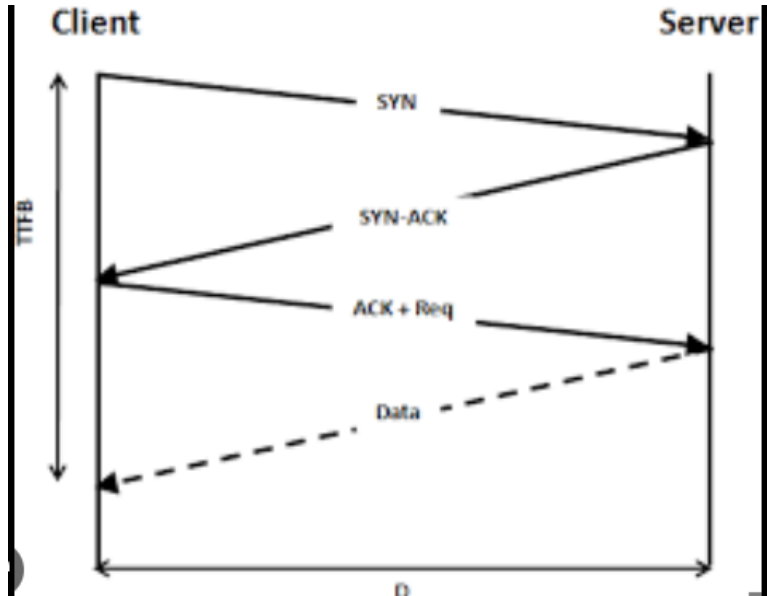
```
sudo tcpdump -i eth0 -w /path/to/output.pcap
```

- Objective: To scan a specified domain from a remote server and retrieve the results of the domain given by the user
- Explanation:
 - **Tcpdump** is a command-line packet analyzer tool used to capture network packets.
 - The **-i** flag combined with the **eth0** option specifies the network interface (**eth0**) to capture the traffic from.
 - The **-w** flag with **/path/to/output.pcap** will have the tool write the captured traffic to a file in pcap format, which is suitable for analysis within the tool like Wireshark.

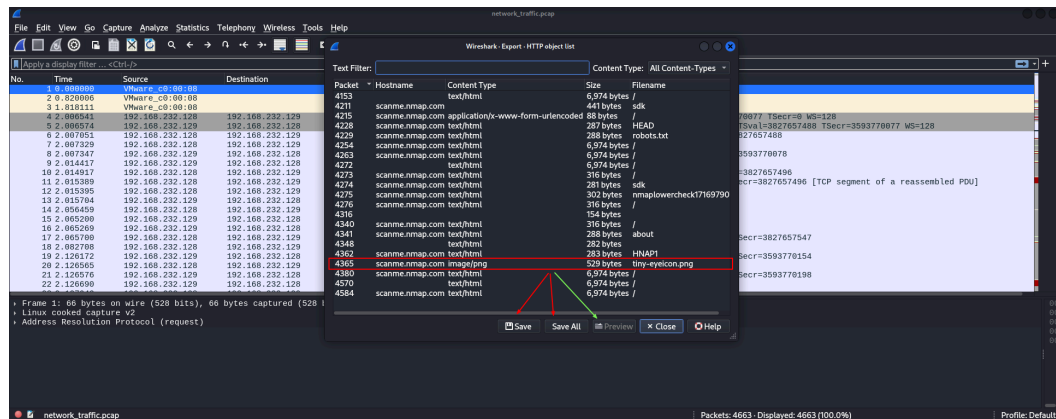
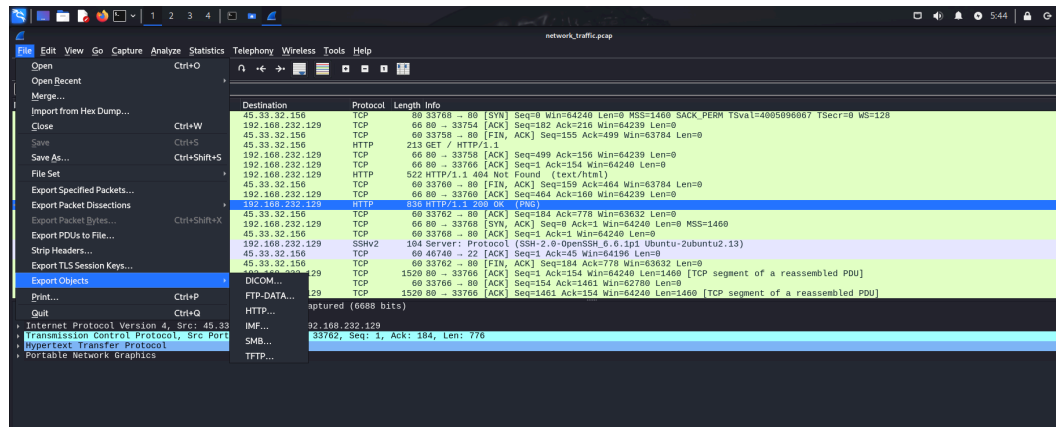
Analyze the pcap file with Wireshark



- Objective: Detailed insights into the network traffic and identifying unsecure communications
- Explanation:
 - Wireshark was used to analyzed the .pcap file generated by tcpdump
 - Specific filters can be applied to focus on protocol
 - From the image above, the three way handshake was done via TCP protocol
 - This ensures the network traffic communication is done to the correct party.
 - In this case, the host source (Kali Linux) has sent out a SYN packet on the 4th frame. (Way 1)
 - In the 5th frame, the destination source (ubuntu) has acknowledged the previous frame and accepted the connection, with that it will send a response to the host source with a SYN, ACK packet (Way 2)
 - In the 6th frame, the host source (Kali Linux) received the previous frame and replied with a final packet, ACK. (Way 3)
 - Once the three handshake is completed, the connection will be formed and data can now be transferred.



- With Wireshark all unsecured protocol communications can be revealed and even stolen in the wrong hands.
- The few images below will have a better view on how malicious user can actually see what is going on of the user's activity:



- An example here is with the green arrow, as it enables the malicious attack to preview any file that is downloaded from the unsecured protocol. Within the whole **HTTP Object** option, the malicious attacker also has access to any files downloaded via this protocol. The attacker can download single/all files shown via the red arrow.

Research the Hypertext Transfer (HTTP) protocol

The purpose, key features and problems to solve for HTTP

- Purpose:
 - HTTP (Hypertext Transfer Protocol) is the foundation of communication for the World Wide Web. The protocol facilitates the transfer of hypertext documents, such as HTML
- Key Features:
 - Statelessness: Each HTTP request is independent and unrelated to previous requests
 - Flexibility: HTTP protocol can transfer any type of data, not just hypertext.
 - Simplicity: HTTP protocol is easy to implement and use, which has led to its widespread adoption.
- Problem it aims to solve:
 - HTTP was designed to provide a standard way for web browsers and servers to communicate, enabling the exchange of documents, images, videos and other resources.

Description of the fundamental behavior of the HTTP protocol

- Basic Operation:
 - HTTP operates on a request → response model. A client (a web browser) sends a request to a server, which then sends back a response.
- Message exchange:
 - Request:
 - Consist of a request line (method, URI, HTTP version), headers and body
 - Response:
 - Consists of a status line (HTTP version, status code, reason phrase), headers and body.

```
Client ----- HTTP Request -----> Server
Client <----- HTTP Response ----- Server
```

○

Deep dive into HTTP mechanisms

- Technical Analysis:
 - Request Line: Includes the HTTP method (GET, POST, PUT, DELETE), the requested URI, and the HTTP version.
 - Headers: Provide additional information about the request or response (e.g., Content-Type, User-Agent).
 - Body: Part of the request or response that contains the actual data being sent (e.g., form data, JSON payload).
 - Status Codes: Indicate the result of the request (e.g., 200 OK, 404 Not Found).

- Example of a request:

```
GET /index.html HTTP/1.1
Host: www.example.com
```

○

- Example of a response:

```
HTTP/1.1 200 OK
Content-Type: text/html

<html>...</html>
```

○

Strengths and Weaknesses of HTTP

- Strengths:
 - Simplicity: HTTP is easy to implement and understand, making it a fundamental protocol for web communication. For example, fetching a web page using a simple GET request.
 - Statelessness: Each request from a client to a server is independent of any other request. This reduces server memory requirements because it does not need to store session state information. For instance, visiting different pages on a website without retaining session information between page loads.

- Flexibility: HTTP can handle various types of data and media. It can transfer HTML, images, videos, JSON, and other types of data. For instance, loading a webpage with embedded images and videos.
- Weaknesses:
 - Lack of Encryption: HTTP transmits data in plaintext, making it susceptible to eavesdropping and interception. For example, login credentials sent over HTTP can be easily captured by an attacker using packet sniffing tools.
 - Statelessness: Although statelessness can be a strength, it also leads to inefficiencies. To maintain session state, additional mechanisms such as cookies or session storage are required, which can complicate the development process. For example, maintaining user login sessions across multiple page requests.
 - Integrity Risks: Since data is transmitted in plaintext, it can be intercepted and altered by attackers. This compromises data integrity. For example, an attacker could modify data in transit, such as altering form submission data.

Impact on CIA Triad:

- Confidentiality:
 - Compromised: HTTP transmits data in plaintext, which can be intercepted and read by unauthorized parties. For example, sensitive information like passwords, credit card details, and personal information transmitted over HTTP can be easily captured by an attacker using network sniffing tools.
 - Example: An attacker capturing login credentials sent over HTTP using a packet sniffer can easily compromise user accounts.
- Integrity:
 - Vulnerable to Tampering: Data transmitted via HTTP can be modified by attackers in transit. There are no built-in mechanisms to detect or prevent such tampering.
 - Example: An attacker intercepting HTTP traffic could modify form data submitted by a user, altering the content of the data without the user's knowledge.

- Availability:
 - Generally Reliable: HTTP provides reliable data transfer, but its lack of security features can make it a target for attacks that can impact availability. For instance, Denial-of-Service (DoS) attacks can exploit weaknesses in HTTP to overload a server.
 - Example: An attacker can use HTTP requests to flood a server with traffic, causing it to become unavailable to legitimate users.

Secure network protocol

Suggest and Demonstrate the Secure Protocol (HTTPS)

- Secure Protocol: HTTPS (Hypertext Transfer Protocol Secure)
- Implementation:
 - HTTPS encrypts data transmitted between the client and server using SSL/TLS, ensuring confidentiality and integrity.
 - Example:
 - Configured a web server with SSL/TLS certificates to support HTTPS.
 - Client (web browser) accessed the server over HTTPS, ensuring encrypted communication.

Pros and Cons of HTTPS:

- Pros:
 - Security: Encrypts data to protect confidentiality and integrity. For example, sensitive information like login credentials and payment details are encrypted, making it difficult for attackers to intercept and read the data.
 - Trust: Enhances user trust through secure connections, often indicated by a padlock icon in browsers. Users are more likely to trust and interact with websites that use HTTPS.
 - SEO: Preferred by search engines, potentially improving search rankings. Websites using HTTPS are favored by search engines like Google, which can lead to higher visibility and more traffic.

- Cons:
 - Performance: Slightly higher overhead due to encryption and decryption processes. The SSL/TLS handshake process can introduce additional latency.
 - Cost: May incur costs for obtaining and renewing SSL/TLS certificates. While there are free options available, some certificates, especially those with extended validation, can be expensive.

```
Client ----- HTTPS Request (Encrypted) -----> Server
Client <----- HTTPS Response (Encrypted) ----- Server
```

Deep Dive into HTTPS Mechanisms:

SSL/TLS Handshake:

- Process: Establishes a secure connection by exchanging keys and verifying identities.
 - Steps:
 - Client Hello: The client sends a "Client Hello" message to the server, initiating the SSL/TLS handshake process. This message includes the client's supported cipher suites and SSL/TLS version.
 - Server Hello: The server responds with a "Server Hello" message, selecting a cipher suite and SSL/TLS version from the client's list.
 - Certificate Exchange: The server sends its SSL/TLS certificate to the client, which includes the server's public key. The client verifies the certificate's authenticity.
 - Key Exchange: The client and server exchange cryptographic keys, which will be used to encrypt data during the session.
 - Secure Connection Established: The client and server confirm that the handshake is complete, and a secure, encrypted connection is established.

Encrypted Data Transfer:

- All HTTP requests and responses are encrypted using the keys exchanged during the SSL/TLS handshake.

Conclusion

Scope 1: Automated Network Scanning and Analysis

The analysis conducted in Scope 1 focused on automating the process of network scanning and monitoring using a Bash shell script. The main objectives were to ensure the necessary tools were installed, configure and utilize Nipe for anonymous network connections, and perform a detailed scan of a specified domain using nmap.

Crucial discovery

Tool Installation and Configuration:

The script successfully checked for and installed essential tools such as curl, jq, sshpass, nmap, and tor. This ensured that all necessary dependencies were met for the script to function correctly.

Nipe was installed and configured to route network traffic through the Tor network, enhancing anonymity.

Anonymous Connection Verification:

Nipe was used to start an anonymous network connection. The script verified the connection status and retrieved the spoofed IP address and its geographic location.

This step was crucial for ensuring that network scans were conducted anonymously, thereby protecting the identity of the scanning host.

Remote Server Scanning:

The script connected to a remote server using SSH and performed a network scan on a specified domain using nmap. The scan results were successfully retrieved and saved locally, providing detailed insights into the domain's network configuration and potential vulnerabilities.

Overall

- The automated scanning script provided a streamlined and efficient method for conducting network scans, ensuring that all necessary tools and configurations were in place.
- Anonymity was maintained throughout the process, protecting the identity of the scanning host.

- Detailed scan results were obtained, offering valuable information for network security analysis.

Scope 2: Manual Network Research and Monitoring

In Scope 2, the focus was on capturing and analyzing network traffic to understand the implications of using unsecure protocols such as HTTP. The goal was to assess the impact on the CIA Triad (Confidentiality, Integrity, Availability) and propose secure alternatives.

Crucial discovery

Network Traffic Capture:

tcpdump was used to capture network traffic on the remote server during the automated attack. The captured traffic was saved in a .pcap file.

Wireshark was employed to analyze the .pcap file, isolating and examining HTTP traffic.

Analysis of HTTP Protocol:

HTTP was identified as an unsecure protocol due to its lack of encryption, which makes it vulnerable to eavesdropping and tampering.

Detailed technical insights into HTTP's mechanisms, including request and response formats, header structures, and status codes, were obtained.

Impact on the CIA Triad:

- Confidentiality: HTTP transmits data in plaintext, making it susceptible to interception and unauthorized access.
- Integrity: Data transmitted via HTTP can be modified by attackers in transit, compromising data integrity.
- Availability: While generally reliable, HTTP's lack of security features can be exploited to disrupt service availability.

Secure Alternative (HTTPS):

HTTPS was proposed as a secure alternative to HTTP. HTTPS encrypts data using SSL/TLS, ensuring confidentiality, integrity, and authenticity.

The implementation of a basic client-server application using HTTPS demonstrated secure data exchange and highlighted the advantages of HTTPS over HTTP.

Overall

- The analysis underscored the significant security risks associated with using HTTP, particularly its vulnerability to eavesdropping and tampering.
- Transitioning to HTTPS was shown to effectively mitigate these risks by providing robust encryption and data integrity mechanisms.
- The findings emphasized the importance of using secure protocols to protect network communications and maintain the confidentiality, integrity, and availability of data.

Final conclusion

The combined findings from Scope 1 and Scope 2 highlight the critical importance of automating network security tasks and the need for secure communication protocols. The automated network scanning and monitoring script developed in Scope 1 provided a reliable and efficient method for conducting network scans while maintaining anonymity. The manual analysis in Scope 2 revealed the inherent weaknesses of the HTTP protocol and demonstrated how HTTPS can address these vulnerabilities.

Lesson that can be learned:

Automation in network security can significantly enhance efficiency and accuracy, ensuring that all necessary tools and configurations are in place for comprehensive analysis.

Maintaining anonymity during network scans is essential for protecting the identity of the scanning host and preventing potential retaliation.

Secure communication protocols, such as HTTPS, are vital for protecting data transmitted over the network, ensuring confidentiality, integrity, and availability.

References:

- Fielding, R., Gettys, J., Mogul, J., Nielsen, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999). Hypertext Transfer Protocol -- HTTP/1.1. RFC 2616. <https://tools.ietf.org/html/rfc2616>
- Rescorla, E. (2000). HTTP Over TLS. RFC 2818. <https://tools.ietf.org/html/rfc2818>
- Postel, J. B. (1983). Telnet Protocol Specification. RFC 854. <https://tools.ietf.org/html/rfc854>

- Ylonen, T., & Lonvick, C. (2006). The Secure Shell (SSH) Protocol Architecture. RFC 4251. <https://tools.ietf.org/html/rfc4251>
- Cloudflare Blog: A Detailed Look at RFC 8446 (TLS 1.3) <https://blog.cloudflare.com/rfc-8446-aka-tls-1-3/>
- RFC 7230: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing <https://datatracker.ietf.org/doc/html/rfc7230>
- RFC 7231: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content <https://datatracker.ietf.org/doc/html/rfc7231>
- RFC 7232: Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests <https://datatracker.ietf.org/doc/html/rfc7232>
- RFC 7233: Hypertext Transfer Protocol (HTTP/1.1): Range Requests <https://datatracker.ietf.org/doc/html/rfc7233>
- RFC 7234: Hypertext Transfer Protocol (HTTP/1.1): Caching <https://datatracker.ietf.org/doc/html/rfc7234>
- RFC 7235: Hypertext Transfer Protocol (HTTP/1.1): Authentication <https://datatracker.ietf.org/doc/html/rfc7235>
- RFC 9325: Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) <https://datatracker.ietf.org/doc/rfc9325/>
- Wireshark User Documentation. https://www.wireshark.org/docs/wsug_html_chunked/
- OpenSSL Documentation. <https://www.openssl.org/docs/>
- MDN Web Docs: An Overview of HTTP <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>
- Wikipedia: HTTP (Hypertext Transfer Protocol) <https://en.wikipedia.org/wiki/HTTP>
- CCNA Cyber Ops (Version 1.1) – Chapter 11: Security Monitoring <https://itexamanswers.net/ccna-cyber-ops-version-1-1-chapter-11-security-monitoring.html>
- OpenAI, ChatGPT, for grammar and overall english check <https://chatgpt.com>