The Art of Binary Exploitation

Yusuf Hegazy

1 February 2024

$ whoami

- Electrical Engineering BG
- Currently work @ CyberDefenders
- Pwn, RE, and Hardware by Night

Agenda

- What is Binary Exploitation
- Buffer Overflow Exploits
- Heap Exploits
- Race Conditions

Binary

Binary

WIKIPEDIA
The Free Encyclopedia

A binary file is a computer file that is not a text file.

Exploit

WIKIPEDIA
The Free Encyclopedia

An exploit is a piece of software, a chunk of data, or a sequence of commands that takes advantage of a bug.

The Most Famous Binary Exploit

The Most Famous Binary Exploit

The Buffer Overflow

So a buffer overflow allows us to change the return address of a function, In this way we can change the flow of execution of the program.

strcpy() will then copy [the shellcode] onto buffer without doing any bounds checking, and will overflow the return address, overwriting it with the address where our code is now located Once we reach the end of main and it tried to return it jumps to our code, and execs a shell.

— Aleph1 | Phrack Issue 49

Date: Sun, 10 Aug 1997 17:29:46 –0300
From: Solar Designer <solar@FALSE.COM>
To: BUGTRAQ@NETSPACE.ORG
Subject: Getting around non-executable stack (and fix)

Hello!
I finally decided to post a return-into-libc overflow exploit.
...
That's all for now. I hope I managed to prove that exploiting buffer overflows should be an art.

— Solar Designer | Bugtraq

But is this It?

Heap Exploits

Heap Exploits

- Heap uses metadata to manage Its allocations

Heap Exploits

- Heap uses metadata to manage Its allocations
- Controlling this metadata can allow us to control the heap

```
char* heap_address = malloc(100); // 0x55f72e6bb2c0
strcpy(heap_address, "AAAAAAAABBBBBBBBCCCCCCCC");
```

```
char* heap_address = malloc(100); // 0x55f72e6bb2c0
strcpy(heap_address, "AAAAAAAABBBBBBBBCCCCCCCC");
```

```
gef➤  x/gx 0x55f72e6bb2c0
0x55f72e6bb2c0: 0x4141414141414141
gef➤  x/gx 0x55f72e6bb2c0+8
0x55f72e6bb2c8: 0x4242424242424242
gef➤  x/gx 0x55f72e6bb2c0+16
0x55f72e6bb2d0: 0x4343434343434343
```

```c
char* heap_address = malloc(100); // 0x55f72e6bb2c0
strcpy(heap_address, "AAAAAAAABBBBBBBBCCCCCCCC");
```

```
gef➤  x/gx 0x55f72e6bb2c0
0x55f72e6bb2c0: 0x4141414141414141
gef➤  x/gx 0x55f72e6bb2c0+8
0x55f72e6bb2c8: 0x4242424242424242
gef➤  x/gx 0x55f72e6bb2c0+16
0x55f72e6bb2d0: 0x4343434343434343
```

```
gef➤  x/gx 0x55f72e6bb2c0-8
0x55f72e6bb2b8: 0x0000000000000071
```

```
char* heap_address = malloc(100); // 0x55f72e6bb2c0
strcpy(heap_address, "AAAAAAAABBBBBBBBCCCCCCCC");
```

```
gef➤  x/gx 0x55f72e6bb2c0
0x55f72e6bb2c0: 0x4141414141414141
gef➤  x/gx 0x55f72e6bb2c0+8
0x55f72e6bb2c8: 0x4242424242424242
gef➤  x/gx 0x55f72e6bb2c0+16
0x55f72e6bb2d0: 0x4343434343434343

gef➤  x/gx 0x55f72e6bb2c0-8
0x55f72e6bb2b8: 0x0000000000000071
```

```
char* heap_address = malloc(100); // 0x55f72e6bb2c0
strcpy(heap_address, "AAAAAAAABBBBBBBBCCCCCCCC");
```

```
gef➤  x/gx 0x55f72e6bb2c0
0x55f72e6bb2c0: 0x4141414141414141
gef➤  x/gx 0x55f72e6bb2c0+8
0x55f72e6bb2c8: 0x4242424242424242
gef➤  x/gx 0x55f72e6bb2c0+16
0x55f72e6bb2d0: 0x4343434343434343

gef➤  x/gx 0x55f72e6bb2c0-8
0x55f72e6bb2b8: 0x0000000000000071
```


```
free(heap_address);
char *new_heap_address = malloc(100); // 0x55f72e6bb2c0
```

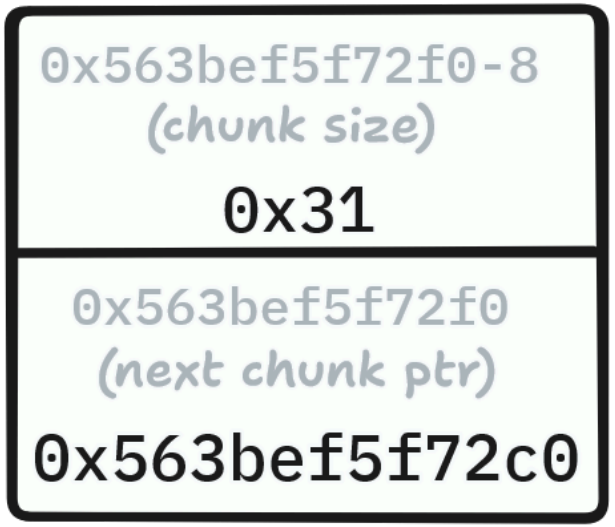
```
char* a = malloc(30); // 0x563bef5f72c0
char* b = malloc(30); // 0x563bef5f72f0
free(a);
free(b);
```

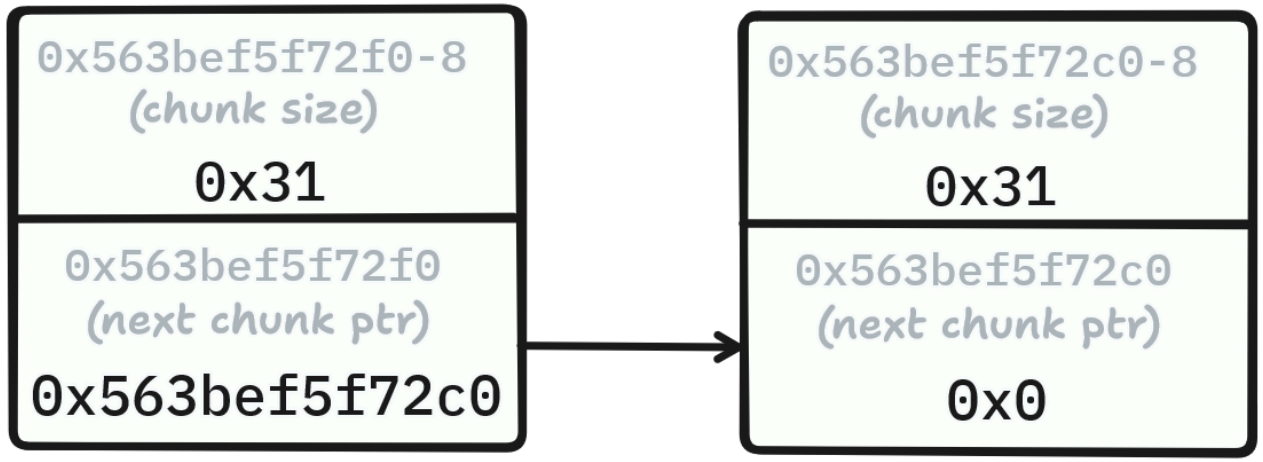
```
char* a = malloc(30); // 0x563bef5f72c0
char* b = malloc(30); // 0x563bef5f72f0
free(a);
free(b);
```

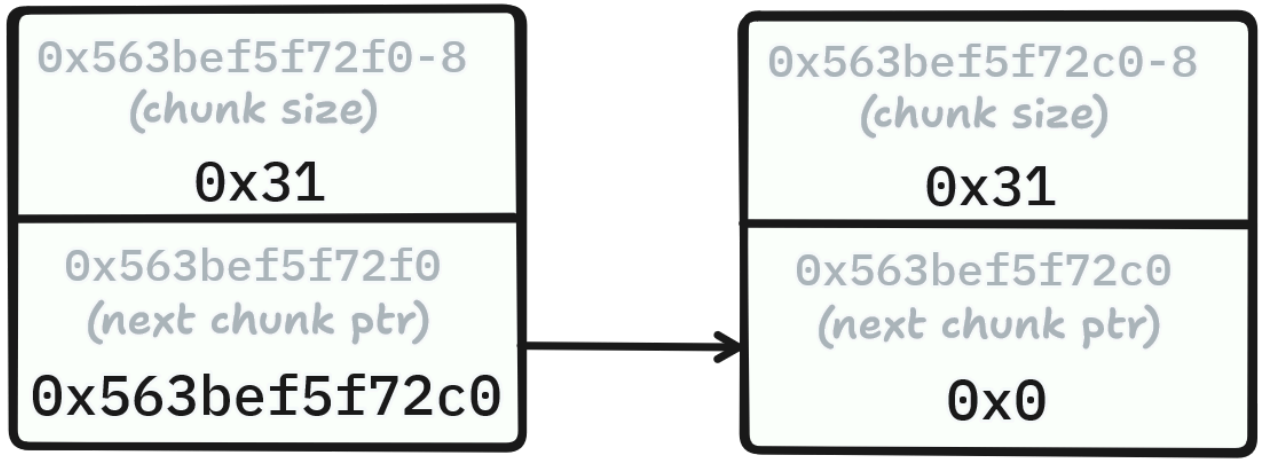
```
gef➤  x/gx 0x563bef5f72f0-8
0x563bef5f72e8: 0x0000000000000031
gef➤  x/gx 0x563bef5f72f0
0x563bef5f72f0: 0x0000563bef5f72c0
```

```c
char* a = malloc(30); // 0x563bef5f72c0
char* b = malloc(30); // 0x563bef5f72f0
free(a);
free(b);
```

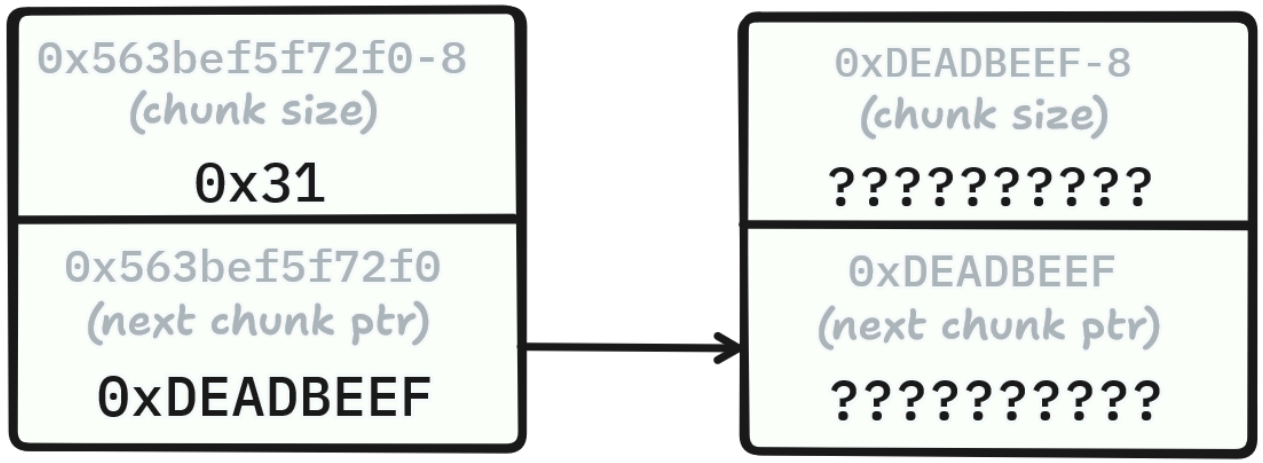
```
gef➤  x/gx 0x563bef5f72f0-8
0x563bef5f72e8: 0x0000000000000031
gef➤  x/gx 0x563bef5f72f0
0x563bef5f72f0: 0x0000563bef5f72c0
```

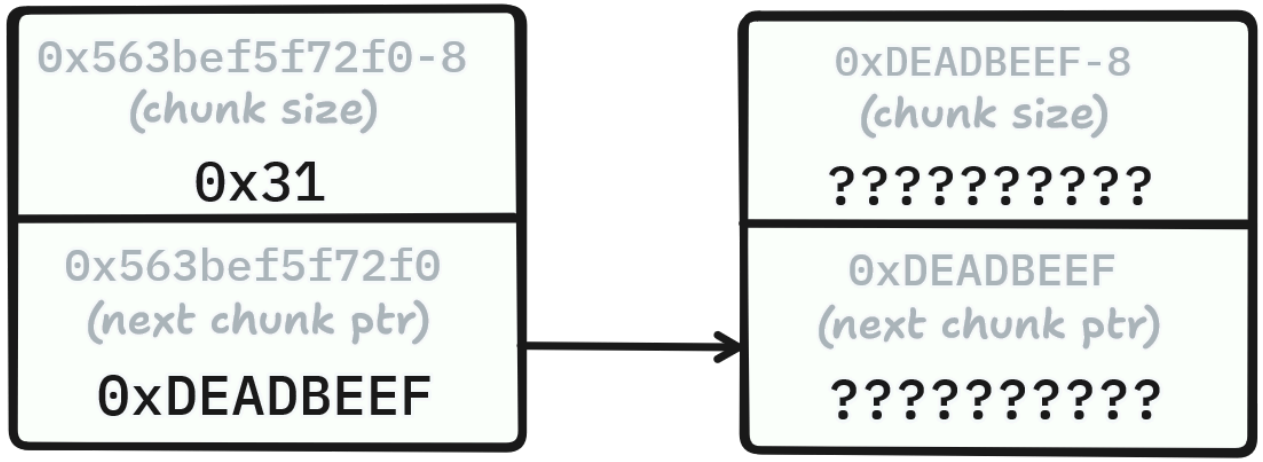



0x563bef5f72f0-8
(chunk size)
0x31
0x563bef5f72f0
(next chunk ptr)
0x563bef5f72c0
0x563bef5f72c0-8
(chunk size)
0x31
0x563bef5f72c0
(next chunk ptr)
0x0




```
char* x = malloc(30); // 0x563bef5f72f0
char* y = malloc(30); // 0x563bef5f72c0
```


0x563bef5f72f0-8
(chunk size)
0x31
0x563bef5f72f0
(next chunk ptr)
0xDEADBEEF
0xDEADBEEF-8
(chunk size)
??????????
0xDEADBEEF
(next chunk ptr)
??????????




```
char* x = malloc(30); // 0x563bef5f72f0
char* y = malloc(30); // 0xDEADBEEF
```

more heap techniques

https://github.com/shellphish/how2heap

Race Conditions

Demo

```
int main(int argc, char **argv)
{
  int fd = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0755);
  write(fd, "#!/bin/sh\necho SAFE\n", 20);
  close(fd);
  execl("/bin/sh", "/bin/sh", argv[1], NULL);
}
```

TOCTOU

Time of Check Time of Use

- when we can modify data between time of check and time of use

- when we can modify data between time of check and time of use

```
if (access("file", W_OK) != 0) {
  exit(1);
}
fd = open("file", O_WRONLY);
write(fd, buffer, sizeof(buffer));
```

How to win races?

1. Increase our speed
2. Slow down the victim

Increasing our Speed

- Use c/python instead of bash

Increasing our Speed

- Use c/python instead of bash
- renameat2() magic syscall

Slowing Down the Victim

- `nice` and `ionice` from GNU coreutils

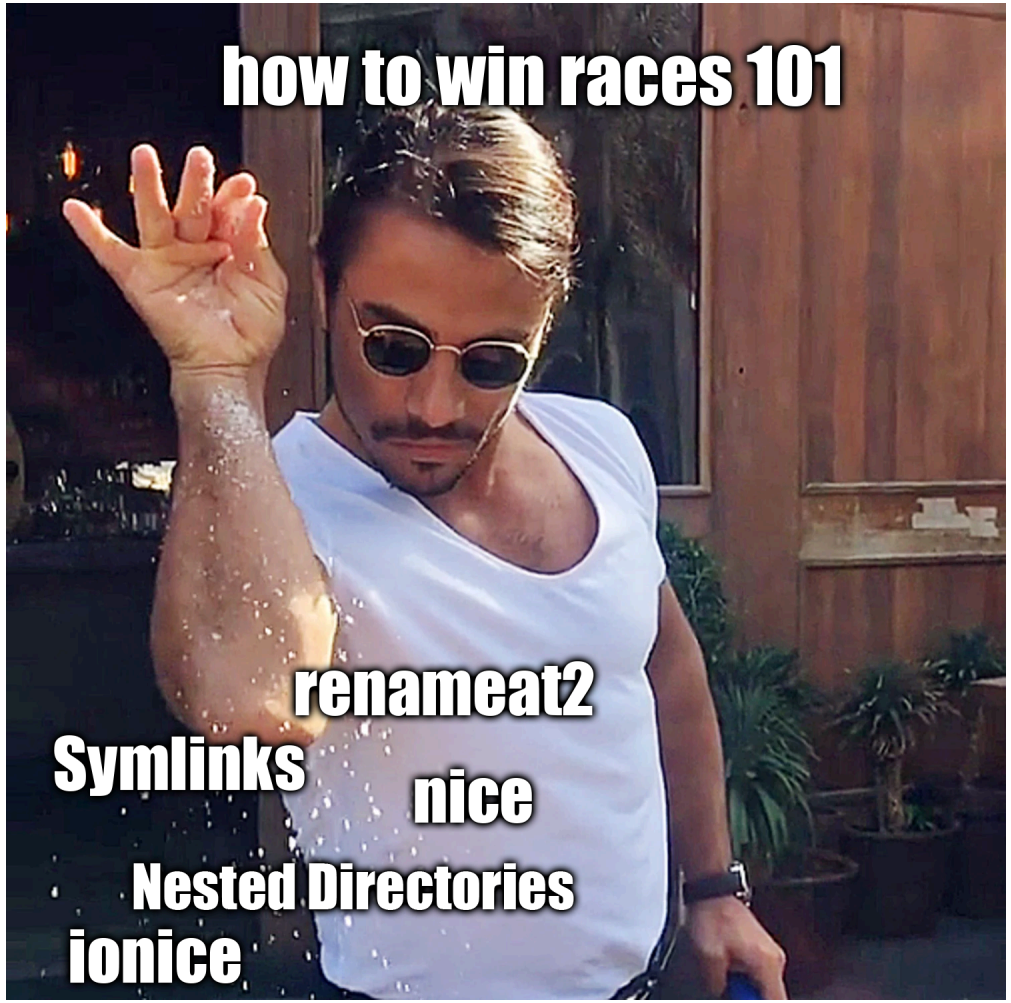
Slowing Down the Victim

- `nice` and `ionice` from GNU coreutils
- using filesystem mazes

Filesystem Maze

Which of these takes more time?

- cat file.txt
- cat a/b/c/d/e/f/g/h/i/j/k/l/m/n/o/p/q/r/s/t/u/v/w/x/y/z/file.txt


how to win races 101
renameat2
Symlinks
nice
Nested Directories
ionice

FS Maze Demo

Resources

- https://pwn.college
- https://www.youtube.com/@LiveOverflow
- https://www.youtube.com/@GynvaelEN