
# Linux Command - Flag Test

## ▼ 🧪 Challenge 1: The Shadowed SUID

### Challenge Question:

Inside your CTF folder, there's a **file owned by root**, has the **SUID bit** set, and **no group or**

**others permissions**.

> 🔍 Task:
>
> Find the file. Show its full path and metadata.
>
> Also, explain in one sentence what danger SUID files pose in DFIR.

Command(s): `find`, `ls -l`, `stat`, `file`, etc.

### Command:

```
find CTF-Lite -perm -4000 -exec ls -l {} \; -exec stat {} \;
```

| Segment | Meaning |
| --- | --- |
| `find` | The Unix command used to **search for files and directories** recursively. |
| `CTF-Lite` | The **starting directory** where the search begins. |
| `-perm -4000` | Find files that have the **SUID (Set User ID) permission bit set**. |
| `-exec` | This tells `find` to **run a command on each file it finds** that matches the condition. |
| `ls -l` | A detailed listing of file: permissions, owner, size, and timestamp. |
| `{}` | Placeholder for the **current file name** found by `find`. |
| `\;` | Required to **end the** `-exec` **command** (escaped semicolon). |
| `-exec stat {}` | After `ls -l`, run the `stat` command on each file for **more detailed metadata.** |
| `\;` | Again, ends the second `-exec` command. |

## Summary:

- File path: `~/CTF-Lite/staging/vault/root_exploit.sh`
- Permissions: `-rwsr-xr-x`
- Owner: `root`
- **Why it's dangerous**:
  SUID bit, allows user to execute the file with root level privileges. If the file has malware or exploiter by attackers, it can grant them root level elevated-privileges [which is essentially what they are aiming to achieve more often than not].

## Output:

```
┌──(jynx㉿kali)-[~/CTF-Lite]
└─$ find -perm -4000 -exec ls -l {} \; -exec stat {} \;
-rwsr-xr-x 1 root root 0 Jul  3 13:30 ./staging/vault/root_exploit.sh
  File: ./staging/vault/root_exploit.sh
  Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: 8,1     Inode: 3019047     Links: 1
Access: (4755/-rwsr-xr-x)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2025-07-03 13:33:48.255843352 -0400
Modify: 2025-07-03 13:30:13.754050236 -0400
Change: 2025-07-03 13:30:13.761951037 -0400
 Birth: 2025-07-03 13:30:13.754050236 -0400
```

## ▼ 🧪 Challenge 2: Decoy in the Dark

### Challenge Question:

A file exists inside your `vault` or `secrets` directory that has **0000 permissions**, **non-zero file size**, and **.txt** extension.

> 🔍 Task:
>
> Recover its content if you can. How would an attacker hide a decoy this way?

Command(s): `find` , `chmod` , `ls -alh` , `head` / `cat`

### Command:

```
find -perm 0000 -exec ls -alH {} \; -exec chmod 400 {} \; -exec cat {} \;
```

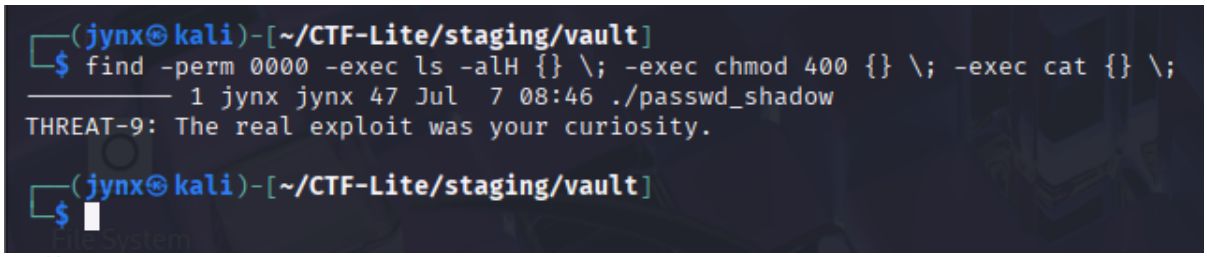
| Segment | Meaning |
| --- | --- |
| `find` | Start searching recursively |
| `-perm 0000` | Find files with **no permissions** set (not readable, writable, or executable by anyone) |
| `-exec ls -alh {}` | Show the file in **long, human-readable format** |
| `\;` | Ends the `-exec` block |

| Segment | Meaning |
| --- | --- |
| `-exec chmod 400 {}` | Change permissions to **read-only** for the owner |
| `\;` | Ends the `chmod` block |
| `-exec cat {}` | Display the contents of the file |
| `\;` | Ends the `cat` block |

### Summary:

- File path: `~/CTF-Lite/staging/vault/passwd_shadow`
- Permissions: `-r--------`
- Owner: `jynx` ***[it's me- because I designed the CTF challenge]***
- **Why it's dangerous**: **Permission 0000** often hints at **high-value or decoy files** in CTFs.
- **FLAG**: ***THREAT-9: The real exploit was your curiosity.***

### Output:




## ▼ 🧪 Challenge 3: Lost & Found

### Challenge Question:

There's a `.pdf` or `.docx` file with **different owner** than you. It was **created after being**

**modified** (which is logically suspicious).

> 🔍 Task:
>
> Locate it. Use `stat` and explain why it's weird in 1 line.

> Log the output.

Command(s): `find` , `stat` , `ls -l` , `grep` , `file`

## Command:

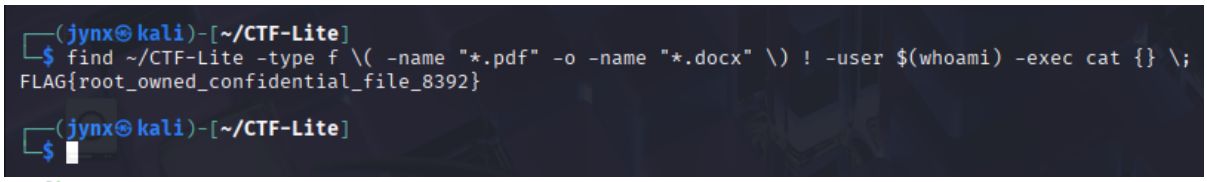
> find ~/CTF-Lite -type f \( -name "*.pdf" -o -name "*.docx" \) ! -user $(whoami) -exec cat {} \;

| Component | Explanation |
|---|---|
| `find` | The command-line utility to search for files and directories in a directory hierarchy |
| `~/CTF-Lite` | Tells `find` to start looking inside the `CTF-Lite` directory in the home directory |
| `-type f` | Limits the search to files only ( `f` stands for file) |
| `\( ... \)` | Groups the next conditions logically — required for combining multiple `-name` filters |
| `-name "*.pdf"` | Matches any file that ends with `.pdf` |
| `-o` | Logical OR — tells `find` to also include files that match the next pattern |
| `-name "*.docx"` | Matches any file that ends with `.docx` |
| `! -user $(whoami)` | `!` negates the test — this filters **files not owned by the current user** |
| `-exec cat {} \;` | For every match, `cat` is executed on the file. `{}` is a placeholder for the found file |
| `\;` | Ends the `-exec` clause — **must** be escaped to avoid being interpreted by the shell |

## Summary:

- File path: `~/CTF-Lite/challenge3/root_docs/top_secret.pdf`
- Permissions: `-rw-r--r--`
- Owner: `root`
- FLAG: ***FLAG{root_owned_confidential_file_8392}***

## Output:




# 🧪 Challenge 4: Permissions Puzzle

## Challenge Question:

There's a file that shows:

```
-rwsr-xr-- 1 root jynx 0 Jul  4 13:30 hacker_key
```

> 🔍 Task:
>
> What does each permission mean?
>
> What group(s) can read it?
>
> Can you execute it as a user?
>
> If `hacker_key` were malware, what would make it dangerous?

Explain — Permission analysis.

## Command:

```
ls -alHR | grep "hacker_key"
```

## Summary:

### 1. What does each permission mean?

→ Here's the permission breakdown for `-rwsr-xr--` in tabular format:

- The `s` in the owner's execute position means both execute permission AND SUID bit are set.

- Only owner/user of the file and root user are permitted to set SUID bits.
- SUID (Set User ID) means when anyone executes this file, it runs with the privileges of the file's owner
- The full octal representation would be `4754` (the leading 4 indicates SUID)

**Permission Summary:**

- **Owner**: Full control (read, write, execute) + SUID
- **Group**: Read and execute only
- **Others**: Read only

This is a common pattern for system utilities that need elevated privileges to function properly, like `passwd` or `sudo`.

### ▼ 2. Can you execute it as a user?

Yes you absolutely can execute it as user/owner of the file.

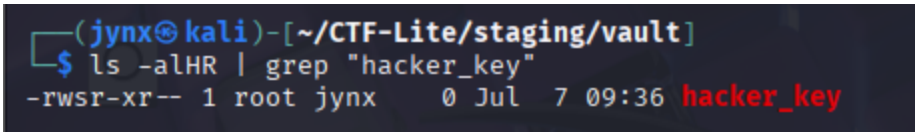
### ▼ 3. What group(s) can read it?

All the three types of users, owner , group and others can read the file.

### ▼ 4. If `hacker_key` were malware, what would make it dangerous?

The ability of attacker to execute ***'hacker_key'*** with sudo privileges, because of the SUID bit, and a potent to capture root privileges.

**Output:**




## ▼ 🧪 Challenge 5: Flag Hunter

**Challenge Question:**

All `.txt` files contain either normal decoys or “FLAG” lines.

Search across entire `CTF-Lite` folder and print **only the lines that contain the word `FLAG` (case-sensitive)**.

> 🔍 Task:
>
> Count how many flags there are. Paste the 3 most relevant lines below your solution.

Command(s): `grep` , `find` , `cat` , `xargs` , `wc`

## Command:

```
find CTF-Lite -name "*.txt" -exec cat {} \; | grep -i "flag"
```

| Component | Description |
|---|---|
| `find CTF-Lite` | Recursively search inside the CTF-Lite directory |
| `-name "*.txt"` | Match all files that end in `.txt` |
| `-exec cat {} \;` | For each `.txt` file found, print its content |
| `grep -i "flag"` | filter out text based on whether the contents of the file has ‘flag’ keyword. |

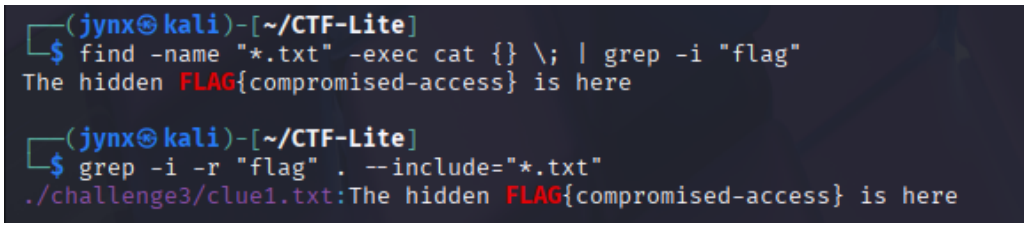
**Want to see which file the flag was in?**

```
grep -i -r "flag" CTF-Lite --include="*.txt"
```

## Summary:

- File path: `~/CTF-Lite/challenge3/root_docs/top_secret.pdf`
- Permissions: `-rw-r--r--`
- Owner: `root`
- FLAG: ***FLAG{root_owned_confidential_file_8392}***

## Output:


┌──(jynx㉿kali)-[~/CTF-Lite]
└─$ find -name "*.txt" -exec cat {} \; | grep -i "flag"
The hidden FLAG{compromised-access} is here
┌──(jynx㉿kali)-[~/CTF-Lite]
└─$ grep -i -r "flag" . --include="*.txt"
./challenge3/clue1.txt:The hidden FLAG{compromised-access} is here