Report on TryHackMe Nmap Room Tasks

Author: TOM SHINJO THOMAS

Introduction

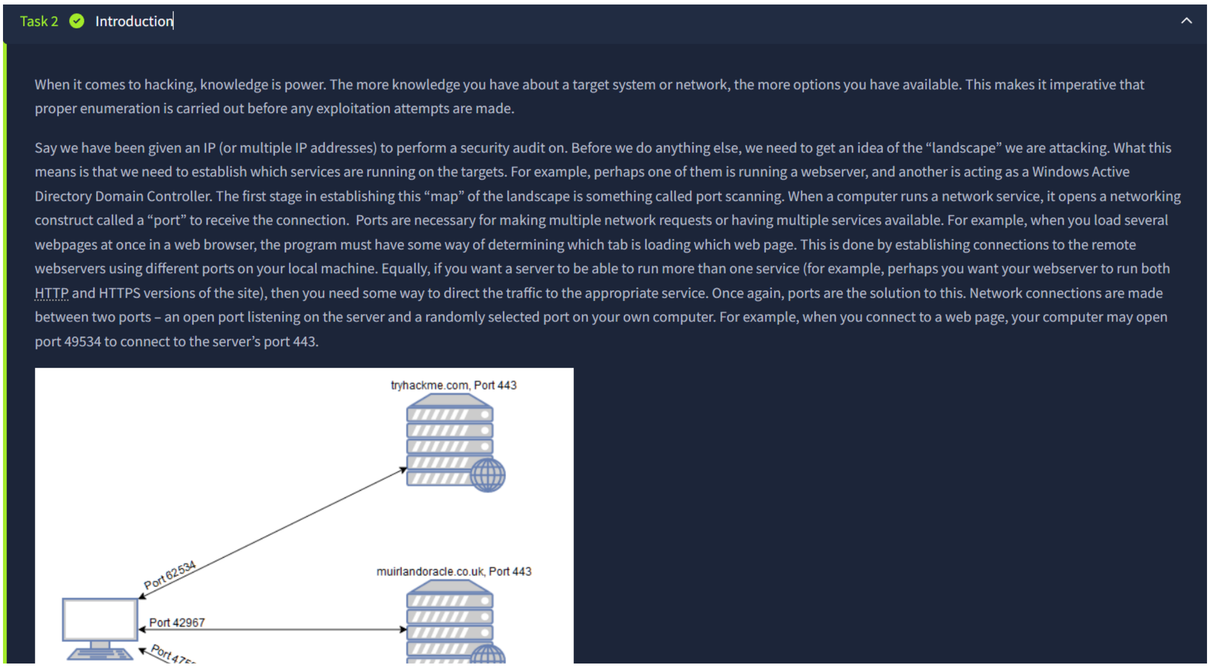
This report outlines the progress and key concepts covered in the TryHackMe "Nmap" room. The tasks focus on understanding and utilizing Nmap (Network Mapper), a powerful open-source tool for network exploration and security auditing. The screenshots provided cover the entire room, from fundamental Nmap concepts and scan types to advanced topics like the Nmap Scripting Engine, firewall evasion, and a final practical challenge.

Task Analysis

Task 1 & 2: Introduction to Nmap

The initial tasks introduce the importance of enumeration and establish Nmap as a primary tool for this purpose. The room explains that before any exploitation attempts, it's crucial to build a "map" of the target network landscape. This process begins with port scanning to identify which services are running. The concept of ports is explained as a mechanism for handling multiple network requests on a single server.

The screenshots also show the use of the man Nmap command, which brings up the Nmap Reference Guide. This guide provides a comprehensive description of Nmap, its synopsis, and how it works. It explains that Nmap uses raw IP packets to discover hosts, services, operating systems, and potential firewall presence. The concept of the "interesting ports table" is introduced, which lists the port number, protocol, service name, and state (open, closed, filtered, or unfiltered).



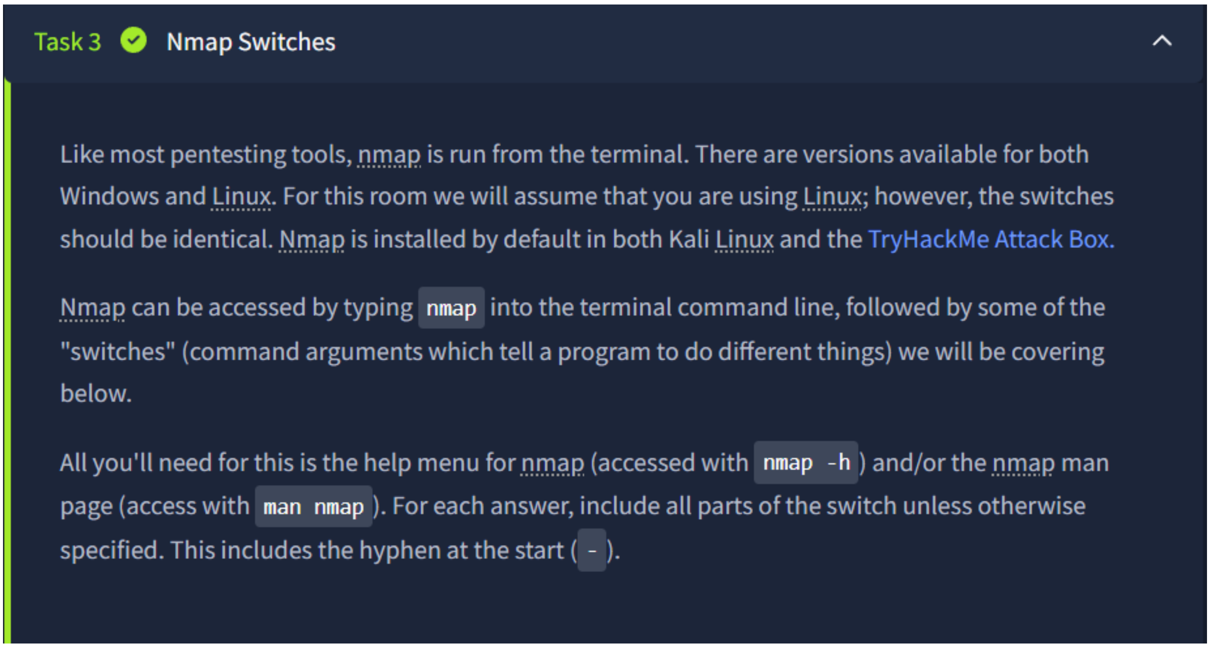
Task 2 Introduction

When it comes to hacking, knowledge is power. The more knowledge you have about a target system or network, the more options you have available. This makes it imperative that proper enumeration is carried out before any exploitation attempts are made.

Say we have been given an IP (or multiple IP addresses) to perform a security audit on. Before we do anything else, we need to get an idea of the "landscape" we are attacking. What this means is that we need to establish which services are running on the targets. For example, perhaps one of them is running a webserver, and another is acting as a Windows Active Directory Domain Controller. The first stage in establishing this "map" of the landscape is something called port scanning. When a computer runs a network service, it opens a networking construct called a "port" to receive the connection. Ports are necessary for making multiple network requests or having multiple services available. For example, when you load several webpages at once in a web browser, the program must have some way of determining which tab is loading which web page. This is done by establishing connections to the remote webservers using different ports on your local machine. Equally, if you want a server to be able to run more than one service (for example, perhaps you want your webserver to run both HTTP and HTTPS versions of the site), then you need some way to direct the traffic to the appropriate service. Once again, ports are the solution to this. Network connections are made between two ports – an open port listening on the server and a randomly selected port on your own computer. For example, when you connect to a web page, your computer may open port 49534 to connect to the server's port 443.

Task 3: Nmap Switches

This task focuses on the command-line arguments, or "switches," used to control Nmap's behaviour. It emphasizes that Nmap is run from the terminal and that a variety of switches can be appended to the nmap command to perform different types of scans and actions. The task requires using the help menu (nmap -h) or the man page (man nmap) to find the correct switches for various operations.



Task 3 Nmap Switches

Like most pentesting tools, nmap is run from the terminal. There are versions available for both Windows and Linux. For this room we will assume that you are using Linux; however, the switches should be identical. Nmap is installed by default in both Kali Linux and the TryHackMe Attack Box.

Nmap can be accessed by typing `nmap` into the terminal command line, followed by some of the "switches" (command arguments which tell a program to do different things) we will be covering below.

All you'll need for this is the help menu for nmap (accessed with `nmap -h`) and/or the nmap man page (access with `man nmap`). For each answer, include all parts of the switch unless otherwise specified. This includes the hyphen at the start (`-`).

Task 5: TCP Connect Scans (-sT)

This section delves into one of the most fundamental scan types: the TCP Connect scan. It explains the underlying mechanism, which is the **TCP three-way handshake**:

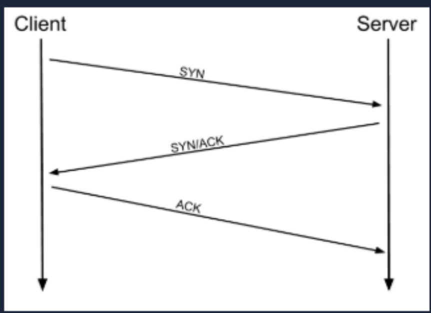
1. **SYN:** The attacking machine sends a TCP packet with the SYN (synchronize) flag set.
2. **SYN/ACK:** If the port is open, the target server responds with a TCP packet containing both the SYN and ACK (acknowledgment) flags.
3. **ACK:** The attacker completes the handshake by sending an ACK packet.

By completing this handshake for each port, Nmap can reliably determine if the port is open.

Task 5 Scan Types TCP Connect Scans

To understand TCP Connect scans (`-sT`), it's important that you're comfortable with the TCP *three-way handshake*. If this term is new to you then completing Introductory Networking before continuing would be advisable.

As a brief recap, the three-way handshake consists of three stages. First the connecting terminal (our attacking machine, in this instance) sends a TCP request to the target server with the SYN flag set. The server then acknowledges this packet with a TCP response containing the SYN flag, as well as the ACK flag. Finally, our terminal completes the handshake by sending a TCP request with the ACK flag set.



No.	Time	Source	Destination	Protocol	Length	Info
21	2.009477639	192.168.1.142	192.168.1.141	TCP	74	60516 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2310196 TSecr=0 WS=128
22	2.009847598	192.168.1.141	192.168.1.142	TCP	66	80 → 60516 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
23	2.009886244	192.168.1.142	192.168.1.141	TCP	54	60516 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0

This is one of the fundamental principles of TCP/IP networking, but how does it relate to Nmap?

Well, as the name suggests, a TCP Connect scan works by performing the three-way handshake with each target port in turn. In other words, Nmap tries to connect to each specified TCP port, and determines whether the service is open by the response it receives.

Task 6: SYN Scans (-sS)

Often referred to as "half-open" or "stealth" scans, SYN scans are a more discreet alternative to TCP Connect scans. The process is as follows:

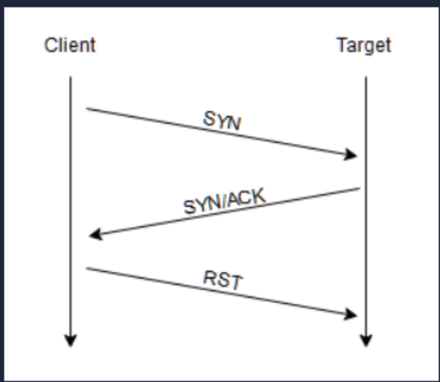
1. **SYN:** The attacker sends a SYN packet, just like in a connect scan.
2. **SYN/ACK:** An open port will respond with a SYN/ACK packet.
3. **RST:** Instead of completing the handshake with an ACK, the attacker sends a RST (reset) packet.

Because the full connection is never established, this method is less likely to be logged by the target system, making it stealthier.

Task 6 Scan Types SYN Scans

As with TCP scans, SYN scans (`-sS`) are used to scan the TCP port-range of a target or targets; however, the two scan types work slightly differently. SYN scans are sometimes referred to as "*Half-open*" scans, or "*Stealth*" scans.

Where TCP scans perform a full three-way handshake with the target, SYN scans sends back a RST TCP packet after receiving a SYN/ACK from the server (this prevents the server from repeatedly trying to make the request). In other words, the sequence for scanning an **open** port looks like this:



No.	Time	Source	Destination	Protocol	Length	Info
39	8.389443540	192.168.1.142	192.168.1.238	TCP	58	53425 → 80 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
40	8.389900067	192.168.1.238	192.168.1.142	TCP	60	80 → 53425 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
41	8.389992786	192.168.1.142	192.168.1.238	TCP	54	53425 → 80 [RST] Seq=1 Win=0 Len=0

This has a variety of advantages for us as hackers:

- It can be used to bypass older Intrusion Detection systems as they are looking out for a full three way handshake. This is often no longer the case with modern IDS solutions; it is for this reason that SYN scans are still frequently referred to as "stealth" scans.

Task 7: UDP Scans (-sU)

This task explains the challenges of scanning for UDP ports. Unlike TCP, UDP is a stateless protocol. Nmap's UDP scan works as follows:

- **Open Port:** If a UDP packet is sent to an open port, there should be **no response**. The port is then marked as open|filtered.
- **Closed Port:** If the port is closed, the target should respond with an **ICMP "port unreachable"** message.

UDP scans are significantly slower than TCP scans. The task recommends using the --top-ports <number> switch to save time.

Task 7 Scan Types **UDP Scans**

Unlike TCP, UDP connections are *stateless*. This means that, rather than initiating a connection with a back-and-forth "handshake", UDP connections rely on sending packets to a target port and essentially hoping that they make it. This makes UDP superb for connections which rely on speed over quality (e.g. video sharing), but the lack of acknowledgement makes UDP significantly more difficult (and much slower) to scan. The switch for an Nmap UDP scan is (-sU)

When a packet is sent to an open UDP port, there should be no response. When this happens, Nmap refers to the port as being open|filtered . In other words, it suspects that the port is open, but it could be firewalled. If it gets a UDP response (which is very unusual), then the port is marked as *open*. More commonly there is no response, in which case the request is sent a second time as a double-check. If there is still no response then the port is marked *open|filtered* and Nmap moves on.

When a packet is sent to a *closed* UDP port, the target should respond with an ICMP (ping) packet containing a message that the port is unreachable. This clearly identifies closed ports, which Nmap marks as such and moves on.

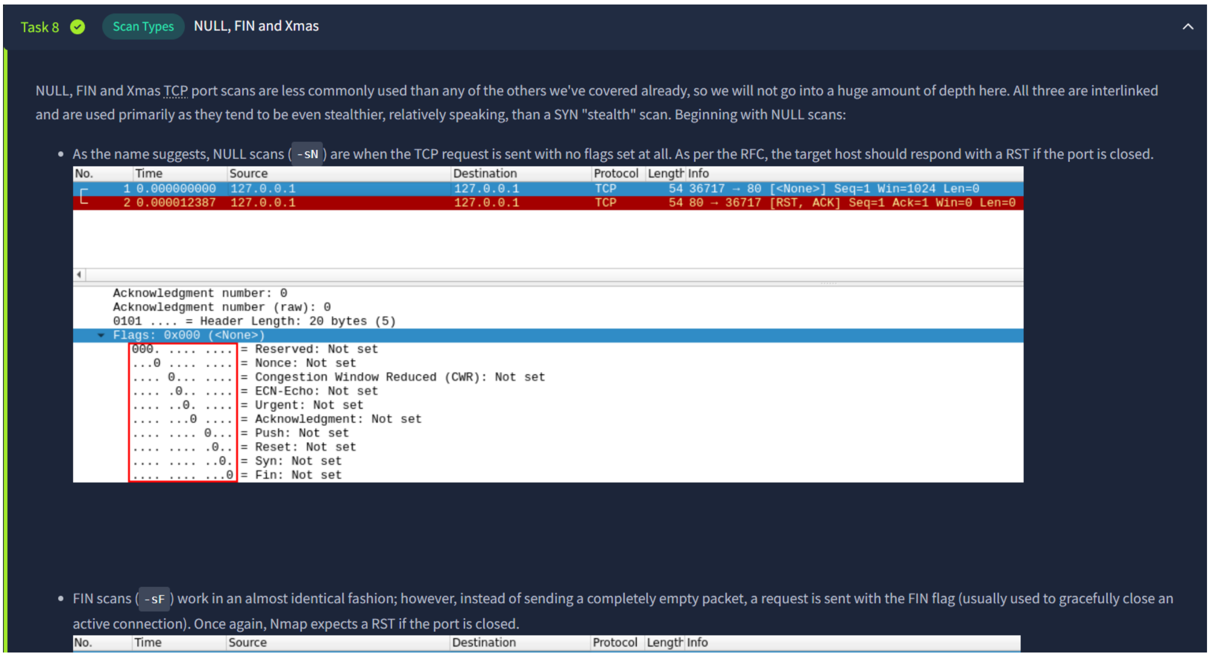
Due to this difficulty in identifying whether a UDP port is actually open, UDP scans tend to be incredibly slow in comparison to the various TCP scans (in the region of 20 minutes to scan the first 1000 ports, with a good connection). For this reason it's usually good practice to run an Nmap scan with --top-ports <number> enabled. For example, scanning with nmap -sU --top-ports 20 <target> . Will scan the top 20 most commonly used UDP ports, resulting in a much more acceptable scan time.

Task 8: NULL, FIN, and Xmas Scans (-sN, -sF, -sX)

These are even stealthier scan types that rely on RFC-compliant TCP stacks:

- **NULL Scan (-sN):** Sends a TCP packet with no flags set.
- **FIN Scan (-sF):** Sends a TCP packet with only the FIN flag set.
- **Xmas Scan (-sX):** Sends a packet with the FIN, PSH, and URG flags set.

For all three scans, a closed port should respond with a RST packet, while an open port should not respond at all.


Task 8 Scan Types NULL, FIN and Xmas

NULL, FIN and Xmas TCP port scans are less commonly used than any of the others we've covered already, so we will not go into a huge amount of depth here. All three are interlinked and are used primarily as they tend to be even stealthier, relatively speaking, than a SYN "stealth" scan. Beginning with NULL scans:

- As the name suggests, NULL scans (`-sN`) are when the TCP request is sent with no flags set at all. As per the RFC, the target host should respond with a RST if the port is closed.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	54	36717 → 80 [<None>] Seq=1 Win=1024 Len=0
2	0.000012387	127.0.0.1	127.0.0.1	TCP	54	80 → 36717 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

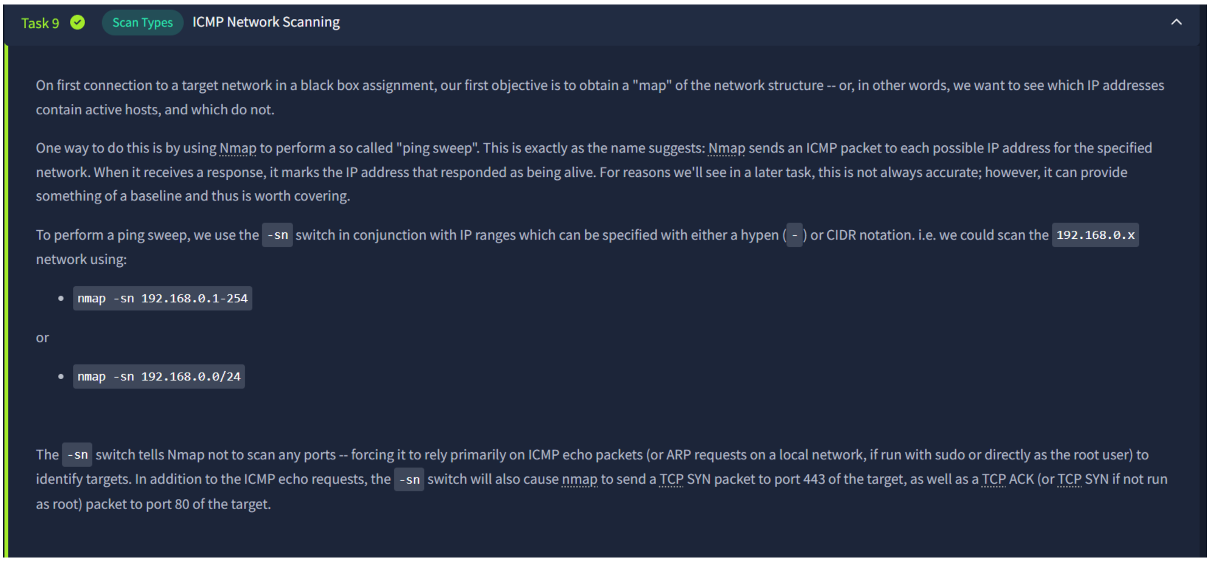
```
Acknowledgment number: 0
Acknowledgment number (raw): 0
0101 .... = Header Length: 20 bytes (5)
Flags: 0x000 (<None>)
   000. .... .... = Reserved: Not set
   ...0 .... .... = Nonce: Not set
   .... 0... .... = Congestion Window Reduced (CWR): Not set
   .... .0.. .... = ECN-Echo: Not set
   .... ..0. .... = Urgent: Not set
   .... ...0 .... = Acknowledgment: Not set
   .... .... 0... = Push: Not set
   .... .... .0.. = Reset: Not set
   .... .... ..0. = Syn: Not set
   .... .... ...0 = Fin: Not set
```

- FIN scans (`-sF`) work in an almost identical fashion; however, instead of sending a completely empty packet, a request is sent with the FIN flag (usually used to gracefully close an active connection). Once again, Nmap expects a RST if the port is closed.

No.	Time	Source	Destination	Protocol	Length	Info

Task 9: ICMP Network Scanning (-sn)

This section covers the "ping sweep," used to discover active hosts without port scanning them. The -sn switch tells Nmap to send an ICMP echo request to each IP in a range. Hosts that respond are marked as "alive."


Task 9 Scan Types ICMP Network Scanning

On first connection to a target network in a black box assignment, our first objective is to obtain a "map" of the network structure -- or, in other words, we want to see which IP addresses contain active hosts, and which do not.

One way to do this is by using Nmap to perform a so called "ping sweep". This is exactly as the name suggests: Nmap sends an ICMP packet to each possible IP address for the specified network. When it receives a response, it marks the IP address that responded as being alive. For reasons we'll see in a later task, this is not always accurate; however, it can provide something of a baseline and thus is worth covering.

To perform a ping sweep, we use the `-sn` switch in conjunction with IP ranges which can be specified with either a hypen (`-`) or CIDR notation. i.e. we could scan the `192.168.0.x` network using:

- `nmap -sn 192.168.0.1-254`

or

- `nmap -sn 192.168.0.0/24`

The `-sn` switch tells Nmap not to scan any ports -- forcing it to rely primarily on ICMP echo packets (or ARP requests on a local network, if run with sudo or directly as the root user) to identify targets. In addition to the ICMP echo requests, the `-sn` switch will also cause nmap to send a TCP SYN packet to port 443 of the target, as well as a TCP ACK (or TCP SYN if not run as root) packet to port 80 of the target.

Tasks 10, 11, & 12: Nmap Scripting Engine (NSE)

These tasks introduce the Nmap Scripting Engine (NSE), which uses Lua scripts to extend Nmap's functionality. The script categories include safe, intrusive, vuln, exploit, auth, brute, and discovery. The tasks cover how to run scripts by category (--script=vuln) or by name (--script=<script-name>). Task 12 explains how to find these scripts, which are located in the /usr/share/nmap/scripts/ directory, and how to search for them using the script.db file.

Task 10 NSE Scripts Overview

The **N**map **S**cripting **E**ngine (NSE) is an incredibly powerful addition to Nmap, extending its functionality quite considerably. NSE Scripts are written in the *Lua* programming language, and can be used to do a variety of things: from scanning for vulnerabilities, to automating exploits for them. The NSE is particularly useful for reconnaisance, however, it is well worth bearing in mind how extensive the script library is.

There are many categories available. Some useful categories include:

- `safe` :- Won't affect the target
- `intrusive` :- Not safe: likely to affect the target
- `vuln` :- Scan for vulnerabilities
- `exploit` :- Attempt to exploit a vulnerability
- `auth` :- Attempt to bypass authentication for running services (e.g. Log into an FTP server anonymously)
- `brute` :- Attempt to bruteforce credentials for running services
- `discovery` :- Attempt to query running services for further information about the network (e.g. query an SNMP server).

A more exhaustive list can be found here.

In the next task we'll look at how to interact with the NSE and make use of the scripts in these categories.

Task 11 NSE Scripts Working with the NSE

In Task 3 we looked very briefly at the `--script` switch for activating NSE scripts from the `vuln` category using `--script=vuln`. It should come as no surprise that the other categories work in exactly the same way. If the command `--script=safe` is run, then any applicable safe scripts will be run against the target (Note: only scripts which target an active service will be activated).

To run a specific script, we would use `--script=<script-name>` , e.g. `--script=http-fileupload-exploiter`.

Multiple scripts can be run simultaneously in this fashion by separating them by a comma. For example: `--script=smb-enum-users,smb-enum-shares`.

Some scripts require arguments (for example, credentials, if they're exploiting an authenticated vulnerability). These can be given with the `--script-args` Nmap switch. An example of this would be with the `http-put` script (used to upload files using the PUT method). This takes two arguments: the URL to upload the file to, and the file's location on disk. For example:

```
nmap -p 80 --script http-put --script-args http-put.url='/dav/shell.php',http-put.file='./shell.php'
```

Note that the arguments are separated by commas, and connected to the corresponding script with periods (i.e. `<script-name>.<argument>`).

A full list of scripts and their corresponding arguments (along with example use cases) can be found here.

Nmap scripts come with built-in help menus, which can be accessed using `nmap --script-help <script-name>`. This tends not to be as extensive as in the link given above, however, it can still be useful when working locally.

Task 12 NSE Scripts Searching for Scripts

Ok, so we know how to use the scripts in Nmap, but we don't yet know how to *find* these scripts.

We have two options for this, which should ideally be used in conjunction with each other. The first is the page on the Nmap website (mentioned in the previous task) which contains a list of all official scripts. The second is the local storage on your attacking machine. Nmap stores its scripts on Linux at `/usr/share/nmap/scripts`. All of the NSE scripts are stored in this directory by default -- this is where Nmap looks for scripts when you specify them.

There are two ways to search for installed scripts. One is by using the `/usr/share/nmap/scripts/script.db` file. Despite the extension, this isn't actually a database so much as a formatted text file containing filenames and categories for each available script.

```
muri@augury:/usr/share/nmap/scripts$ file script.db
script.db: ASCII text
muri@augury:/usr/share/nmap/scripts$ head script.db
Entry { filename = "acarsd-info.nse", categories = { "discovery", "safe", } }
Entry { filename = "address-info.nse", categories = { "default", "safe", } }
Entry { filename = "afp-brute.nse", categories = { "brute", "intrusive", } }
Entry { filename = "afp-ls.nse", categories = { "discovery", "safe", } }
Entry { filename = "afp-path-vuln.nse", categories = { "exploit", "intrusive", "vuln", } }
Entry { filename = "afp-serverinfo.nse", categories = { "default", "discovery", "safe", } }
Entry { filename = "afp-showmount.nse", categories = { "discovery", "safe", } }
Entry { filename = "ajp-auth.nse", categories = { "auth", "default", "safe", } }
Entry { filename = "ajp-brute.nse", categories = { "brute", "intrusive", } }
Entry { filename = "ajp-headers.nse", categories = { "discovery", "safe", } }
```

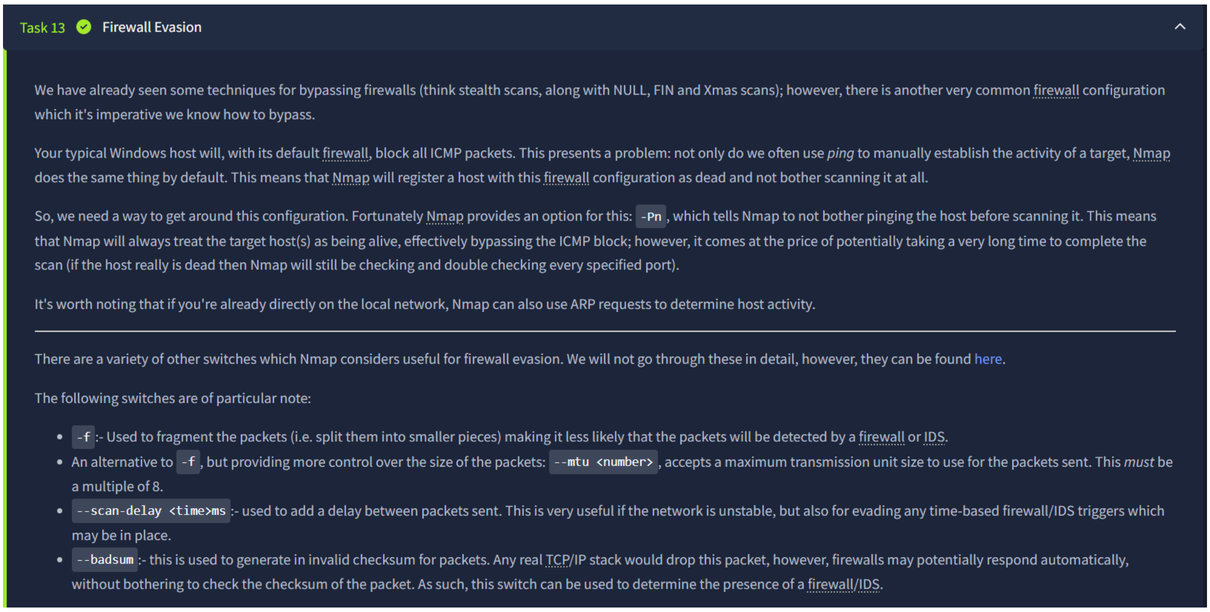
Nmap uses this file to keep track of (and utilise) scripts for the scripting engine; however, we can also *grep* through it to look for scripts. For example: `grep "ftp" /usr/share/nmap/scripts/script.db`.

```
muri@augury:/usr/share/nmap/scripts$ grep "ftp" /usr/share/nmap/scripts/script.db
Entry { filename = "ftp-anon.nse", categories = { "auth", "default", "safe", } }
Entry { filename = "ftp-bounce.nse", categories = { "default", "safe", } }
Entry { filename = "ftp-brute.nse", categories = { "brute", "intrusive", } }
```

Task 13: Firewall Evasion

This task focuses on techniques for bypassing firewalls and Intrusion Detection Systems (IDS). Key switches covered include:

- **-Pn**: Skips the host discovery (ping) phase, treating all targets as online. This is useful if the target blocks ICMP requests.
- **-f**: Fragments packets, making them harder for firewalls to detect.
- **--scan-delay <time>**: Adds a delay between probes to avoid triggering time-based firewall rules.
- **--badsum**: Sends packets with an invalid TCP/UDP checksum. While regular hosts would drop these, some firewalls might not check the checksum and respond, revealing their presence.



Task 13 Firewall Evasion

We have already seen some techniques for bypassing firewalls (think stealth scans, along with NULL, FIN and Xmas scans); however, there is another very common firewall configuration which it's imperative we know how to bypass.

Your typical Windows host will, with its default firewall, block all ICMP packets. This presents a problem: not only do we often use *ping* to manually establish the activity of a target, Nmap does the same thing by default. This means that Nmap will register a host with this firewall configuration as dead and not bother scanning it at all.

So, we need a way to get around this configuration. Fortunately Nmap provides an option for this: `-Pn`, which tells Nmap to not bother pinging the host before scanning it. This means that Nmap will always treat the target host(s) as being alive, effectively bypassing the ICMP block; however, it comes at the price of potentially taking a very long time to complete the scan (if the host really is dead then Nmap will still be checking and double checking every specified port).

It's worth noting that if you're already directly on the local network, Nmap can also use ARP requests to determine host activity.

There are a variety of other switches which Nmap considers useful for firewall evasion. We will not go through these in detail, however, they can be found here.

The following switches are of particular note:

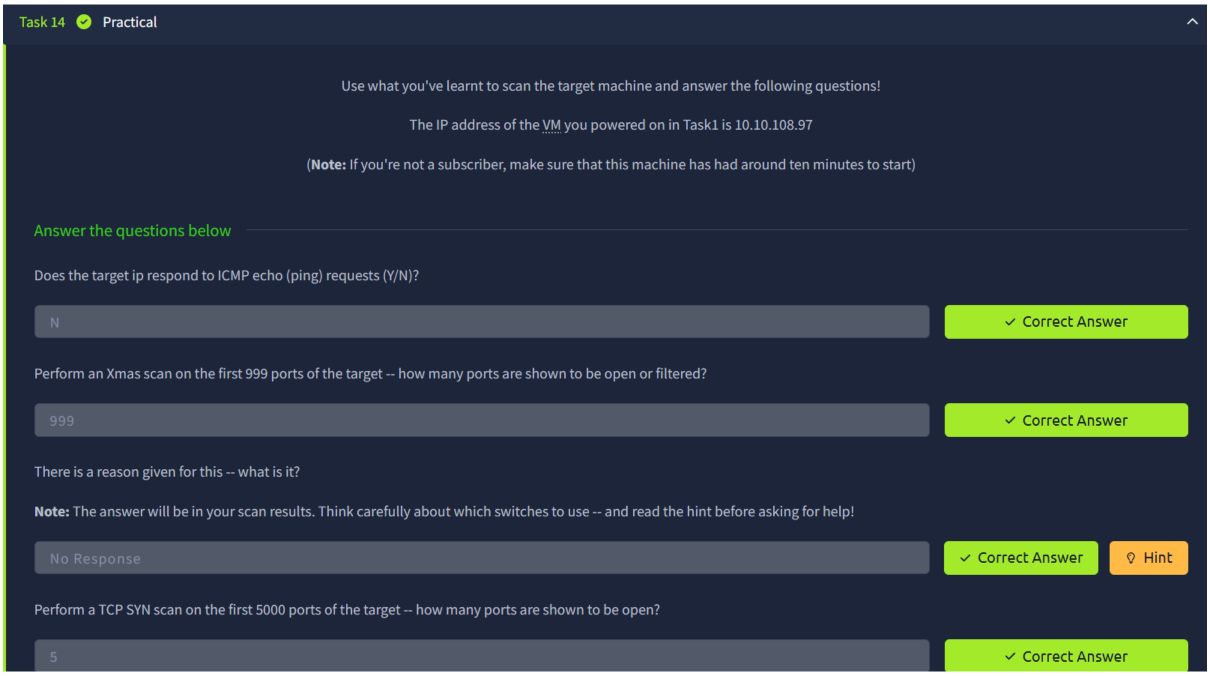
- `-f` :- Used to fragment the packets (i.e. split them into smaller pieces) making it less likely that the packets will be detected by a firewall or IDS.
- An alternative to `-f`, but providing more control over the size of the packets: `--mtu <number>`, accepts a maximum transmission unit size to use for the packets sent. This *must* be a multiple of 8.
- `--scan-delay <time>ms` :- used to add a delay between packets sent. This is very useful if the network is unstable, but also for evading any time-based firewall/IDS triggers which may be in place.
- `--badsum` :- this is used to generate in invalid checksum for packets. Any real TCP/IP stack would drop this packet, however, firewalls may potentially respond automatically, without bothering to check the checksum of the packet. As such, this switch can be used to determine the presence of a firewall/IDS.

Task 14: Practical

The final task is a practical exercise that requires applying all the learned concepts to a target machine. The questions confirm the user's ability to:

- Determine if a host responds to pings.
- Perform specific scan types like an Xmas scan.
- Analyze scan results to understand why certain ports appear open or filtered.
- Run a comprehensive TCP SYN scan to identify all open ports.

This serves as a capstone for the room, ensuring the user can effectively use Nmap in a real-world scenario.



Conclusion

The TryHackMe Nmap room provides a comprehensive, hands-on journey through the capabilities of the Nmap tool. The provided screenshots demonstrate a logical progression from foundational knowledge of network ports and basic scans to advanced techniques involving the Nmap Scripting Engine and firewall evasion. The room culminates in a practical challenge that solidifies the user's understanding and prepares them for using Nmap in security assessments and penetration testing engagements.