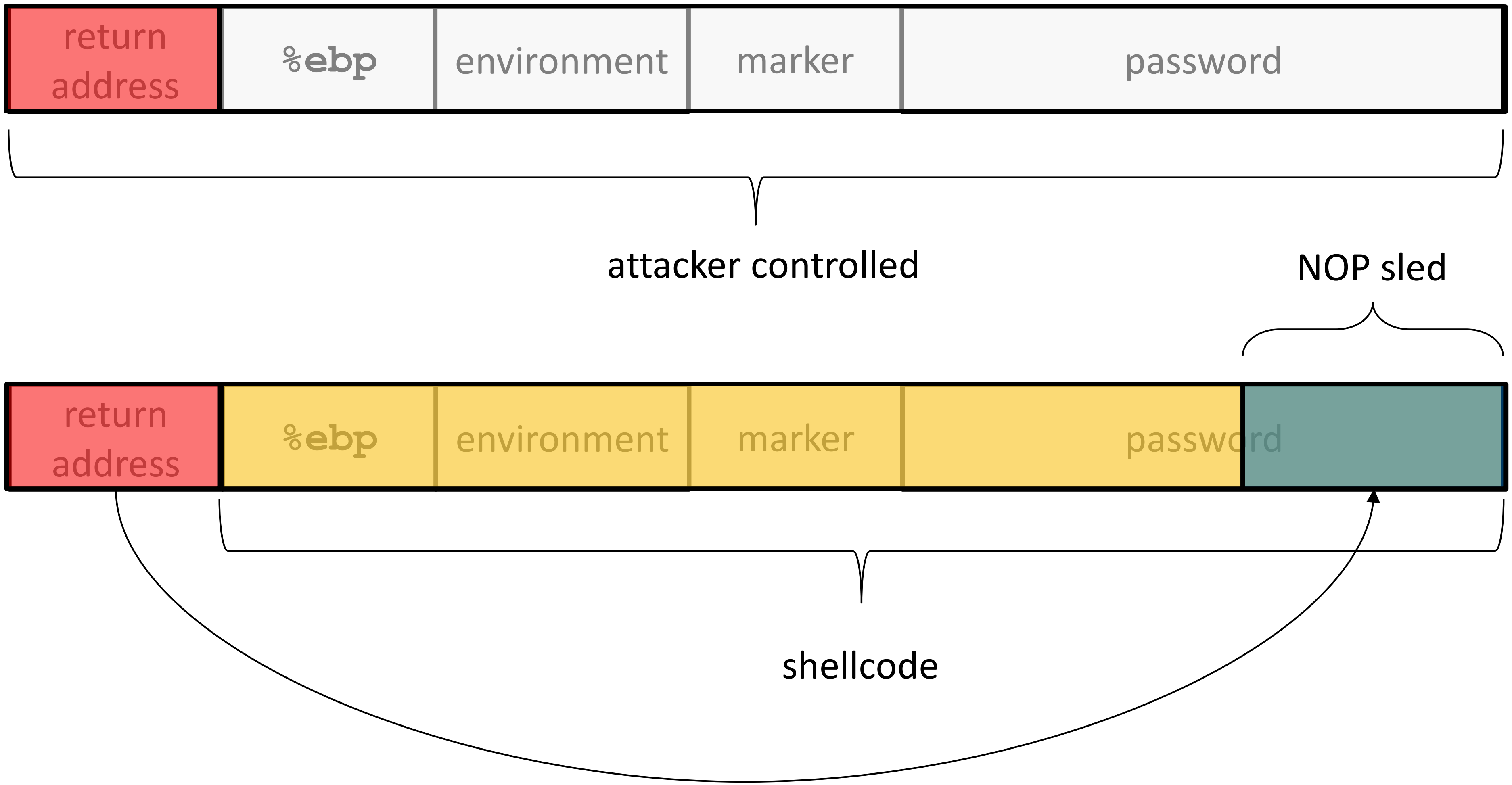
CS326 – Systems Security

Lecture 14

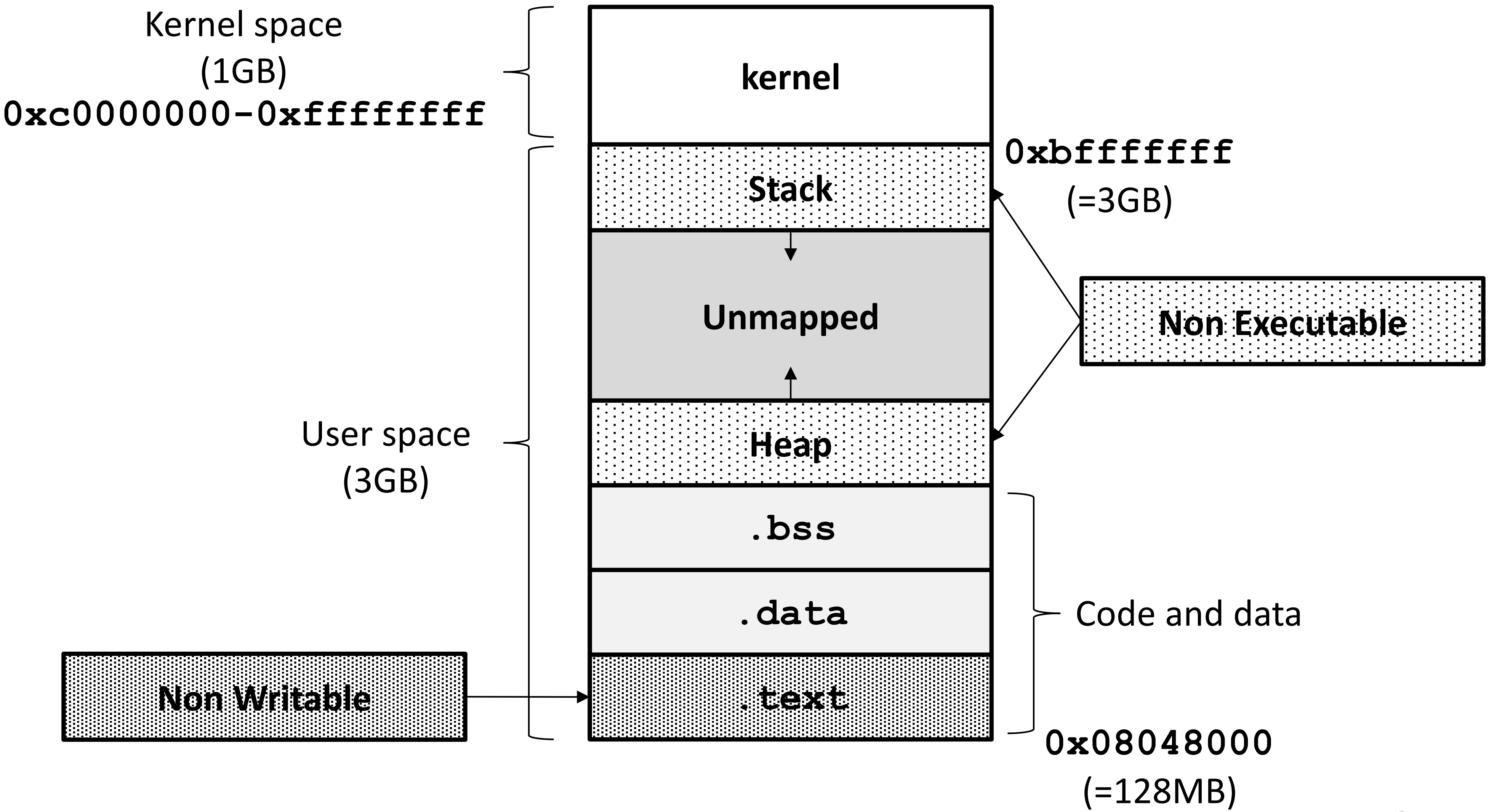
Return-oriented Programming (ROP)

Elias Athanasopoulos
athanasopoulos.elias@ucy.ac.cy

Code Injection

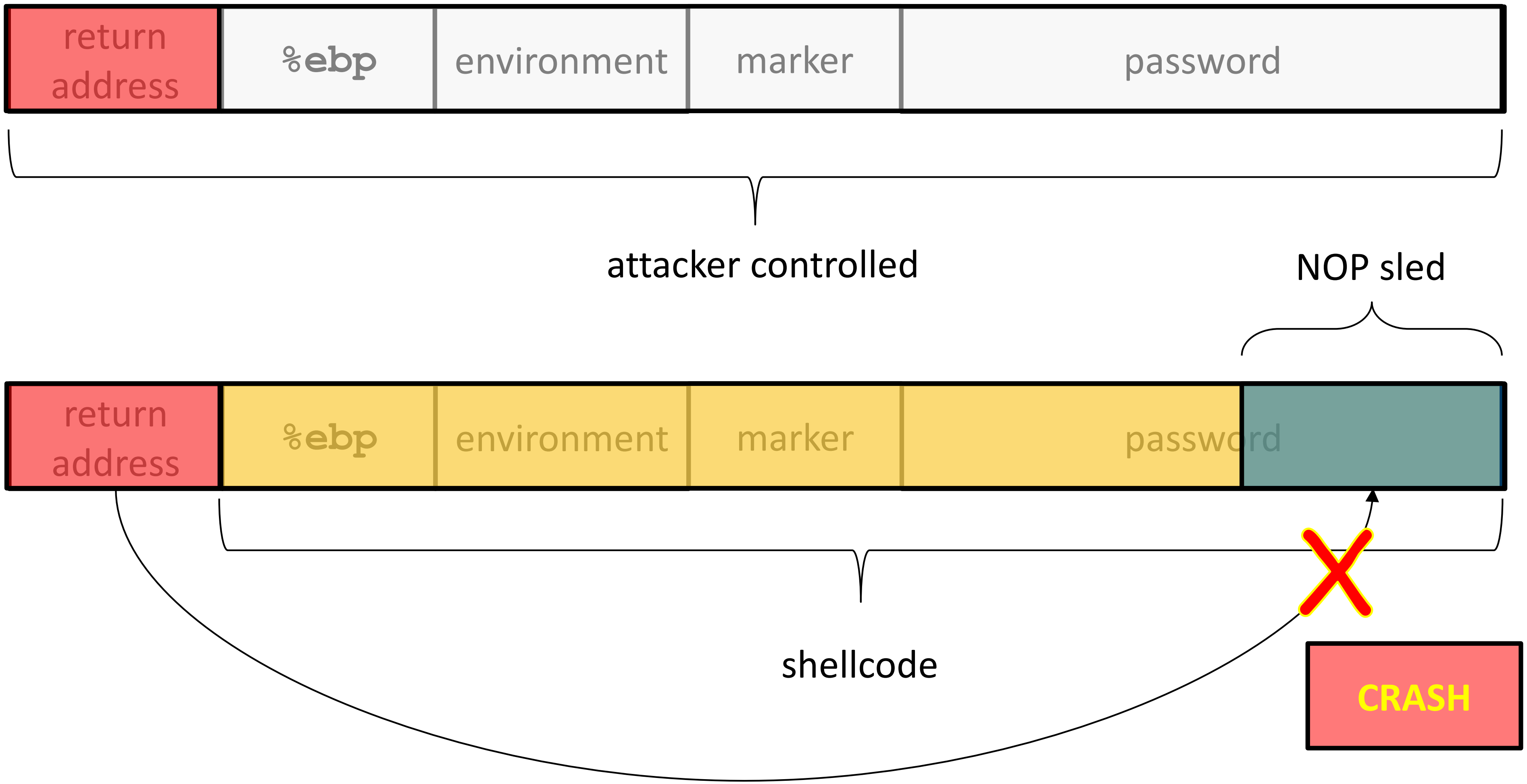


Process Memory Layout
32-bit, Linux (W^X)



Code Injection (W^X)

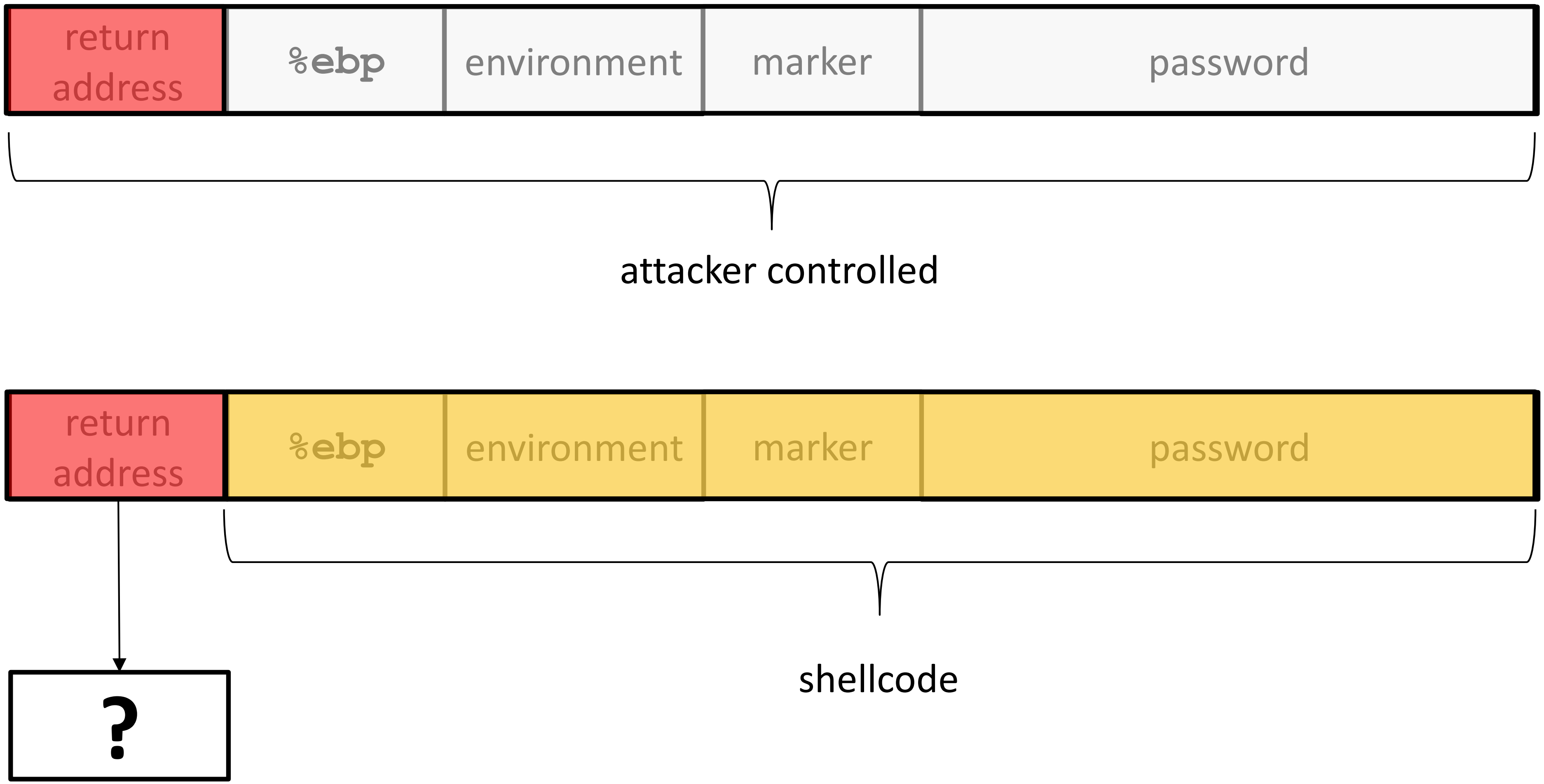
When we did the code injection the program was compiled with `-z execstack`

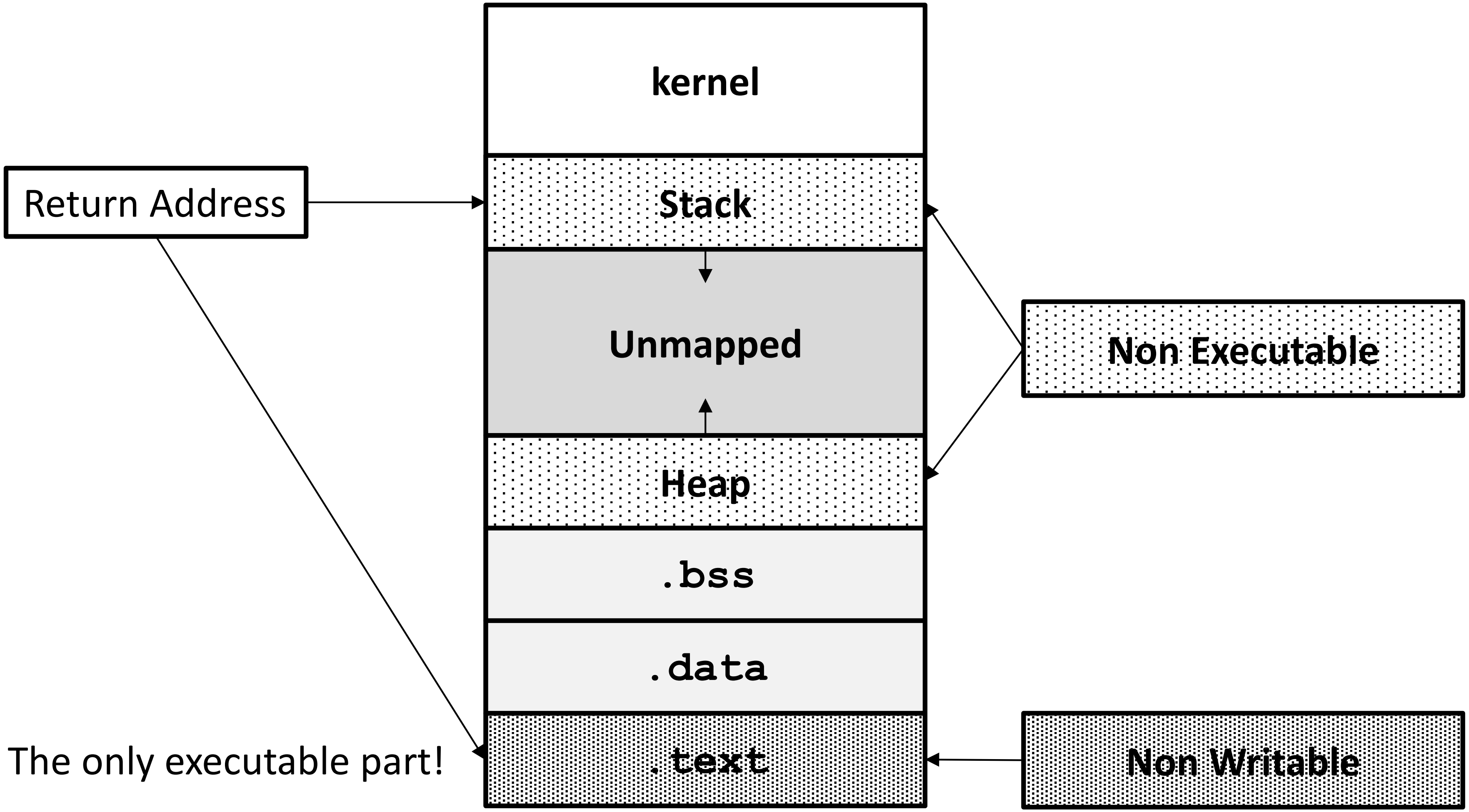


Non executable memory

- Several names
  - NX-bit (Non-Executable Bit)
  - DEP (Data Execution Prevention)
  - W^X (Write XOR Execute)
- Enforced in hardware (MMU)
- Memory pages cannot be executable and writable
  - Stack and heap are writable but not executable
  - Code is executable but not writable
- Permissions can change using system calls
  - `mprotect()` for Linux
  - `VirtualProtect()` for Windows

Where should we jump?




kernel
Return Address
Stack
Unmapped
Non Executable
Heap
.bss
.data
The only executable part!
.text
Non Writable

Return-Oriented Programming

Stack

esp	0xb8800000
	0x00000001
	0xb8800010
	0x00000002
	0xb8800020
	0xb8800010
	0x00400000
	0xb8800030

Code

```
0xb8800000:
  pop eax
  ret
...
0xb8800010:
  pop ebx
  ret
...
0xb8800020:
  add eax, ebx
  ret
...
0xb8800030:
  mov [ebx], eax
  ret
```

Actions

```
eax = 1
ebx = 2
eax += ebx
ebx = 0x400000
*ebx = eax
```

Slide taken from http://www.ieee-security.org/TC/SP2012/slides/Smashing%20the%20Gadgets.pptx

Code Reuse

- Programs include large parts of existing code
  - Available during exploitation
- Small sequences of instructions ending with a `ret`
  - They called gadgets
  - They execute some code and *return* (i.e., read the stack for the next gadget)
  - Stack is attacker-controlled (use `esp` as the program counter)
- Chaining several gadgets
  - Return-Oriented Programming (ROP)
  - Turing complete
  - In practice used to make a region executable

Gadgets

- We saw gadgets like
  - `add eax, ebx; ret`
  - `mov [ebx], eax; ret`
  - `pop eax; ret`
  - `...`
- Is this code realistic?

Intel and the CISC architecture

- Dense instruction set
  - All values map to a valid instruction
- Non-aligned instructions
- Variable-length instructions
  - Jumping in the middle of an instruction generates new instructions

Defending ROP

- ROP is based on exact knowledge of the code layout
  - Code addresses are the beginning of the ROP gadgets
- Randomization
  - Address Space Layout Randomization (ASLR) (`/proc/sys/kernel/randomize_va_space`)
  - Fine-grained
- Position Independent Code

Information Leaks

- Bugs that let you read the process layout
  - So far, overflows were used to overwrite control data
- ASLR
  - Revealing the address where a shared library is mapped is enough
  - All gadgets are just moved to a new offset
- Stack canaries
  - Revealing the contents of the stack is enough
  - The canary is stored in the stack

Information Leaks

- Can be used repeatedly to uncover the layout of the process
- Web browsers
  - Bugs can be abused by JavaScript
  - A malicious JavaScript can use an information-leak bug programmatically
- Suggested Read
  - http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2012-1876
  - Kevin Z. Snow et al. **Just-In-Time Code Reuse: On the Effectiveness of Fine-Grained Address Space Layout Randomization.** In *Proceedings of the 2013 IEEE Symposium on Security and Privacy* (SP '13).

Defenses

- Non-executable pages (NX-bit, DEP, W^X)
  - Stack and heap are not executable
  - Input data cannot be executed
  - Code injection is not possible
  - **Bypass**: ROP
- Randomization (ASLR, PIEs)
  - Code space is randomized
  - Addresses of ROP gadgets are not known
  - **Bypass**: Information Leaks
- Stack canaries
  - Return address cannot be overwritten using a linear overflow
  - **Bypass**: Information Leaks, use *forward edge* (overwrite function pointers, VTable pointers)

Software Exploitation Current Threat Model

- Arbitrary Read Primitive
  - The attacker has all bugs that can help them read the process' memory arbitrarily
- Arbitrary Write Primitive
  - The attacker has all bugs that cab help them write the process memory arbitrarily
- Defenses are in place
  - Non-executable pages, ASLR, Stack Canaries

Suggested Reading

http://10kstudents.eu/material/#syssec_10k_countermeasures