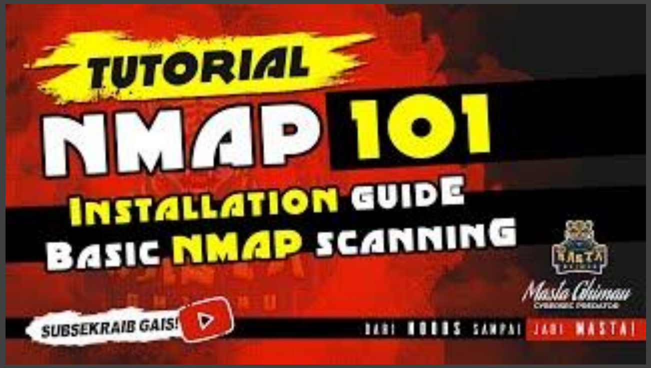
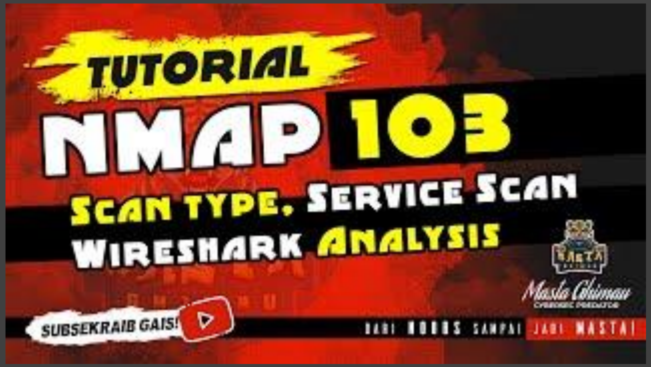
COSMOLOGY BEYOND NMAP

WHO AM I?

https://www.youtube.com/@mastaghimau





HACKER'S MANIFESTO

Volume One, Issue 7, Phile 3 of 10

Yes, I am a criminal. My crime is that of curiosity. My crime is that of judging people by what they say and think, not what they look like.

My crime is that of outsmarting you, something that you will never forgive me for.

I am a hacker, and this is my manifesto. You may stop this individual, but you can't stop us all... after all, we're all alike.

The Conscience of a Hacker
by The Mentor
Written on January 8, 1986

INTRODUCE YOURSELF

Name:

University:

Experience in Cyber Security:

Experience in using nmap:

Expectations from this short course:

WHAT IS NMAP?

Introduction:

Nmap, short for "**Network Mapper**," is a powerful and versatile open-source tool designed for **network exploration and security auditing**. Developed by **Gordon Lyon**, commonly known as **Fyodor**, Nmap has become a go-to solution for cybersecurity professionals, network administrators, and ethical hackers.

WHAT IS NMAP?

Analogy:

Think of Nmap as a **skilled detective equipped with a myriad of tools**, visiting a neighborhood to **map out every house, identify its occupants, and understand the security measures in place**.

In the world of cybersecurity, Nmap plays the role of this detective, **revealing the details of devices on a network**, the **services they offer**, and **potential vulnerabilities**.

WHAT IS NMAP?

Key Features:

Host Discovery	Nmap can determine **which hosts are active on a network**, akin to figuring out which houses are occupied in our detective analogy.
Port Scanning	It **identifies open ports on a target system**, resembling the detective checking doors and windows for potential entry points.
Service Version Detection	Nmap **can determine the type and version of services running on open ports**, providing insights into the software in use.
Operating System Detection	Like our detective identifying the lifestyle of residents by observing their homes, Nmap **can infer the operating system running on a device**.

WHAT IS NMAP?

Use Cases and Scenarios:

Security Auditing

A company wants to **assess the security of its servers**. Nmap can be used to **identify open ports and services, revealing potential vulnerabilities** that malicious actors might exploit.

Penetration Testing

Ethical hackers **simulate cyber-attacks to identify weaknesses in a client's network**. Nmap aids in **mapping the network landscape, guiding further penetration testing efforts**.

WHAT IS NMAP?

Use Cases and Scenarios:

Network Mapping

A network administrator **needs a comprehensive map of devices and services on the network**. Nmap **helps create an inventory, aiding in efficient network management**.

Vulnerability Assessment

A security team wants to **proactively identify and address vulnerabilities**. Nmap scans can **pinpoint potential weak points in the network infrastructure**.

WHAT IS NMAP?

Use Cases and Scenarios:

Intrusion Detection and Prevention

An organization wants to **monitor its network for unauthorized access**. Nmap can be employed to **detect unexpected open ports or services that might indicate a security breach**.

WHAT IS NMAP?

The Need for Nmap in Security Assessment:

Visibility

Nmap provides **a clear view of what's happening on a network**, offering visibility into devices, services, and potential vulnerabilities.

Efficiency

It streamlines the security assessment process, allowing professionals to **focus on critical areas and prioritize remediation efforts**.

WHAT IS NMAP?

The Need for Nmap in Security Assessment:

Proactive Defense

By **identifying potential weaknesses**, Nmap empowers organizations to a**ddress vulnerabilities before they can be exploited by malicious actors**.

Risk Reduction

Nmap aids in **reducing the overall risk by uncovering hidden or overlooked security issues**, enabling proactive risk management.

WHAT IS NMAP?

Uncovering hidden or overlooked security issues:

Undiscovered Open Ports

Unintentionally open port on a server.

```
nmap -p 1-1000 <target_ip>
```

Obsolete or Unpatched Services

Vulnerable ColdFusion Server

```
nmap -p 80 --script http-vuln-cve2010-2861 <target_ip>
```

WHAT IS NMAP?

Uncovering hidden or overlooked security issues:

Unauthorized Services

Unauthorized file-sharing service

```
nmap -p 1-1000 --script smb-enum-shares <target_ip>
```

Misconfigured Firewalls

Firewall allowing external access to a database.

```
nmap -p 3306 <target_ip>
```

WHAT IS NMAP?

Uncovering hidden or overlooked security issues:

Hidden Devices

Unauthorized personal device

```
nmap -sn <subnet>
```

Insecure Protocols

Outdated FTP server

```
nmap -p 21 --script ftp-anon <target_ip>
```

WHAT IS NMAP?

Uncovering hidden or overlooked security issues:

Default Configurations

Network switch with default credentials.

```
nmap -p 161 --script snmp-brute <target_ip>
```

INSTALLATION AND SETUP

Windows

Installer:

- Visit the official Nmap download page on the Nmap website.
- Download the latest stable version for Windows (e.g., nmap-7.x-installer.exe).
- Run the installer and follow the on-screen instructions.

Environment Variables (Optional):

- Add the Nmap installation directory to the system's PATH environment variable for easier command-line access.

Verification:

- Open Command Prompt and type `nmap` to verify the installation.

INSTALLATION AND SETUP

Linux (General - Source Installation)

Install required dependencies:

```
sudo apt-get install build-essential libssl-dev libncurses5-dev libpcap-dev
```

Download and Compile:

- Download the source tarball from the official Nmap download page.
- Extract the tarball and navigate to the extracted directory.

INSTALLATION AND SETUP

Linux (General - Source Installation) - cont

Run the following commands:

```
./configure
make
sudo make install
```

Verification:

- Open a new terminal and type `nmap` to verify the installation.

INSTALLATION AND SETUP

Ubuntu

Package Manager:

- Run the following commands:

```
sudo apt-get update
sudo apt-get install nmap
```

Verification:

- Type `nmap` in the terminal to verify the installation.

INSTALLATION AND SETUP

CentOS

Package Manager:

- Run the following command:

```
sudo yum install nmap
```

Verification:

- Type `nmap` in the terminal to verify the installation.

INSTALLATION AND SETUP

macOS

Homebrew (Package Manager):

- Install Homebrew (if not already installed)

```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Install Nmap:

```
brew install nmap
```

Verification:

- Open a terminal and type `nmap` to verify the installation.

INSTALLATION AND SETUP

Chocolatey (Windows)

Install Chocolatey:

- Open Command Prompt as Administrator.
- Run the following command to install Chocolatey:

```
Set-ExecutionPolicy Bypass -Scope Process -Force;
[System.Net.ServicePointManager]::SecurityProtocol =
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-
Object
System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))
```

INSTALLATION AND SETUP

Chocolatey (Windows) - cont

Install Nmap:

- After installing Chocolatey, run the following command to install Nmap:

```
choco install nmap
```

Verification:

- Open Command Prompt and type `nmap` to verify the installation.

INSTALLATION AND SETUP

Installing from Source	Using a Package Manager
Source Code: You download the source code of the software from the official website or a version control system.	**Binary Packages:** The package manager downloads precompiled binary packages from a repository. These packages are often optimized for the specific operating system and architecture.
Compilation: You manually compile the source code on your system. This involves running commands like ./configure, make, and make install. This process builds the executable binaries from the source code.	**Installation:** The installation process is streamlined. You typically use a single command (e.g., apt-get, yum, brew, choco) to install the software and its dependencies.
Dependencies: You need to manually install any dependencies required by the software. If a library or tool is missing, the compilation process may fail.	**Dependency Management:** The package manager automatically handles dependencies. If a package requires other libraries or tools, they are automatically installed.

INSTALLATION AND SETUP

Installing from Source	Using a Package Manager
Customization: Installing from source allows for customization. You can often configure compilation options to suit your needs.	**Updates:** Updating installed software is usually a single command, and the package manager takes care of fetching and installing the latest version from the repository.
Upgrading: Upgrading to a newer version involves repeating the download, compilation, and installation steps manually.	**Uninstallation:** Uninstalling software is also simplified. You use a package manager command to remove the software and its associated files.

INSTALLATION AND SETUP

Installing from Source	Using a Package Manager
Pros: More control over compilation options, customization, and potential optimization for specific hardware.	**Pros:** Easy installation, automatic dependency management, streamlined updates, and uninstallation.
Cons: Manual management of dependencies, potentially longer installation process, and less streamlined upgrade and uninstallation procedures.	**Cons:** Limited customization compared to source installations, potential delays in receiving the latest software versions.

The choice between installing from source and using a package manager depends on your preferences, system requirements, and the level of control and customization you need. In general, package managers are favored for simplicity and ease of management, while source installations are preferred when customization or specific compilation options are crucial.

INSTALLATION AND SETUP

Using **Snap** to install Nmap provides a convenient way to manage software packages on Linux systems. However, there are a few considerations and potential issues associated with Snap installations:

Advantages:
Ease of Installation: Snap simplifies the installation process by bundling dependencies and isolating applications in a containerized environment.
Automatic Updates: Snap applications can be set to update automatically, ensuring that you have the latest version without manual intervention.
Isolation: Snap packages are isolated from the rest of the system, reducing potential conflicts with other installed software.

INSTALLATION AND SETUP

Using **Snap** to install Nmap provides a convenient way to manage software packages on Linux systems. However, there are a few considerations and potential issues associated with Snap installations:

Considerations:
Resource Usage: Snap packages may use more disk space compared to traditional package manager installations because they contain their dependencies.
Strict Confinement: Snap applications run in a sandboxed environment by default, which means they have limited access to the system. This confinement may restrict certain functionalities.
Network Access: Snap applications might have restricted network access by default, potentially affecting Nmap's ability to perform certain types of scans. You may need to adjust permissions.

LAB EXERCISE 1

BASIC COMMAND STRUCTURE

The basic command structure of Nmap involves specifying the target(s), selecting various scan types, and customizing the scan according to your objectives.

Basic Syntax:
`nmap [Scan Type(s)] [Options] [Target(s)]`

Scan Types:	
Specify the type of scan you want to perform. Common scan types include:	
-sn: Ping scan (host discovery)	-sV: Service version detection
-sP: Same as -sn	-A: Aggressive scan (combination of various scans)
-sS: TCP SYN scan (default)	... and many more.
-sT: TCP connect scan	
-sU: UDP scan	

BASIC COMMAND STRUCTURE

The basic command structure of Nmap involves specifying the target(s), selecting various scan types, and customizing the scan according to your objectives.

Basic Syntax:
`nmap [Scan Type(s)] [Options] [Target(s)]`

Options:	
Fine-tune the scan by adding various options. Options start with a hyphen (-) followed by a letter or word. Some common options include:	
`-p: Specify ports or port ranges to scan.`	`-oX: Save results in XML format.`
`-O: Enable OS detection.`	`-v: Increase verbosity`
`-sC: Run default scripts.`	`-T: Set the timing template`
`--script: Specify scripts to run.`	`... and many more`
`-oN: Save results in normal format.`	

BASIC COMMAND STRUCTURE

The basic command structure of Nmap involves specifying the target(s), selecting various scan types, and customizing the scan according to your objectives.

Basic Syntax:
nmap [Scan Type(s)] [Options] [Target(s)]

Target(s):
Specify the target(s) you want to scan. This can be an IP address, hostname, IP range, or a combination.
`Single IP: nmap 192.168.1.1`
`Range: nmap 192.168.1.1-50`
`CIDR notation: nmap 192.168.1.0/24`
`Hostname: nmap example.com`

BASIC SCANNING TECHNIQUES

Ping Scan (Host Discovery):

Ping scanning, also known as host discovery, is used to identify live hosts on a network. It sends ICMP Echo Requests (ping) to the target hosts and determines which hosts respond.

```
nmap -sn 192.168.1.1
```

- -sn: Specifies a Ping Scan.
- 192.168.1.1: Replace with the target IP address or range.

BASIC SCANNING TECHNIQUES

TCP Scan:

TCP scanning involves checking for open TCP ports on a target host. It sends TCP SYN packets to specific ports and analyzes responses to identify open ports.

```
nmap -sS -p 1-1000 192.168.1.1
```

- -sS: Specifies a TCP SYN Scan.
- -p 1-1000: Scans ports 1 through 1000.
- 192.168.1.1: Replace with the target IP address.

BASIC SCANNING TECHNIQUES

UDP Scan:

UDP scanning is used to identify open UDP ports on a target host. It sends UDP packets to specific ports and analyzes responses to identify open ports.

```
nmap -sU -p 1-1000 192.168.1.1
```

- -sU: Specifies a UDP Scan.
- -p 1-1000: Scans UDP ports 1 through 1000.
- 192.168.1.1: Replace with the target IP address.

RUNNING NMAP

Running Nmap with administrator or root privileges provides **additional capabilities and access to certain features that may not be available to a regular user**. Here are the key differences between running Nmap with elevated privileges and as a normal user:

Normal User	Administrator or Root Privileges
Port Scanning: Regular users can perform basic port scanning on their local machine or on remote systems for which they have permission. They can discover open ports on target systems using non-privileged port numbers.	**Raw Socket Access:** Administrator/root privileges provide raw socket access, enabling Nmap to send and receive raw packets. This is essential for various advanced scanning techniques.
Service Version Detection: Limited ability to detect service versions on open ports, as some service information may be restricted.	**Reserved Ports Scanning:** Allows scanning of services using well-known and reserved ports, including those with numbers below 1024.

RUNNING NMAP

Running Nmap with administrator or root privileges provides **additional capabilities and access to certain features that may not be available to a regular user**. Here are the key differences between running Nmap with elevated privileges and as a normal user:

Normal User	Administrator or Root Privileges
Script Execution: Depending on the system's configuration, some Nmap scripts may require elevated privileges to execute successfully.	**OS Detection:** Enhanced ability to perform OS detection, as raw socket access is required for certain advanced OS fingerprinting techniques.
	Aggressive Scanning: Administrator/root access is often necessary for executing aggressive scans (-A option) that include OS detection, version detection, script scanning, and traceroute.

RUNNING NMAP

Running Nmap with administrator or root privileges provides **additional capabilities and access to certain features that may not be available to a regular user**. Here are the key differences between running Nmap with elevated privileges and as a normal user:

Normal User	Administrator or Root Privileges
	Increased Script Execution: Certain Nmap scripts require elevated privileges to access and interact with system resources. Running Nmap as an administrator or root user facilitates the execution of a broader range of scripts.
	Deeper Network Exploration: Administrator/root access allows for more extensive network exploration, including the ability to send specific types of ICMP messages and manipulate certain low-level network parameters.

RAW SOCKET IN NMAP

A raw socket in Nmap **allows direct access to the data link layer of the network stack**, enabling the **construction and manipulation of packets**.

A raw packet is a network packet at the raw data link layer without higher-level protocol encapsulation. It includes the packet's raw binary data, including the Ethernet or IP header, as needed.

Normal packets are typically constructed and sent using higher-level abstractions provided by the operating system's network stack (e.g., using sockets with TCP/IP or UDP). Raw packets, on the other hand, allow for more direct control over packet content and headers.

RAW SOCKET IN NMAP

Purpose of Raw Packets in Nmap
Packet Crafting: Nmap can craft packets with specific characteristics, including setting custom flags, manipulating headers, or constructing packets for less common protocols.
Port Scanning Techniques: Raw sockets are crucial for certain advanced port scanning techniques, such as TCP SYN scanning (-sS option), where crafted TCP SYN packets are sent to target hosts without completing the full TCP handshake.
Operating System Detection: Raw sockets are used in Nmap's operating system detection (-O option) to send packets with subtle variations that different operating systems might respond to differently.

RAW SOCKET IN NMAP

Purpose of Raw Packets in Nmap
Packet Crafting: Nmap can craft packets with specific characteristics, including setting custom flags, manipulating headers, or constructing packets for less common protocols.
Port Scanning Techniques: Raw sockets are crucial for certain advanced port scanning techniques, such as TCP SYN scanning (-sS option), where crafted TCP SYN packets are sent to target hosts without completing the full TCP handshake.
Operating System Detection: Raw sockets are used in Nmap's operating system detection (-O option) to send packets with subtle variations that different operating systems might respond to differently.

LAB EXERCISE 2

DEFAULT SCAN

By default, when you use **nmap to scan a target IP address (e.g., nmap x.x.x.x), it performs a TCP SYN scan**. The TCP SYN scan is one of the most common and widely used scan types in Nmap. It is also known as a "half-open" or "stealth" scan because it does not complete the full TCP handshake.

In a TCP SYN scan:

- Nmap **sends a TCP SYN packet to the target port**.
- If the **port is open, the target responds with a TCP SYN-ACK packet**.
- If the **port is closed, the target responds with a TCP RST (reset) packet**.
- Nmap analyzes the responses to determine the state of the port.

SHORT BREAK

SERVICE VERSION DETECTION

Service version detection is a **critical aspect of advanced scanning techniques in Nmap**. It involves **identifying the specific version and characteristics of services running on open ports**. By determining the service versions, security analysts can gain **insights into potential vulnerabilities, apply appropriate patches, and enhance overall network security**.

Understanding Service Version Detection:

Service version detection in Nmap involves **sending specific probes to open ports and analyzing the responses**. **Nmap compares these responses against a comprehensive fingerprint database to determine the service and its version**. This technique goes beyond identifying open ports and reveals the exact software and version running on those ports.

SERVICE VERSION DETECTION

Benefits of Service Version Detection:
Vulnerability Assessment: Accurate service version detection helps identify known vulnerabilities associated with specific software versions.
Patch Management: Knowing the service versions enables administrators to apply the latest patches and updates, reducing the risk of exploitation.
Network Inventory: Creating a detailed inventory of service versions facilitates better network management and security policy implementation.

SERVICE VERSION DETECTION

Nmap Options for Service Version Detection:

Default Service Version Detection:

```
nmap -A x.x.x.x
```

The -A option enables aggressive scanning, including service version detection.

Service Version Detection Only:

```
nmap -sV x.x.x.x
```

The -sV option specifically focuses on service version detection.

SERVICE VERSION DETECTION

Version detection is enabled and controlled with the following options:

`-sV (Version detection)`

Enables version detection. Alternatively, you can use -A, which enables version detection among other things.

`--allports (Don't exclude any ports from version detection)`

By default, **Nmap version detection skips TCP port 9100 because some printers simply print anything sent to that port**, leading to dozens of pages of HTTP GET requests, binary SSL session requests, etc. This behavior can be changed by modifying or removing the Exclude directive in nmap-service-probes, or **you can specify --allports to scan all ports regardless of any Exclude directive.**

SERVICE VERSION DETECTION

Version detection is enabled and controlled with the following options:

`--version-intensity <intensity> (Set version scan intensity)`

When performing a version scan (-sV), **Nmap sends a series of probes, each of which is assigned a rarity value between one and nine**. The intensity level specifies which probes should be applied. **The higher the number, the more likely it is the service will be correctly identified**. However, **high intensity scans take longer**. The **intensity must be between 0 and 9**. The **default is 7**.

`--version-light (Enable light mode)`

This is a **convenience alias for --version-intensity 2**. This light mode **makes version scanning much faster, but it is slightly less likely to identify services**.

SERVICE VERSION DETECTION

Version detection is enabled and controlled with the following options:
`--version-all (Try every single probe)` An **alias for --version-intensity 9**, ensuring **that every single probe is attempted against each port**.
`--version-trace (Trace version scan activity)` This causes Nmap to print out extensive debugging info about what version scanning is doing. It is a subset of what you get with --packet-trace.

LAB EXERCISE 3

OPERATING SYSTEM DETECTION

Operating System (OS) detection is a **crucial component of advanced scanning techniques in Nmap**. It involves analyzing network responses to identify the underlying operating system of a target. **By accurately determining the OS, security analysts can tailor their strategies, apply appropriate security measures, and gain a deeper understanding of the target environment**.

Understanding OS Detection:

OS detection in Nmap is performed **by analyzing subtle variations in how different operating systems respond to network probes**. Nmap **uses a combination of techniques, including TCP/IP stack fingerprinting, to match responses against a database of known OS signatures**.

OPERATING SYSTEM DETECTION

Benefits of OS Detection:
Target Profiling: Accurate OS detection provides insights into the target's technological landscape, aiding in precise target profiling.
Security Configuration: Knowledge of the OS helps security teams configure defenses, apply security patches, and implement OS-specific security measures.
Intrusion Detection: OS detection assists in identifying potentially rogue devices or unauthorized operating systems within a network.

OPERATING SYSTEM DETECTION

Nmap Options for OS Detection:

Default OS Detection:

```
nmap -O x.x.x.x
```

The -O option enables basic OS detection during a scan.

Aggressive OS Detection:

```
nmap -A x.x.x.x
```

The -A option includes aggressive scanning, combining service version detection, OS detection, and script scanning.

OPERATING SYSTEM DETECTION

Nmap Options for OS Detection:

Default OS Detection:

```
nmap -O x.x.x.x
```

The -O option enables basic OS detection during a scan.

Aggressive OS Detection:

```
nmap -A x.x.x.x
```

The -A option includes aggressive scanning, combining service version detection, OS detection, and script scanning.

OPERATING SYSTEM DETECTION

OS detection is enabled and controlled with the following options:

`-O (Enable OS detection)`

Enables OS detection. Alternatively, you can use -A to enable OS detection along with other things.

`--osscan-limit (Limit OS detection to promising targets)`

OS detection is far more effective if at least one open and one closed TCP port are found. Set this option and **Nmap will not even try OS detection against hosts that do not meet this criteria**. This can **save substantial time**, particularly on -Pn scans against many hosts. It only matters when OS detection is requested with -O or -A.

OPERATING SYSTEM DETECTION

OS detection is enabled and controlled with the following options:

```
--osscan-guess; --fuzzy (Guess OS detection results)
```

When Nmap is unable to detect a perfect OS match, it sometimes offers up near-matches as possibilities. The match has to be very close for Nmap to do this by default. Either of these (equivalent) options make Nmap guess more aggressively. **Nmap will still tell you when an imperfect match is printed and display its confidence level (percentage) for each guess**.

OPERATING SYSTEM DETECTION

OS detection is enabled and controlled with the following options:

`--max-os-tries (Set the maximum number of OS detection tries against a target)`

When **Nmap performs OS detection against a target and fails to find a perfect match, it usually repeats the attempt**. By default, **Nmap tries five times if conditions are favorable for OS fingerprint submission**, and **twice when conditions aren't so good**. Specifying a **lower --max-os-tries value (such as 1) speeds Nmap up**, though you miss out on retries which could potentially identify the OS. **Alternatively, a high value may be set to allow even more retries when conditions are favorable**. This is rarely done, **except to generate better fingerprints for submission and integration into the Nmap OS database**.

FIREWALL EVASION TECHNIQUES

Firewalls act as a critical defense mechanism, controlling and monitoring incoming and outgoing network traffic. While they play a crucial role in enhancing network security, **advanced scanning techniques often require strategies to bypass or evade firewall restrictions**. This section explores some techniques used to circumvent firewalls during network reconnaissance.

Understanding Firewall Evasion:
Firewall evasion involves tactics and methods to circumvent or trick firewalls into allowing unauthorized access to specific resources. While the primary goal of a firewall is to enforce security policies, certain scanning activities may need evasion techniques for successful execution.

FIREWALL EVASION TECHNIQUES

Common Firewall Evasion Techniques:
Fragmentation: Splitting packets into smaller fragments can help evade packet filtering rules. Firewalls may struggle to analyze and apply rules to fragmented packets.
Packet Size Manipulation: Modifying packet sizes to stay within the allowed limits set by the firewall can prevent detection of larger, potentially suspicious packets.
TCP Idle Scanning: Leveraging idle, seemingly unused machines as proxies to indirectly scan the target without directly interacting with the firewall.

FIREWALL EVASION TECHNIQUES

Common Firewall Evasion Techniques:
Using Non-Standard Ports: Scanning on non-standard ports (other than well-known ports) can help avoid detection by standard firewall rules.
HTTP Tunneling: Utilizing HTTP tunnels to pass scanning traffic through the firewall disguised as regular web traffic.
Stealthy Timing and Rate Control: Adjusting timing options to slower settings (-T0) can reduce the likelihood of triggering firewall alerts.

FIREWALL EVASION TECHNIQUES

Common Firewall Evasion Techniques:
Rate Control: Controlling the rate of scanning to avoid overwhelming the firewall and triggering alerts.
Encrypted Traffic: Encrypting scanning traffic using SSL/TLS protocols can help bypass packet inspection rules in some cases.
Scripting Engine for Firewall Evasion: Developing custom scripts using Nmap's scripting engine (NSE) to implement specific firewall evasion techniques. bash

FIREWALL EVASION TECHNIQUES

Common Firewall Evasion Techniques:
Randomizing Scan Order: Randomizing the order of scanned ports can help evade simple port-based firewall rules.
Dynamic IP Spoofing: Dynamic IP spoofing involves altering the source IP address during scanning to hide the true origin of the traffic.

FIREWALL EVASION TECHNIQUES

Firewall evasion techniques, including those in tools like Nmap, **remain relevant in certain scenarios**, but **their practicality and applicability depend on various factors**. It's essential to understand the context, ethical considerations, and legal implications when considering the use of firewall evasion techniques.

Practicality and Applicability:

Complex Firewalls:

Advanced firewalls with sophisticated intrusion detection and prevention systems may be more challenging to evade. They often employ deep packet inspection, anomaly detection, and behavior analysis.

Network Architecture:

In modern network architectures, especially in cloud environments, there might be additional layers of security, making evasion more challenging.

FIREWALL EVASION TECHNIQUES

Practicality and Applicability:
Advanced Threat Detection: Some firewalls utilize advanced threat detection mechanisms, including machine learning algorithms, making it harder for evasion techniques to go undetected.

FIREWALL EVASION TECHNIQUES

Use or Purpose of Firewall Evasion:
Penetration Testing: Ethical hackers and penetration testers may use firewall evasion techniques during authorized security assessments to identify potential weaknesses in security configurations.
Red Team Exercises: In red team exercises, security professionals simulate real-world attacks to assess an organization's security posture. Firewall evasion techniques can be employed to test the effectiveness of defensive measures.

FIREWALL EVASION TECHNIQUES

Use or Purpose of Firewall Evasion:

Security Awareness Training:

Demonstrating firewall evasion techniques in security training helps educate defenders about potential vulnerabilities and the importance of robust security configurations.

Understanding Weaknesses:

Identifying weaknesses in firewall configurations allows organizations to strengthen their security posture and implement effective countermeasures.

LAB EXERCISE 4

ALL PORTS OPEN?

Getting a result where all ports are open when scanning a target with Nmap may be indicative of several scenarios.

Here are some possible explanations:

Possible Reasons:
Firewall or Network Filtering: The target network may have a firewall or network filtering device configured to respond to all port scans with open status. This can be a deliberate attempt to mislead or confuse potential attackers.
Stateless Filtering Devices: Some stateless filtering devices or intrusion prevention systems (IPS) may respond to all port scan requests with open status as a default behavior. This is often done to prevent reconnaissance attempts by attackers.

ALL PORTS OPEN?

Possible Reasons:

Network Load Balancers or Proxies:

If the target is behind a network load balancer or a reverse proxy, these devices might be configured to respond to all port scans with open status to maintain service availability while distributing traffic across multiple servers.

Malicious Deception:

In some cases, the target system may intentionally deceive scanners by responding to all ports with open status to make it harder for attackers to identify potential vulnerabilities.

Port Redirection or Honeypot:

The target system may be configured with port redirection or may be a honeypot, designed to attract and analyze potentially malicious activities. In such cases, all ports may be reported as open to entice further interaction.

ALL PORTS OPEN?

Tips for Further Investigation:
Service Banner Analysis: Analyze the service banners returned by open ports to see if they correspond to expected services. Genuine services should have identifiable banners.
Additional Scanning Techniques: Use additional scanning techniques such as version detection (-sV), script scanning, or other Nmap options to gather more information about the open ports and services.

ALL PORTS OPEN?

Tips for Further Investigation:
Packet Capture: Capture and analyze network traffic during the scan using tools like Wireshark. This can help in understanding the nature of responses and potential network manipulations.
Manual Verification: Manually verify the open ports by attempting to connect to them using tools like telnet or netcat. This can help determine if the reported open status is accurate.

ALL PORTS OPEN?

Getting a result where all ports are open when scanning a target with Nmap may be indicative of several scenarios.

Here are some possible explanations:

Possible Reasons:
Firewall or Network Filtering: The target network may have a firewall or network filtering device configured to respond to all port scans with open status. This can be a deliberate attempt to mislead or confuse potential attackers.
Stateless Filtering Devices: Some stateless filtering devices or intrusion prevention systems (IPS) may respond to all port scan requests with open status as a default behavior. This is often done to prevent reconnaissance attempts by attackers.

TIMING AND PERFORMANCE OPTIMIZATION

Timing and performance optimization in Nmap are **crucial aspects to consider when conducting network scans**. Efficient timing settings help balance the need for thorough reconnaissance with the desire to complete scans in a reasonable time frame. Below are key considerations and strategies for optimizing timing and performance in Nmap:

Timing and Performance optimization

Timing Templates:

Nmap provides predefined timing templates (-T0 to -T5) that control the overall speed and aggressiveness of the scan.

```
nmap -T4 target
```

Recommendation: Choose a timing template that aligns with the goals of your scan. Aggressive templates (e.g., -T4 or -T5) may speed up the scan but increase the likelihood of detection.

TIMING AND PERFORMANCE OPTIMIZATION

Timing and Performance optimization

Individual Timing Options:

Fine-tune timing with individual options like --min-hostgroup, --max-hostgroup, --min-parallelism, and --max-parallelism.

```
nmap --min-parallelism 5 --max-parallelism 10 target
```

Recommendation: Adjust these options based on the network conditions and the level of aggressiveness desired.

TIMING AND PERFORMANCE OPTIMIZATION

Timing and Performance optimization

Round-Trip Time (RTT) Estimates:

Use Nmap's RTT estimates to adjust timing parameters dynamically. RTT is calculated during the initial host discovery phase.

```
nmap --min-rtt-timeout 300 --max-rtt-timeout 800 target
```

Recommendation: Customize RTT timeout values to suit the network characteristics and responsiveness.

TIMING AND PERFORMANCE OPTIMIZATION

Timing and Performance optimization

Rate Control:

Control the rate of scanning with the --max-rate option to avoid overwhelming networks or triggering intrusion detection systems.

```
nmap --max-rate 100 target
```

Recommendation: Adjust the rate based on the network's capacity and the desire to avoid detection.

TIMING AND PERFORMANCE OPTIMIZATION

Timing and Performance optimization

Parallel Scanning:

Increase parallelism to scan multiple hosts or ports simultaneously, speeding up the overall scan.

```
nmap --min-parallelism 10 --max-parallelism 20 target
```

Recommendation: Adjust parallelism based on available system resources and network conditions.

TIMING AND PERFORMANCE OPTIMIZATION

Timing and Performance optimization

Timing Intensity:

Use the --host-timeout option to control the time spent on each host. Adjusting the intensity can impact the thoroughness of the scan.

```
nmap --host-timeout 20m target
```

Recommendation: Customize timeout values based on the network size and desired thoroughness.

TIMING AND PERFORMANCE OPTIMIZATION

Timing and Performance optimization

Optimizing Scripting Engine:

If using Nmap's scripting engine (NSE), optimize scripts for performance by selecting only those relevant to the assessment.

```
nmap --script vuln target
```

Recommendation: Choose scripts judiciously to avoid unnecessary overhead.

TIMING AND PERFORMANCE OPTIMIZATION

Timing and Performance optimization

Host and Port Specification:

Specify hosts and ports explicitly to focus the scan on relevant targets, reducing unnecessary scanning overhead.

```
nmap target1 target2 -p 22,80,443
```

Tip: Tailor the scan to include only essential hosts and ports for faster results.

TIMING AND PERFORMANCE OPTIMIZATION

Timing and Performance optimization

Service Version Detection:

Use the -sV option selectively to perform service version detection only on essential ports, saving time and resources.

```
nmap -p 22,80 -sV target
```

Tip: Limit service version detection to ports of interest for faster scans.

TIMING AND PERFORMANCE OPTIMIZATION

Chaining masscan with nmap is a common practice for achieving faster and more efficient network scans. masscan is known for its ability to quickly identify live hosts and open ports, while nmap provides more detailed information about services, versions, and potential vulnerabilities.

LAB EXERCISE 5

JANGAN LUPA UNTUK LIKE, SHARE, KOMEN DAN SUBSEKERAIB CHANNEL MASTA GHIMAU!

INTRODUCTION TO NSE

One of the interesting features of Nmap is the Nmap Script Engine (NSE), which brings even **more flexibility and efficiency to it**. It enables you to **write your own scripts in Lua programming language**, and possibly share these scripts with other Nmap users out there.

There are four types of NSE scripts, namely:
Prerule scripts: Scripts that run before any of Nmap's scan operations, they are executed when Nmap hasn't gathered any information about a target yet.
Host scripts: Scripts executed after Nmap has performed normal operations such as host discovery, port scanning, version detection, and OS detection against a target host.

INTRODUCTION TO NSE

Note: Before we move any further, you should take a note of these key points:

- Do not execute scripts from third parties without critically looking through them or only if you trust the authors. This is because these scripts are not run in a sandbox and thus could unexpectedly or maliciously damage your system or invade your privacy.

- Secondly, many of these scripts may possibly run as either a prerule or postrule script. Considering this, it is recommended to use a prerule for purposes of consistency.

- Nmap uses the scripts/script.db database to figure out the available default scripts and categories.

INTRODUCTION TO NSE

NSE scripts are loaded using the **`--script`** flag, which also allows you to run your own scripts by providing categories, script file names, or the name of directories where your scripts are located.

The syntax for enabling scripts is as follows:

```
$ nmap -sC target #load default scripts
```

OR

```
$ nmap --script filename|category|directory|expression,...   target
```

INTRODUCTION TO NSE

You can view a description of a script with the --script-help option. Additionally, you can pass arguments to some scripts via the --script-args and --script-args-file options, the later is used to provide a filename rather than a command-line arg.

To perform a scan with most of the default scripts, use the -sC flag or alternatively use --script=default as shown.

```
$ nmap -sC scanme.nmap.org
```

OR

```
$ nmap --script=default scanme.nmap.org
```

INTRODUCTION TO NSE

To use a script for the appropriate purpose, you can, first of all, get a brief description of what it actually does, for instance, http-headers.

```
$ nmap --script-help http-headers scanme.nmap.org
```

Once you know what a script does, you can perform a scan using it. You can use one script or enter a comma-separated list of script names. The command below will enable you to view the HTTP headers configured on the webserver at the target host.

```
$ nmap --script http-headers scanme.nmap.org
```

INTRODUCTION TO NSE

You can also load scripts from one category or from a comma-separated list of categories. In this example, we are using all scripts in the default and broadcast category to carry out a scan on the host 192.168.56.1.

```
$ nmap --script default,broadcast 192.168.56.1
```

Using * Wildcard. This is useful when you want to select scripts with a given name pattern. For example to load all scripts with names starting with ssh, run the command below on the terminal:

```
$ nmap --script "ssh-*" 192.168.56.1
```

INTRODUCTION TO NSE

The next command looks a little complicated but it is easy to understand, it selects scripts in the default, or broadcast categories, leaving out those with names starting with ssh-:

```
$ nmap --script "(default or broadcast) and not ssh-*" 192.168.56.10
```

Importantly, it is possible to combine categories, script names, a directory containing your custom scripts, or a boolean expression to load scripts, like this:

```
$ nmap --script broadcast,vuln,ssh-auth-methods,/path/to/custom/scripts
192.168.56.10
```

INTRODUCTION TO NSE

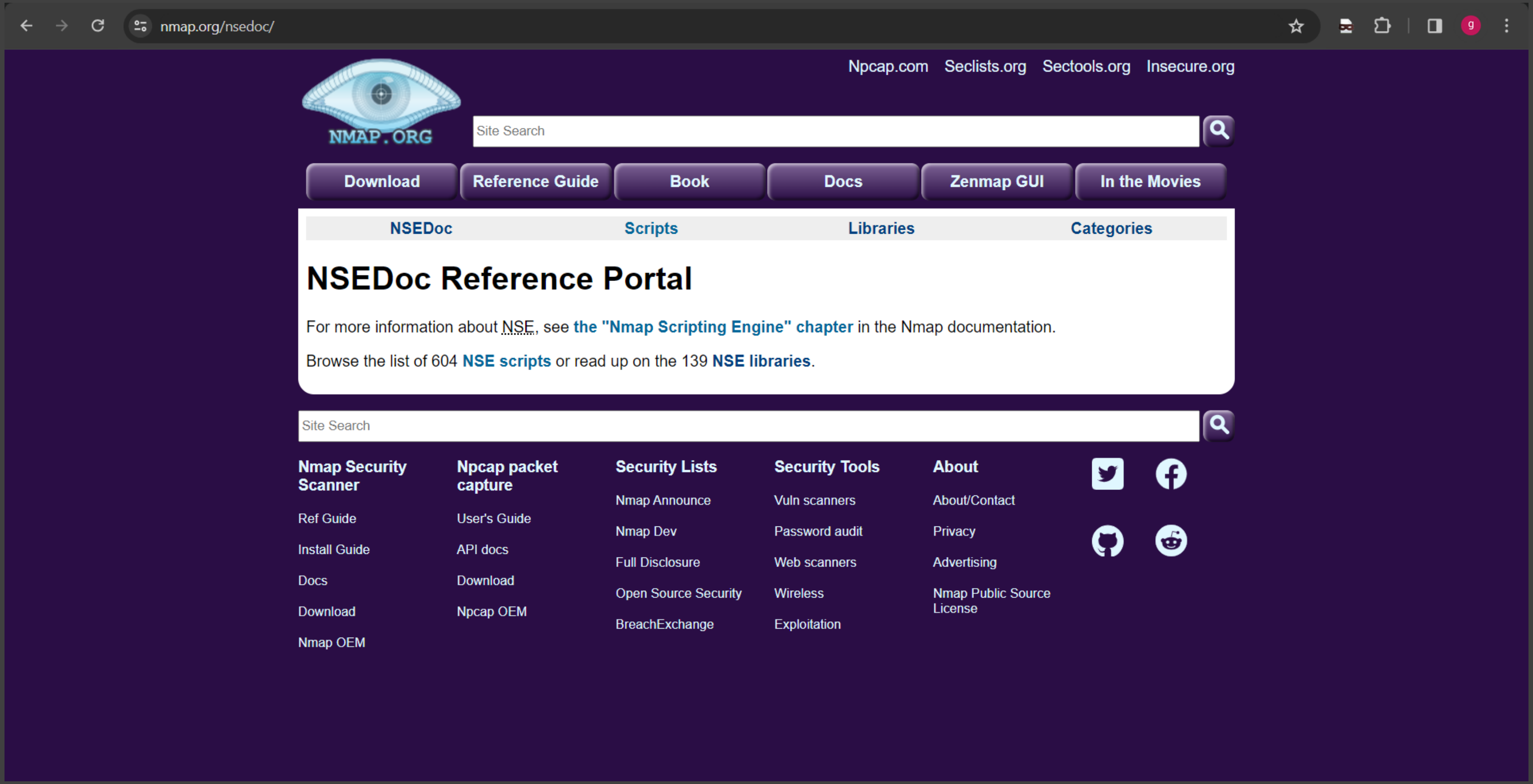
Passing Arguments to NSE Scripts. Below is an example showing how to pass arguments to scripts with the –script-args option:

```
$ nmap --script mysql-audit --script-args "mysql-audit.username='root', \
mysql-audit.password='password_here', mysql-audit.filename='nselib/data/mysql-
cis.audit'"
```

To pass a port number, use the -p nmap option:

```
$ nmap -p 3306 --script mysql-audit --script-args "mysql-audit.username='root', \
mysql-audit.password='password_here' , mysql-audit.filename='nselib/data/mysql-
cis.audit'"
```

INTRODUCTION TO NSE



nmap.org/nsedoc/

Npcap.com Seclists.org Sectools.org Insecure.org

NMAP.ORG

Site Search

Download | Reference Guide | Book | Docs | Zenmap GUI | In the Movies

NSEDoc | Scripts | Libraries | Categories

NSEDoc Reference Portal

For more information about NSE, see **the "Nmap Scripting Engine" chapter** in the Nmap documentation.

Browse the list of 604 **NSE scripts** or read up on the 139 **NSE libraries**.

Site Search

Nmap Security Scanner

- Ref Guide
- Install Guide
- Docs
- Download
- Nmap OEM

Npcap packet capture

- User's Guide
- API docs
- Download
- Npcap OEM

Security Lists

- Nmap Announce
- Nmap Dev
- Full Disclosure
- Open Source Security
- BreachExchange

Security Tools

- Vuln scanners
- Password audit
- Web scanners
- Wireless
- Exploitation

About

- About/Contact
- Privacy
- Advertising
- Nmap Public Source License

https://nmap.org/nsedoc/

WRITING CUSTOM NSE SCRIPT

Nmap's NSE provides a powerful framework for creating custom scripts to extend its functionality. Developing your own NSE script allows you to tailor Nmap scans to specific requirements, automate tasks, or perform custom checks on target systems. Here's a step-by-step guide on creating your own NSE script.

Step 1: Understand Lua Programming:

NSE scripts are written in the **Lua programming language**. Familiarize yourself with Lua syntax, functions, and features. Lua is known for its simplicity and ease of integration, making it suitable for scripting tasks.

Step 2: Set Up Your Development Environment:

Ensure you have a text editor or integrated development environment (IDE) that supports Lua syntax highlighting. Popular choices include Visual Studio Code, Sublime Text, or Atom. Use a version control system (e.g., Git) to track changes in your script.

WRITING CUSTOM NSE SCRIPT

Step 3: Define Script Arguments:

Consider the **parameters your script might need**. **Define script arguments using the description, categories, and args fields in the script**. This allows users to customize the script's behavior when running it.

```
description = "My Custom NSE Script"
categories = {"safe", "discovery"}
args = {
  {name = "target", description = "Target host", required = true},
  {name = "customOption", description = "Custom option", default = "default_value"}
}
```

WRITING CUSTOM NSE SCRIPT

Step 4: Write Script Logic:

Implement the core logic of your script. **Use NSE functions such as hostrule for host selection, portrule for port filtering, and action for executing actions**. Leverage **existing NSE libraries for common tasks**.

```
hostrule = function(host)
  -- Define host selection logic
  return true
end
```

WRITING CUSTOM NSE SCRIPT

Step 4: Write Script Logic:

Implement the core logic of your script. **Use NSE functions such as hostrule for host selection, portrule for port filtering, and action for executing actions**. Leverage **existing NSE libraries for common tasks**.

```
portrule = function(host, port)
  -- Define port filtering logic
  return port.number == 80
end
```

WRITING CUSTOM NSE SCRIPT

Step 4: Write Script Logic:

Implement the core logic of your script. **Use NSE functions such as hostrule for host selection, portrule for port filtering, and action for executing actions**. Leverage **existing NSE libraries for common tasks**.

```
action = function(host, port)
  -- Execute actions on the selected host and port
  print("Custom action on", host.ip, "port", port.number)
end
```

WRITING CUSTOM NSE SCRIPT

Step 5: Test Your Script:

Test your script on different target scenarios. Ensure it behaves as expected and handles different inputs gracefully.

```
nmap -p 80 --script my_custom_script.nse <target>
```

WRITING CUSTOM NSE SCRIPT

Tips for Creating NSE Scripts:
Follow Nmap Standards: Adhere to **Nmap script standards and guidelines to ensure compatibility and consistency**.
Leverage NSE Libraries: **Use existing NSE libraries for common tasks** (e.g., stdnse for standard functions, http for HTTP-related tasks).
Document Your Script: Provide **clear and concise documentation within the script**, explaining its purpose, usage, and any specific considerations.

WRITING CUSTOM NSE SCRIPT

Tips for Creating NSE Scripts:
Handle Errors Gracefully: Implement **error-checking mechanisms** and handle exceptions gracefully to ensure robust script behavior.
Contribute to the Nmap Community: Consider **contributing your script to the official Nmap Scripting Engine repository to benefit the wider community**.

By following these steps and tips, you can create effective and useful NSE scripts tailored to your specific needs or scenarios. Always test and document your scripts thoroughly for reliability and ease of use.

DEBUGGING CUSTOM NSE SCRIPT TIPS

Debugging NSE (Nmap Scripting Engine) scripts can be a crucial step in ensuring their functionality and identifying any issues. Here are some techniques to debug NSE scripts effectively:

Print Statements:

Use **print statements to output messages and variable values during script execution**. This helps in understanding the flow of your script and identifying potential issues.

```
action = function(host, port)
  print("Executing custom action on", host.ip, "port", port.number)
  -- Your script logic here
end
```

DEBUGGING CUSTOM NSE SCRIPT TIPS

Logging to a File:

Redirect print statements to a log file using io.open. This allows you to review the log file after script execution.

```
local logFile = io.open("/path/to/debug_log.txt", "a")

action = function(host, port)
  logFile:write("Executing custom action on", host.ip, "port", port.number, "\n")
  -- Your script logic here
end
```

DEBUGGING CUSTOM NSE SCRIPT TIPS

Logging to a File:

Redirect print statements to a log file using io.open. This allows you to review the log file after script execution.

```
local logFile = io.open("/path/to/debug_log.txt", "a")

action = function(host, port)
  logFile:write("Executing custom action on", host.ip, "port", port.number, "\n")
  -- Your script logic here
end
logFile:close()
```

DEBUGGING CUSTOM NSE SCRIPT TIPS

Error Handling:

Implement **error-checking mechanisms using assert or other error-handling functions**. This helps identify and handle issues gracefully.

```
action = function(host, port)
  local result, error_message = pcall(function()
    -- Your script logic here
  end)

  if not result then
    print("Error:", error_message)
  end
end
```

DEBUGGING CUSTOM NSE SCRIPT TIPS

Interactive Testing:

Use the Nmap **interactive mode (--interactive or -iI) to test your script interactively**. This allows you to observe the behavior and output at different stages.

```
nmap --script my_script.nse --interactive
```

DEBUGGING CUSTOM NSE SCRIPT TIPS

Reviewing Nmap's Debugging Output:

Examine **the debugging output generated by Nmap (-d option)** to understand how scripts are being loaded and executed. This can provide insights into any issues or unexpected behavior.

```
nmap -p 80 --script my_script.nse -d <target>
```

DEBUGGING CUSTOM NSE SCRIPT TIPS

Check Return Values:

Check return values of NSE functions and library functions. **Incorrect return values might indicate issues**.

```
hostrule = function(host)
  if host.ip == "192.168.1.1" then
    return true
  else
    return false
  end
end
```

DEBUGGING CUSTOM NSE SCRIPT TIPS

Using the --script-trace Option:

The **--script-trace option in Nmap provides detailed information about the execution of NSE scripts**. It shows each script invocation, including input arguments and output.

```
nmap --script my_script.nse --script-trace <target>
```

Choose the debugging method that best fits your needs and the complexity of your NSE script. Combining multiple approaches may provide a more comprehensive understanding of script behavior during development and testing.

NSE SCRIPT SAMPLE

```
-- Nmap: NSE script to detect HTTP title on port 80
-- Script Name: http-title.nse
-- Categories: discovery, safe
-- Author: masta ghimau
-- License: Same as Nmap--See https://nmap.org/book/man-legal.html

description = [[
  Attempts to retrieve the HTTP title from the target's port 80.
]]

--@output
-- 80/tcp   open  http   syn-ack   Example Website
```

NSE SCRIPT SAMPLE - ANATOMY

```
-- The prerule function decides whether to skip or execute the script's action.
prerule = function(host)
  -- Check if port 80 is open on the target.
  local http_port = host:get_port(80, "tcp")
  return http_port and http_port.state == "open"
end
```

NSE SCRIPT SAMPLE

```lua
-- The action function is the main entry point for NSE scripts.
action = function(host, port)
  -- Perform a simple HTTP GET request to retrieve the title.
  local response = http.get(host, port, "/")

  -- Check if the request was successful (200 OK).
  if response.status == 200 then
    -- Extract the title from the HTML response.
    local title = response.body:match("<title>(.-)</title>")
```

NSE SCRIPT SAMPLE

```
-- Print the results in the required Nmap format.
    if title then
      return string.format("%-6s %-4s %-8s %-6s %s", port.number, port.protocol,
port.service.name, "open", title)
    else
      return string.format("%-6s %-4s %-8s %-6s %s", port.number, port.protocol,
port.service.name, "open", "No Title")
    end
  else
    -- If the request was not successful, print an error.
    return string.format("%-6s %-4s %-8s %-6s %s", port.number, port.protocol,
port.service.name, "closed", "HTTP Error")
  end
end
```

NSE SCRIPT SAMPLE

Explanation:
Head Section: The head section contains meta-information about the script, including its name, categories, author, license, and description.
Output Format: The @output section specifies the expected output format when the script is run.
Rule Section (prerule): The prerule function is used to decide whether to skip or execute the script's action based on whether port 80 is open on the target.
Action Section: The action function is the main entry point for NSE scripts. It performs an HTTP GET request to retrieve the title of the target's port 80.

LAB EXERCISE 6

SESSION ENDS TQ

JANGAN LUPA UNTUK LIKE, SHARE, KOMEN DAN SUBSEKERAIB CHANNEL MASTA GHIMAU!