Recall: Steps of Stack Smashing Attack

1. Find a buffer overflow vulnerability in the program
2. Inject shellcode into a known memory address
3. Exploit the buffer overflow vulnerability to overwrite EIP with the shellcode address.
4. Return from the vulnerable function.
5. Start to execute the shellcode.

Solution:

- Non-Executable Memory

Non-Executable Memory

Key idea

- Attackers inject the malicious code into the memory, and then jump to it.
- We can configure the writable memory region to be non-executable, and thus preventing the malicious code from being executed.
- Windows: Data Execution Prevention (DEP)
- Linux: ExecShield

```
# sysctl -w kernel.exec-shield=1   // Enable ExecShield
# sysctl -w kernel.exec-shield=0   // Disable ExecShield
```

Hardware support

- AMD64 (**`NX-bit`**), Intel x86 (**`XD-bit`**), ARM (**`XN-bit`**)
- Each Page Table Entry (PTE) has an attribute to control if the page is executable

Insecurity of Non-Executable Memory

Non-Executable Memory protection does not work when the attacker does not inject malicious code, but just using existing code

Return-to-lib attack:

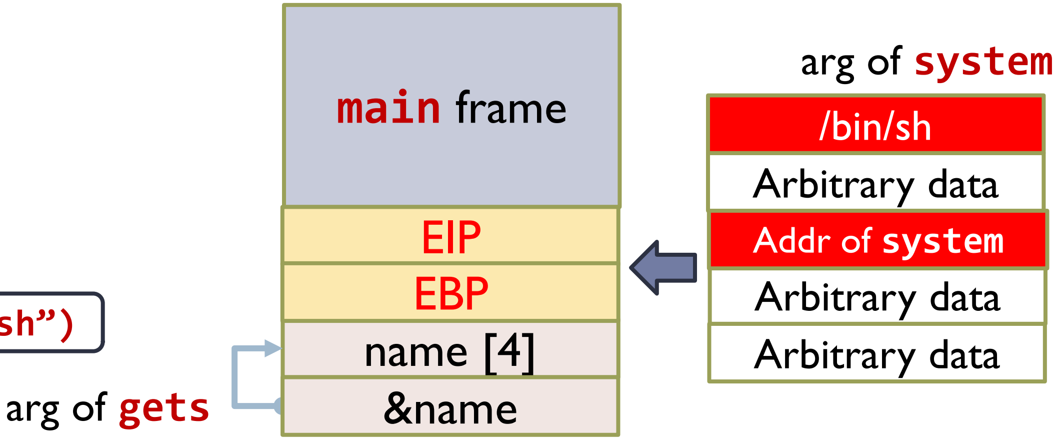
- Replace the return address with the address of an existing function in the standard C library (libc) or common operating system function.

```
int system(char *command);

void vulnerable(void){
    char name [4];
    gets(name);
}

int main(void){
    vulnerable();
    return 0;
}
```

system (“/bin/sh”)



main frame

EIP

EBP

name [4]

&name

arg of gets

arg of system

/bin/sh

Arbitrary data

Addr of system

Arbitrary data

Arbitrary data

Insecurity of Non-Executable Memory

Return-Oriented Programming (ROP):

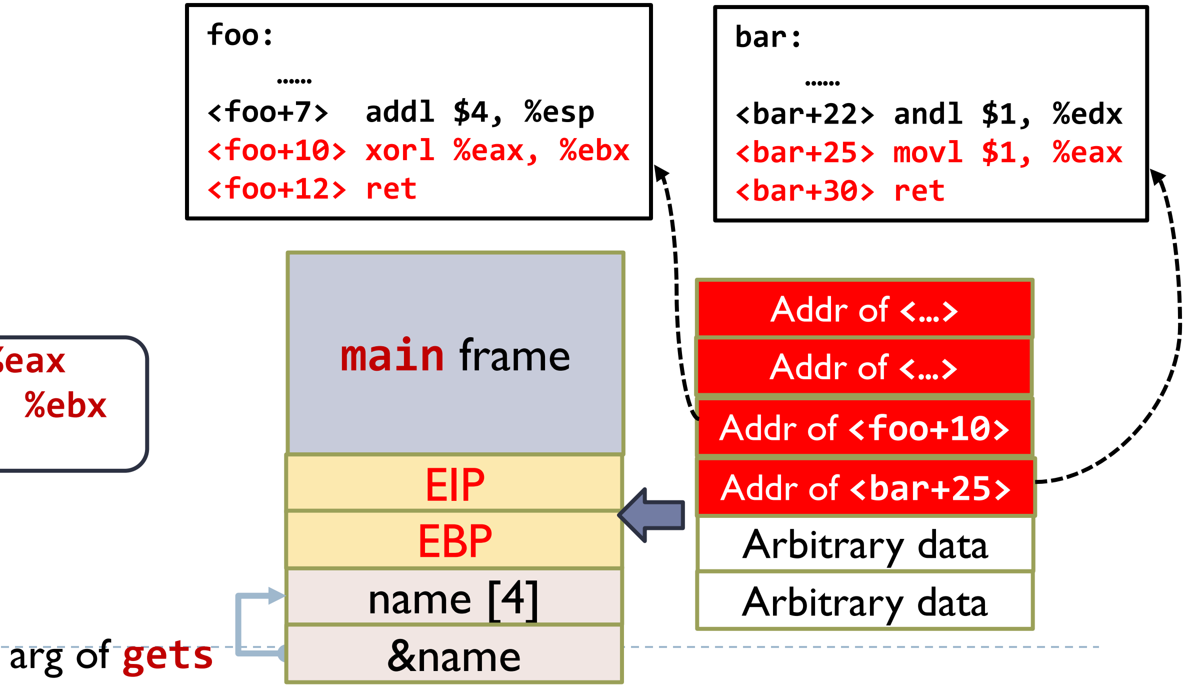
- Construct the malicious code by chaining pieces of existing code (gadget) from different programs.
- Gadget: a small set of assembly instructions that already exist in the system. It usually end with a return instruction (`ret`), which pops the bottom of the stack as the next instruction.

```
int system(char *command);

void vulnerable(void){
    char name [4];
    gets(name);
}

int main(void){
    vulnerable();
    return 0;
}
```

```
movl $1, %eax
xorl %eax, %ebx
......
```



```
foo:
    ......
<foo+7>  addl $4, %esp
<foo+10> xorl %eax, %ebx
<foo+12> ret
```

```
bar:
    ......
<bar+22> andl $1, %edx
<bar+25> movl $1, %eax
<bar+30> ret
```

Limitations of Non-Executable Memory

Two types of executing programs

- Compile a program to the binary code, and then execute it on a machine (C, C++)
- Use an interpreter to interpret the source code and then execute it (Python)

Just-in-Time (JIT) compilation

- Compile heavily-used ("hot") parts of the program (e.g., methods being executed several times), while interpret the rest parts.
- Exploit runtime profiling to perform more targeted optimizations than compilers targeting native code directly

This requires executable heap

- Conflict with the Non-executable Memory protection