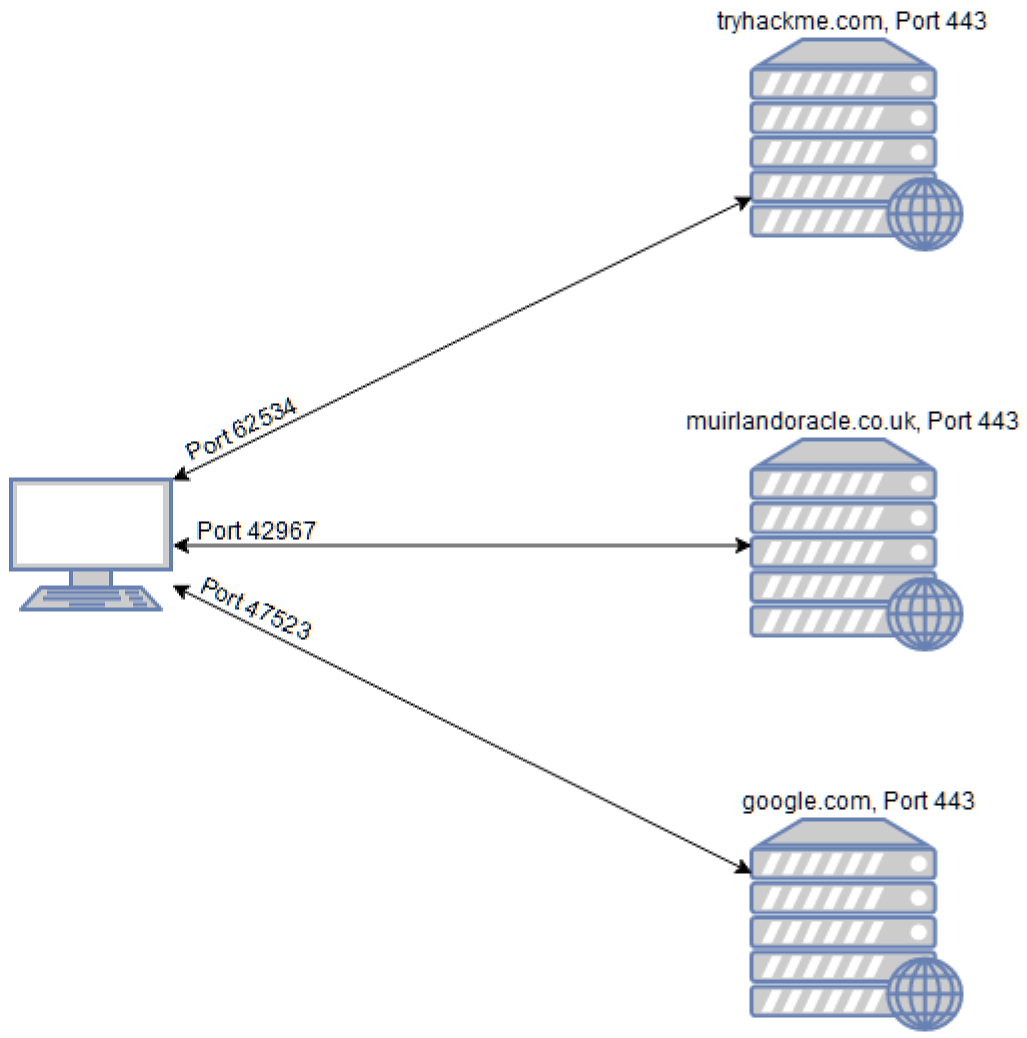
NMAP TUTORIAL

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.



As in the previous example, the diagram shows what happens when you connect to numerous websites at the same time. Your computer opens up a different, high-numbered port (at random), which it uses for all its communications with the remote server.

Every computer has a total of 65535 available ports; however, many of these are registered as standard ports. For example, a HTTP Webservice can nearly always be found on port 80 of the server. A HTTPS Webservice can be found on port 443. Windows NETBIOS can be found on port 139 and SMB can be found on port 445. It is important to note; however, that especially in a CTF setting, it is not unheard of for even these standard ports to be altered, making it even more imperative that we perform appropriate enumeration on the target.

If we do not know which of these ports a server has open, then we do not have a hope of successfully attacking the target; thus, it is crucial that we begin any attack with a port scan. This can be accomplished in a variety of ways – usually using a tool called nmap, which is the focus of this room. Nmap can be used to perform many different kinds of port scan – the most common of these will be introduced in upcoming tasks; however, the basic theory is this: nmap will connect to each port of the target in turn. Depending on how the port responds, it can be determined as being open, closed, or filtered (usually by a firewall). Once we know which ports are open, we can then look at enumerating which services are running on each port – either manually, or more commonly using nmap.

So, why nmap? The short answer is that it's currently the industry standard for a reason: no other port scanning tool comes close to matching its functionality (although some newcomers are now matching it for speed). It is an extremely powerful tool – made even more powerful by its scripting engine which can be used to scan for vulnerabilities, and in some cases even perform the exploit directly! Once again, this will be covered more in upcoming tasks.

For now, it is important that you understand: what port scanning is; why it is necessary; and that nmap is the tool of choice for any kind of initial enumeration

Nmap switch

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 (`-`).

Nmap scan type

When p ort scanning with Nmap, there are three basic scan types. These are:

- TCP Connect Scans (`-sT`)
- SYN "Half-open" Scans (`-sS`)
- UDP Scans (`-sU`)

Additionally there are several less common port scan types, some of which we will also cover (albeit in less detail). These are:

- TCP Null Scans (`-sN`)
- TCP FIN Scans (`-sF`)
- TCP Xmas Scans (`-sX`)

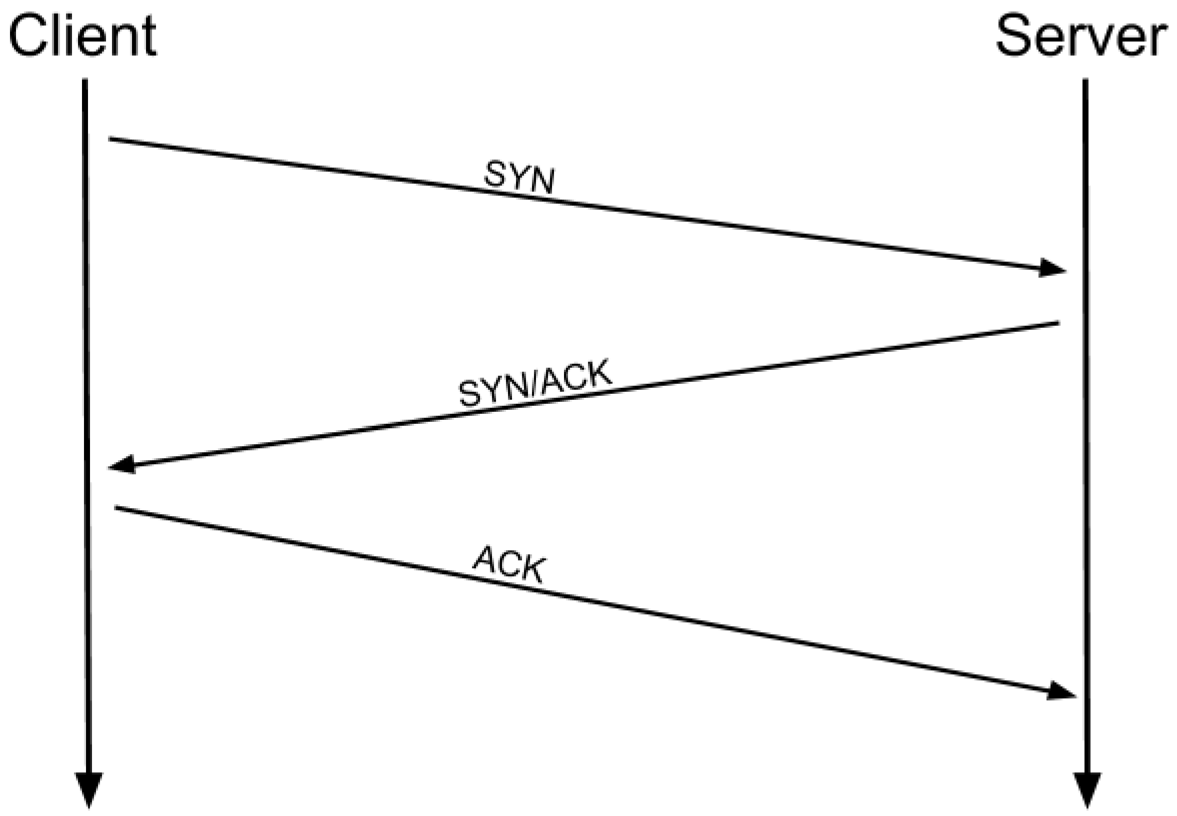
Most of these (with the exception of UDP scans) are used for very similar purposes, however, the way that they work differs between each scan. This means that, whilst one of the first three scans are likely to be your go-to in most situations, it's worth bearing in mind that other scan types exist.

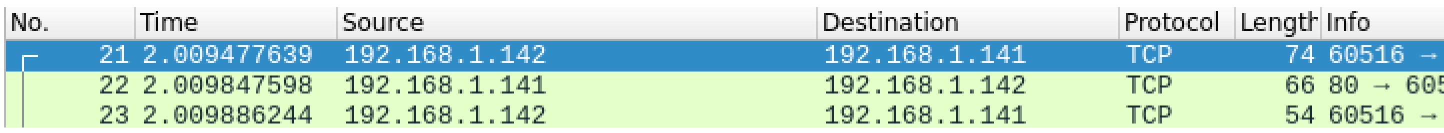
In terms of network scanning, we will also look briefly at ICMP (or "ping") scanning.

Nmap TCP scan

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 →
22	2.009847598	192.168.1.141	192.168.1.142	TCP	66	80 → 605
23	2.009886244	192.168.1.142	192.168.1.141	TCP	54	60516 →

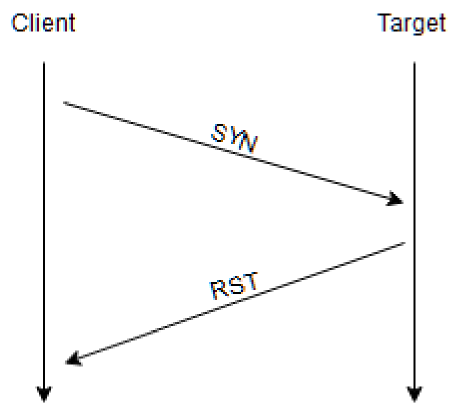
This is one of the fundamental principles of TCP/IP networking, but how does it relate to Nmap?

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.

For example, if a port is closed, RFC 793 states that:

"... If the connection does not exist (CLOSED) then a reset is sent in response to any incoming segment except another reset. In particular, SYNs addressed to a non-existent connection are rejected by this means."

In other words, if Nmap sends a TCP request with the *SYN* flag set to a ***closed*** port, the target server will respond with a TCP packet with the *RST* (Reset) flag set. By this response, Nmap can establish that the port is closed.



If, however, the request is sent to an *open* port, the target will respond with a TCP packet with the SYN/ACK flags set. Nmap then marks this port as being *open* (and completes the handshake by sending back a TCP packet with ACK set).

This is all well and good, however, there is a third possibility.

What if the port is open, but hidden behind a firewall?

Many firewalls are configured to simply **drop** incoming packets. Nmap sends a TCP SYN request, and receives nothing back. This indicates that the port is being protected by a firewall and thus the port is considered to be *filtered*.

That said, it is very easy to configure a firewall to respond with a RST TCP packet. For example, in IPtables for Linux, a simple version of the command would be as follows:

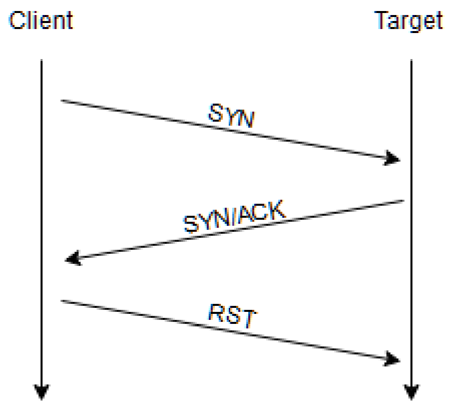
```
iptables -I INPUT -p tcp --dport <port> -j REJECT --reject-with tcp-reset
```

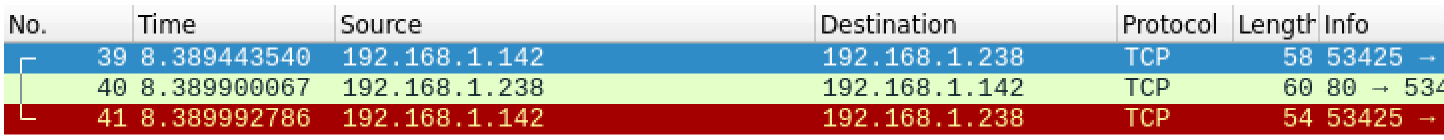
This can make it extremely difficult (if not impossible) to get an accurate reading of the target(s).

Scan Types

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 →
40	8.389900067	192.168.1.238	192.168.1.142	TCP	60	80 → 534
41	8.389992786	192.168.1.142	192.168.1.238	TCP	54	53425 →

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.
- SYN scans are often not logged by applications listening on open ports, as standard practice is to log a connection once it's been fully established. Again, this plays into the idea of SYN scans being stealthy.
- Without having to bother about completing (and disconnecting from) a three-way handshake for every port, SYN scans are significantly faster than a standard TCP Connect scan.

There are, however, a couple of disadvantages to SYN scans, namely:

- They require sudo permissions[1] in order to work correctly in Linux. This is because SYN scans require the ability to create raw packets (as opposed to the full TCP handshake), which is a privilege only the root user has by default.
- Unstable services are sometimes brought down by SYN scans, which could prove problematic if a client has provided a production environment for the test.

All in all, the pros outweigh the cons.

For this reason, SYN scans are the default scans used by Nmap *if run with sudo permissions*. If run **without** sudo permissions, Nmap defaults to the TCP Connect scan we saw in the previous task.

When using a SYN scan to identify closed and filtered ports, the exact same rules as with a TCP Connect scan apply.

If a port is closed then the server responds with a RST TCP packet. If the port is filtered by a firewall then the TCP SYN packet is either dropped, or spoofed with a TCP reset.

In this regard, the two scans are identical: the big difference is in how they handle *open* ports.

---

[1] SYN scans can also be made to work by giving Nmap the CAP_NET_RAW, CAP_NET_ADMIN and CAP_NET_BIND_SERVICE capabilities; however, this may not allow many of the NSE scripts to run properly.

Scan Types

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.

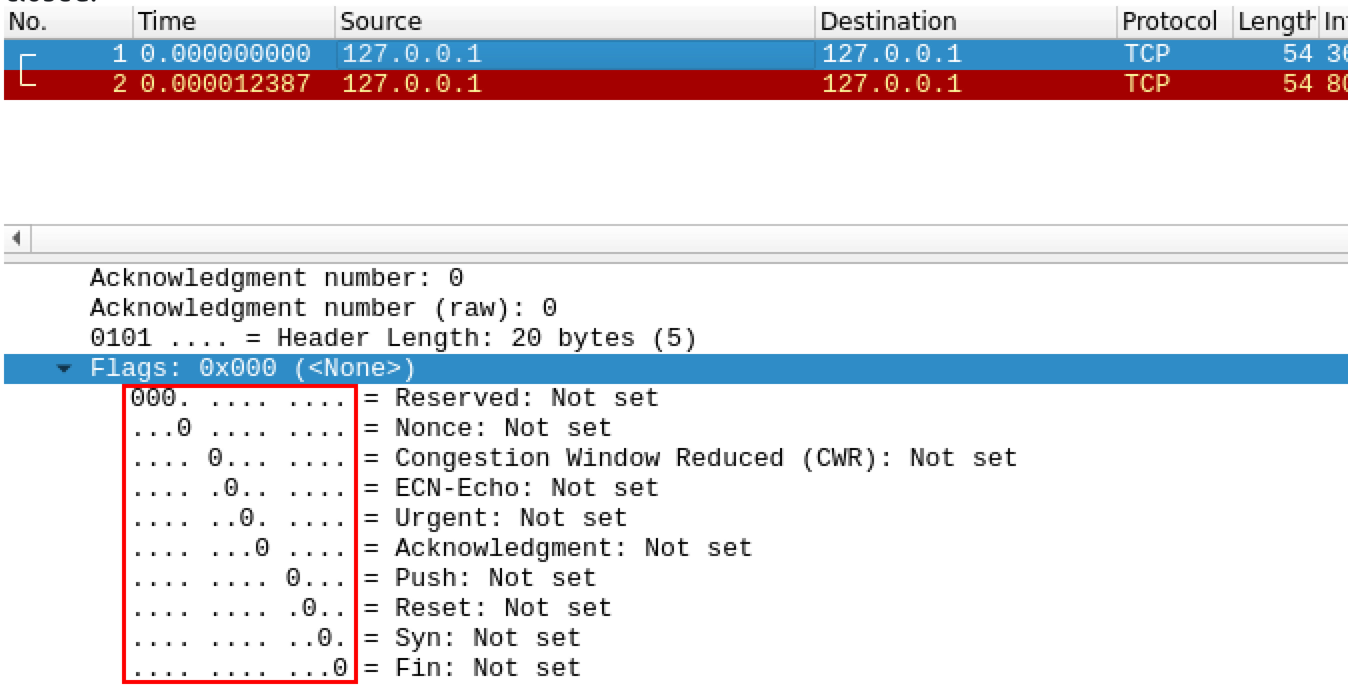
Due to this difficulty in identifying whether a UDP port is actually open, UDP scans tend to be incredibly slow in comparison to the various TCP scans (in the region of 20 minutes to scan the first 1000 ports, with a good connection). For this reason it's usually good practice to run an Nmap scan with `--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.

When scanning UDP ports, Nmap usually sends completely empty requests -- just raw UDP packets. That said, for ports which are usually occupied by well-known services, it will instead send a protocol-specific payload which is more likely to elicit a response from which a more accurate result can be drawn.

Scan Types

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:

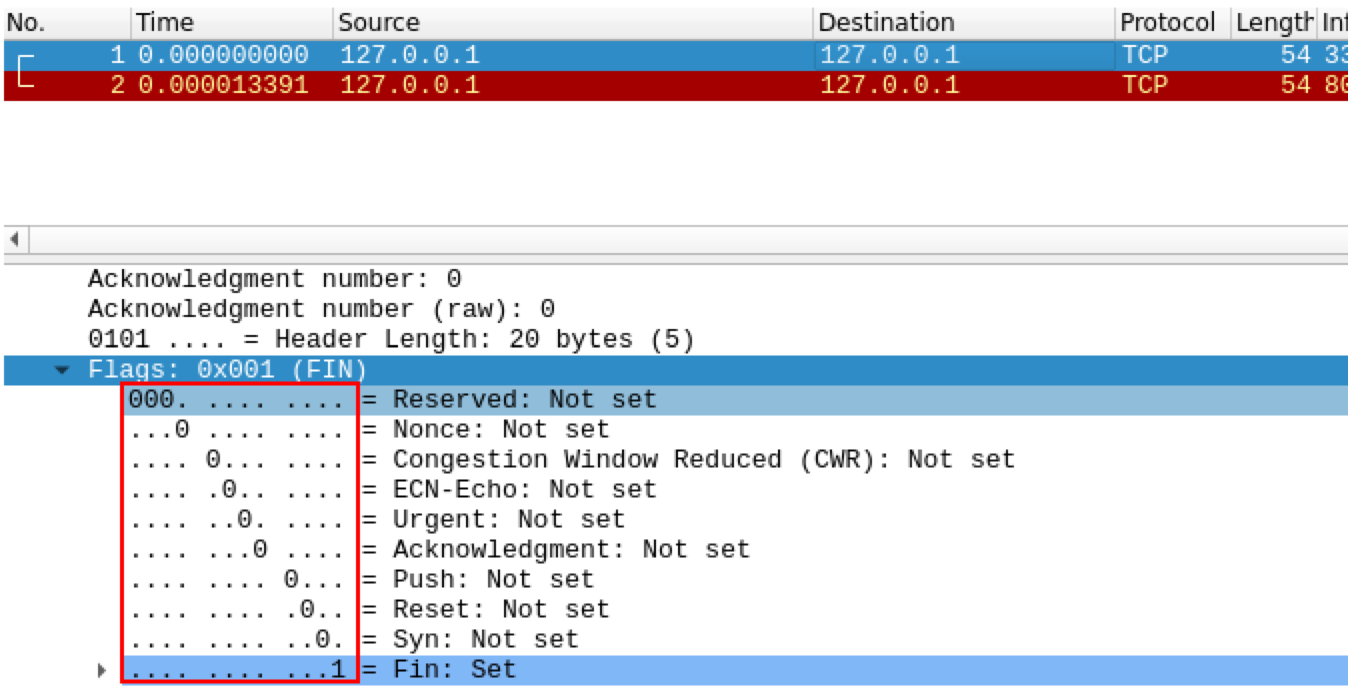
- As the name suggests, NULL scans (`-sN`) are when the TCP request is sent with no flags set at all. As per the RFC, the target host should respond with a RST if the port is closed.



```
No.   Time          Source      Destination   Protocol  Length  In
  1   0.000000000   127.0.0.1   127.0.0.1     TCP       54      3(
  2   0.000012387   127.0.0.1   127.0.0.1     TCP       54      8(

     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
```

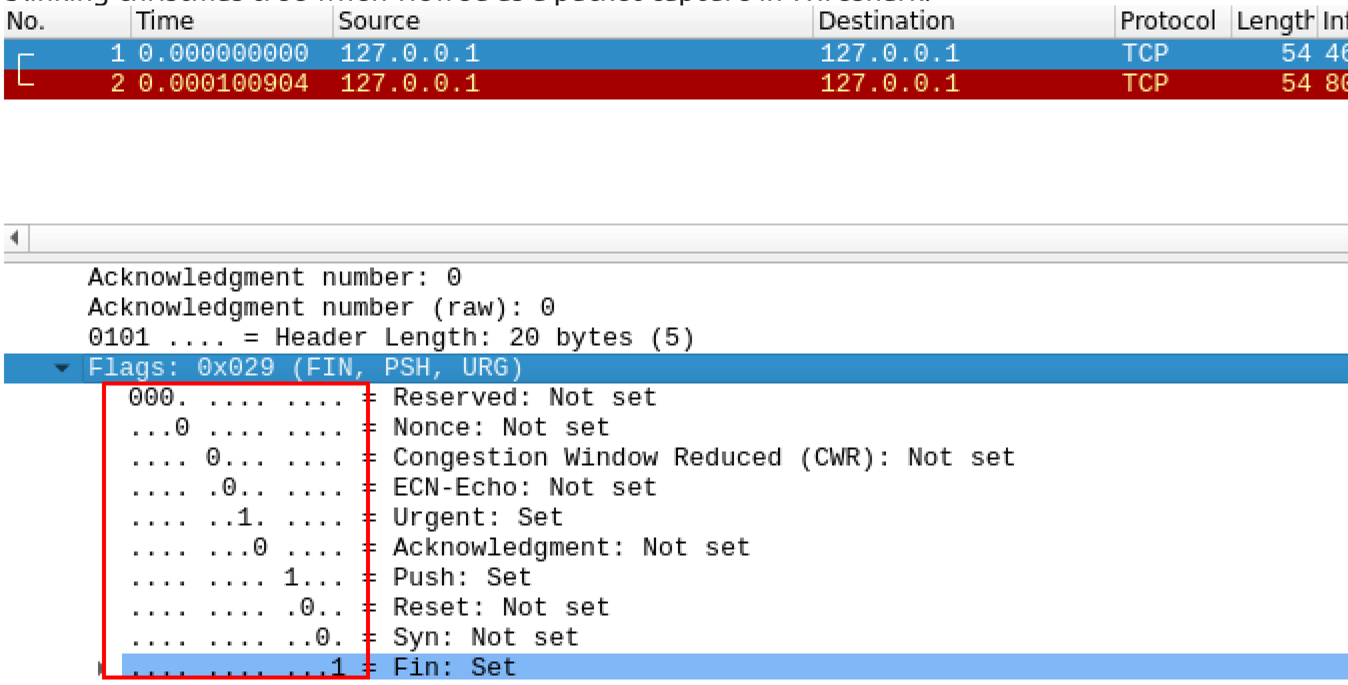
- FIN scans (`-sF`) work in an almost identical fashion; however, instead of sending a completely empty packet, a request is sent with the FIN flag (usually used to gracefully close an active connection). Once again, Nmap expects a RST if the port is closed.



```
No.  Time         Source       Destination  Protocol  Length  Inf
   1 0.000000000  127.0.0.1    127.0.0.1    TCP          54  33
   2 0.000013391  127.0.0.1    127.0.0.1    TCP          54  80
```

```
    Acknowledgment number: 0
    Acknowledgment number (raw): 0
    0101 .... = Header Length: 20 bytes (5)
  Flags: 0x001 (FIN)
      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
      .... .... ...1 = Fin: Set
```

- As with the other two scans in this class, Xmas scans (-sX) send a malformed TCP packet and expects a RST response for closed ports. It's referred to as an xmas scan as the flags that it sets (PSH, URG and FIN) give it the appearance of a blinking christmas tree when viewed as a packet capture in Wireshark.



```
No.  Time         Source       Destination  Protocol  Length  Inf
   1 0.000000000  127.0.0.1    127.0.0.1    TCP          54  46
   2 0.000100904  127.0.0.1    127.0.0.1    TCP          54  80
```

```
    Acknowledgment number: 0
    Acknowledgment number (raw): 0
    0101 .... = Header Length: 20 bytes (5)
  Flags: 0x029 (FIN, PSH, URG)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..1. .... = Urgent: Set
      .... ...0 .... = Acknowledgment: Not set
      .... .... 1... = Push: Set
      .... .... .0.. = Reset: Not set
      .... .... ..0. = Syn: Not set
      .... .... ...1 = Fin: Set
```

The expected response for *open* ports with these scans is also identical, and is very similar to that of a UDP scan. If the port is open then there is no response to the malformed packet. Unfortunately (as with open UDP ports), that is *also* an expected behaviour if the port is protected by a firewall, so NULL, FIN and Xmas scans will only ever identify ports as being *open|filtered*, *closed*, or *filtered*. If a port is identified as filtered with one of these scans then it is usually because the target has responded with an ICMP unreachable packet.

It's also worth noting that while RFC 793 mandates that network hosts respond to malformed packets with a RST TCP packet for closed ports, and don't respond at all for open ports; this is not always the case in practice. In particular Microsoft Windows (and a lot of Cisco network devices) are known to respond with a RST to any malformed TCP packet -- regardless of whether the port is actually open or not. This results in all ports showing up as being closed.

That said, the goal here is, of course, firewall evasion. Many firewalls are configured to drop incoming TCP packets to blocked ports which have the SYN flag set (thus blocking new connection initiation requests). By sending requests which do not contain the SYN flag, we effectively bypass this kind of firewall. Whilst this is good in theory, most modern IDS solutions are savvy to these scan types, so don't rely on them to be 100% effective when dealing with modern systems.

Scan Types

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.

NSE Scripts

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

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>`).

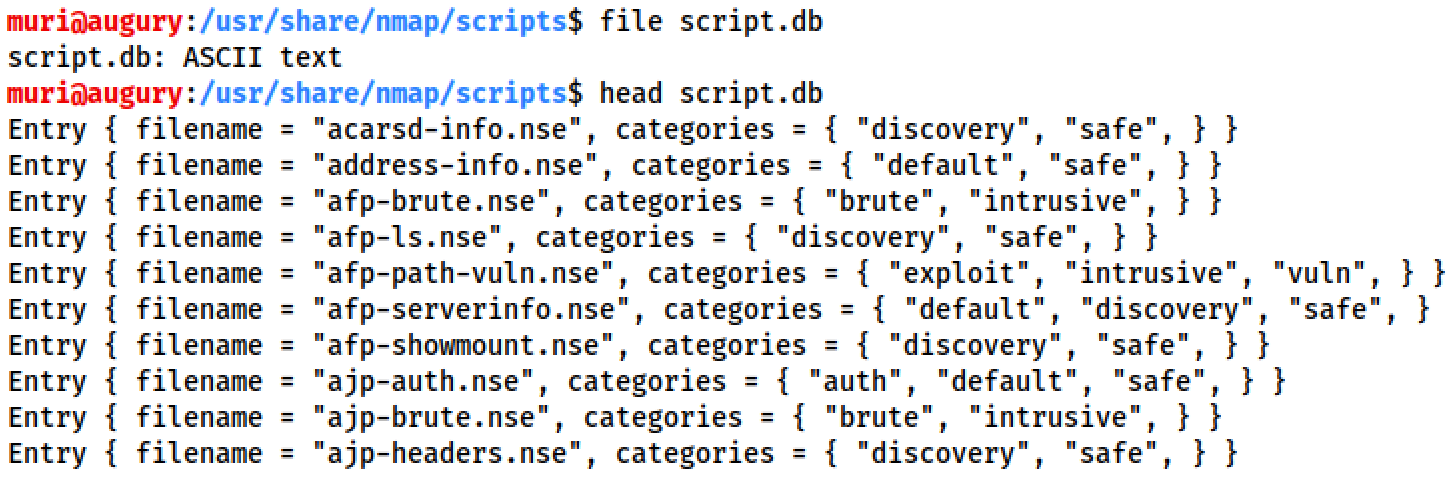
A full list of scripts and their corresponding arguments (along with example use cases) can be found here.

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.

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

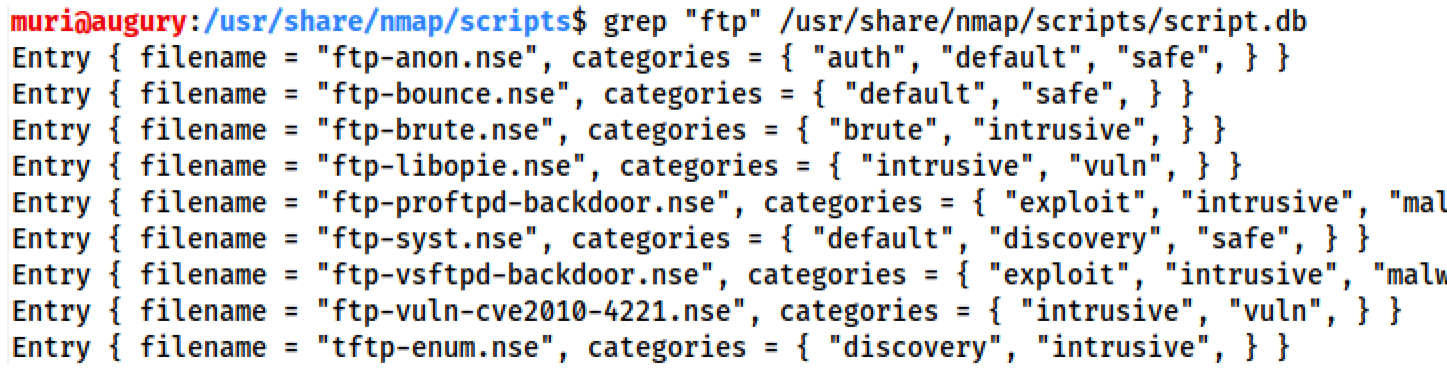
We have two options for this, which should ideally be used in conjunction with each other. The first is the page on the Nmap website (mentioned in the previous task) which contains a list of all official scripts. The second is the local storage on your attacking machine. Nmap stores its scripts on Linux 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.



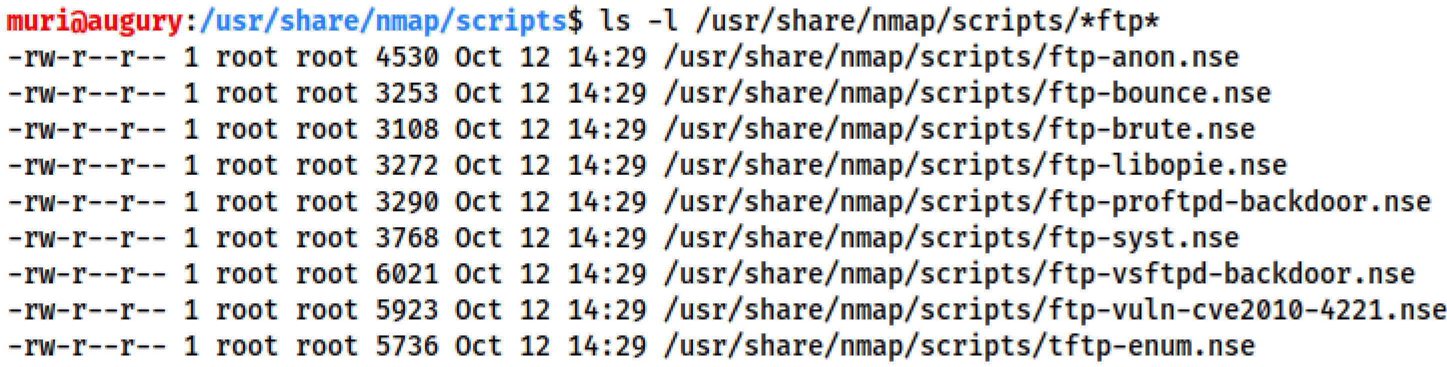
```
muri@augury:/usr/share/nmap/scripts$ file script.db
script.db: ASCII text
muri@augury:/usr/share/nmap/scripts$ head script.db
Entry { filename = "acarsd-info.nse", categories = { "discovery", "safe", } }
Entry { filename = "address-info.nse", categories = { "default", "safe", } }
Entry { filename = "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", } }
Entry { filename = "ftp-libopie.nse", categories = { "intrusive", "vuln", } }
Entry { filename = "ftp-proftpd-backdoor.nse", categories = { "exploit", "intrusive", "mal
Entry { filename = "ftp-syst.nse", categories = { "default", "discovery", "safe", } }
Entry { filename = "ftp-vsftpd-backdoor.nse", categories = { "exploit", "intrusive", "malw
Entry { filename = "ftp-vuln-cve2010-4221.nse", categories = { "intrusive", "vuln", } }
Entry { filename = "tftp-enum.nse", categories = { "discovery", "intrusive", } }
```

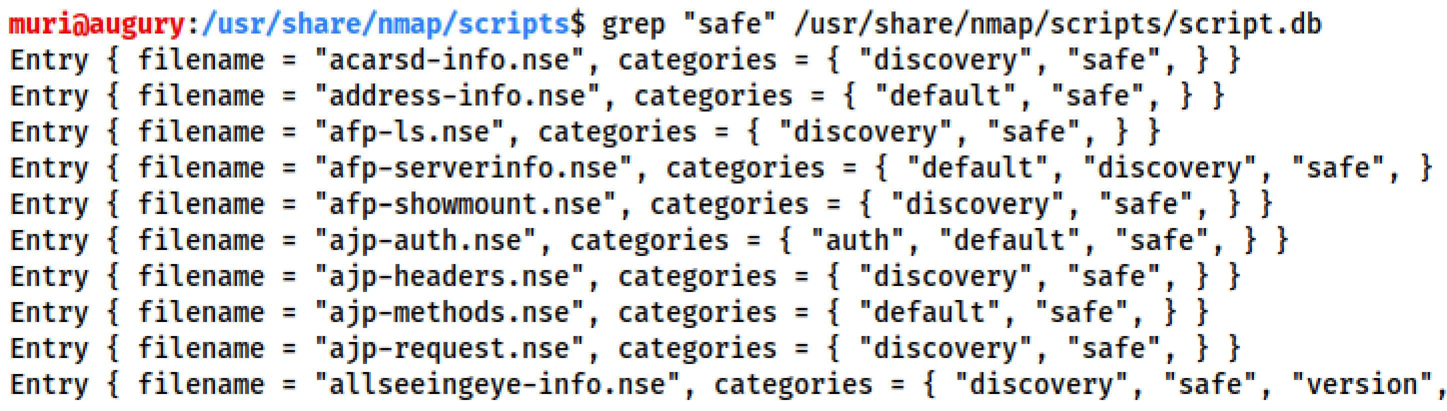
The second way to search for scripts is quite simply to use the `ls` command. For example, we could get the same results as in the previous screenshot by using `ls -l /usr/share/nmap/scripts/*ftp*`:



```
muri@augury:/usr/share/nmap/scripts$ ls -l /usr/share/nmap/scripts/*ftp*
-rw-r--r-- 1 root root 4530 Oct 12 14:29 /usr/share/nmap/scripts/ftp-anon.nse
-rw-r--r-- 1 root root 3253 Oct 12 14:29 /usr/share/nmap/scripts/ftp-bounce.nse
-rw-r--r-- 1 root root 3108 Oct 12 14:29 /usr/share/nmap/scripts/ftp-brute.nse
-rw-r--r-- 1 root root 3272 Oct 12 14:29 /usr/share/nmap/scripts/ftp-libopie.nse
-rw-r--r-- 1 root root 3290 Oct 12 14:29 /usr/share/nmap/scripts/ftp-proftpd-backdoor.nse
-rw-r--r-- 1 root root 3768 Oct 12 14:29 /usr/share/nmap/scripts/ftp-syst.nse
-rw-r--r-- 1 root root 6021 Oct 12 14:29 /usr/share/nmap/scripts/ftp-vsftpd-backdoor.nse
-rw-r--r-- 1 root root 5923 Oct 12 14:29 /usr/share/nmap/scripts/ftp-vuln-cve2010-4221.nse
-rw-r--r-- 1 root root 5736 Oct 12 14:29 /usr/share/nmap/scripts/tftp-enum.nse
```

Note the use of asterisks (`*`) on either side of the search term

The same techniques can also be used to search for categories of script. For example: `grep "safe" /usr/share/nmap/scripts/script.db`



```
muri@augury:/usr/share/nmap/scripts$ grep "safe" /usr/share/nmap/scripts/script.db
Entry { filename = "acarsd-info.nse", categories = { "discovery", "safe", } }
Entry { filename = "address-info.nse", categories = { "default", "safe", } }
Entry { filename = "afp-ls.nse", categories = { "discovery", "safe", } }
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-headers.nse", categories = { "discovery", "safe", } }
Entry { filename = "ajp-methods.nse", categories = { "default", "safe", } }
Entry { filename = "ajp-request.nse", categories = { "discovery", "safe", } }
Entry { filename = "allseeingeye-info.nse", categories = { "discovery", "safe", "version",
```

Installing New Scripts

We mentioned previously that the Nmap website contains a list of scripts, so, what happens if one of these is missing in the `scripts` directory locally? A standard `sudo apt update && sudo apt install nmap` should fix this; however, it's also possible to install the scripts manually by downloading the script from Nmap (`sudo wget -O /usr/share/nmap/scripts/<script-name>.nse https://svn.nmap.org/nmap/scripts/<script-name>.nse`). This must then be followed up with `nmap --script-updatedb`, which updates the `script.db` file to contain the newly downloaded script.

It's worth noting that you would require the same "updatedb" command if you were to make your own NSE script and add it into Nmap -- a more than manageable task with some basic knowledge of Lua!

Task 13

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.

Source url

https://tryhackme.com/room/furthernmap

Features of NMAP

1. Find security issues

2. Identify open ports

3. Detect Vulnerabilities

4. OS Version Detection

5. Provide crucial information like devices types, reverse DNS names, MAC addresses.