Modern Exploitation Addendum

CS-576 Systems Security

Instructor: Georgios Portokalidis

Fall 2018

Recap: Broadly Deployed Security Mechanisms

NX-bit → Prevent arbitrary code execution

Stack canaries → Detect and prevent stack overflows

ASLR → Introduce uncertainty on the location of injected shellcode and existing code in a running program

Attacker Response

NX-bit → Code-reuse (for example, ROP)

Stack canaries → Focus on and exploit heap overflows

ASLR → Find and exploit information leak bug to reveal layout

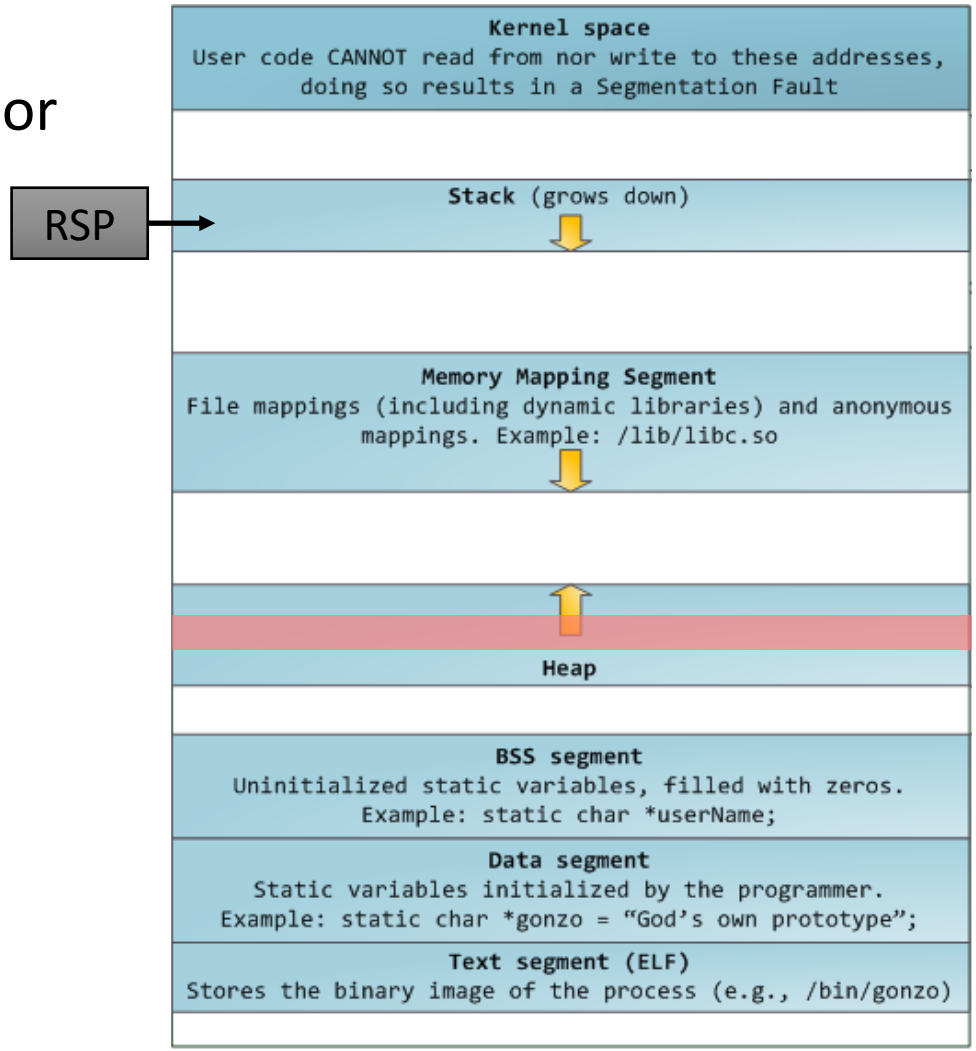
Heap to Stack

Attacker controls:

- the outcome of a call * or jmp *
  - E.g., by overwriting a function pointer in the heap
- An area in the heap

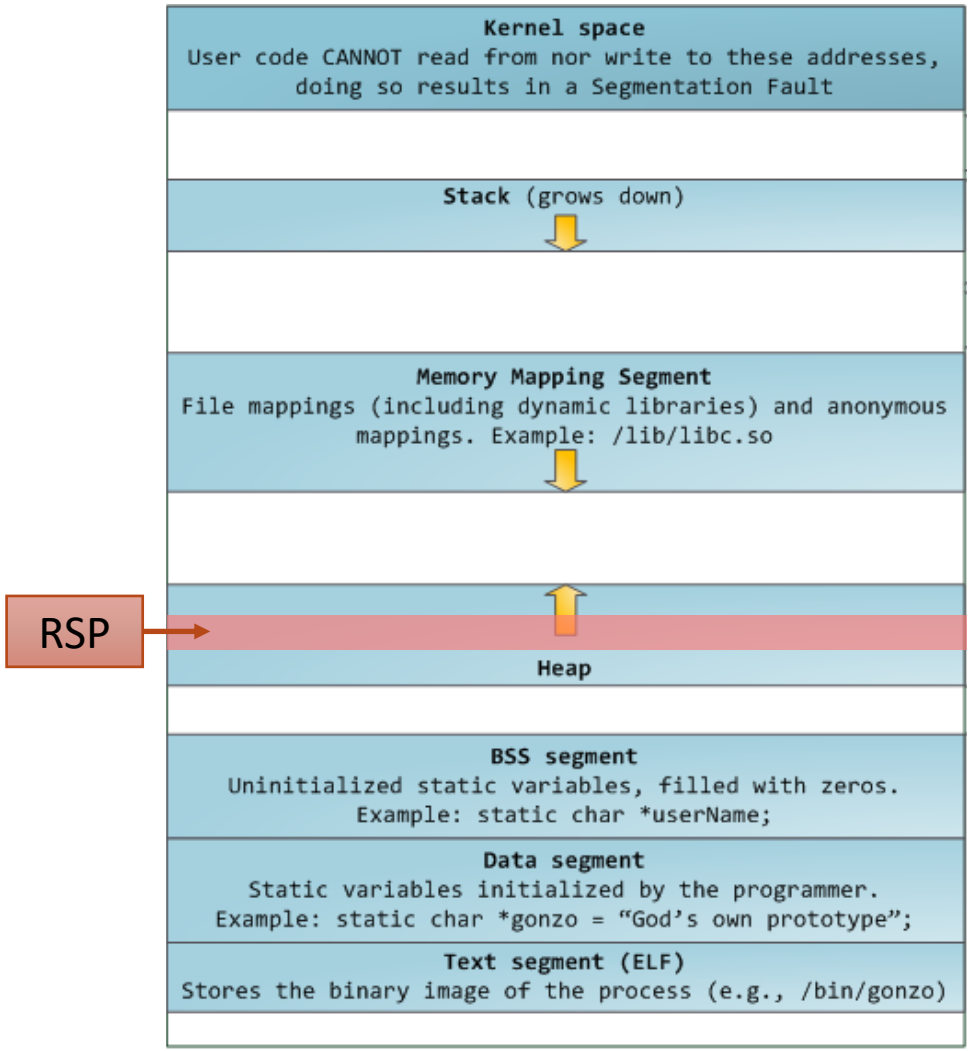
ROP requires controlling the data under RSP

??



Enter Stack Pivoting

Make the stack pointer point to user data



Enter Stack Pivoting

Solution 1

Requirements:

- A register points to the controlled buffer on the heap
- An exchange gadget with esp and that register exists

How:

- Execute the gadget

```
xchg r**, rsp
...
ret
```

Enter Stack Pivoting

Solution 2

Requirements:

- A gadget that adds/subs a large value from the stack pointer
- The result of the above points the SP to user controlled data

How:

- Execute the gadget

```
add 0x***, rsp
...
ret
```

```
sub 0x***, rsp
...
ret
```

Enter Stack Pivoting

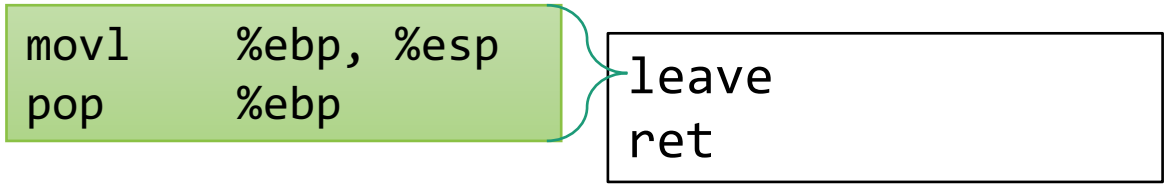
Solution 3

Requirements:

- You control RBP
- A leave gadget exists

How:

- Execute the gadget



More Stack Pivoting

Combining multiple pivots is possible

- For example, executing a sub rsp, 0x**** gadget in a loop

Any instruction sequence that updates the RSP with user-controlled data will do

Example:

```
push rax
pop rsp
...
ret
```

Defenses

Check that RSP is pointing into the stack area

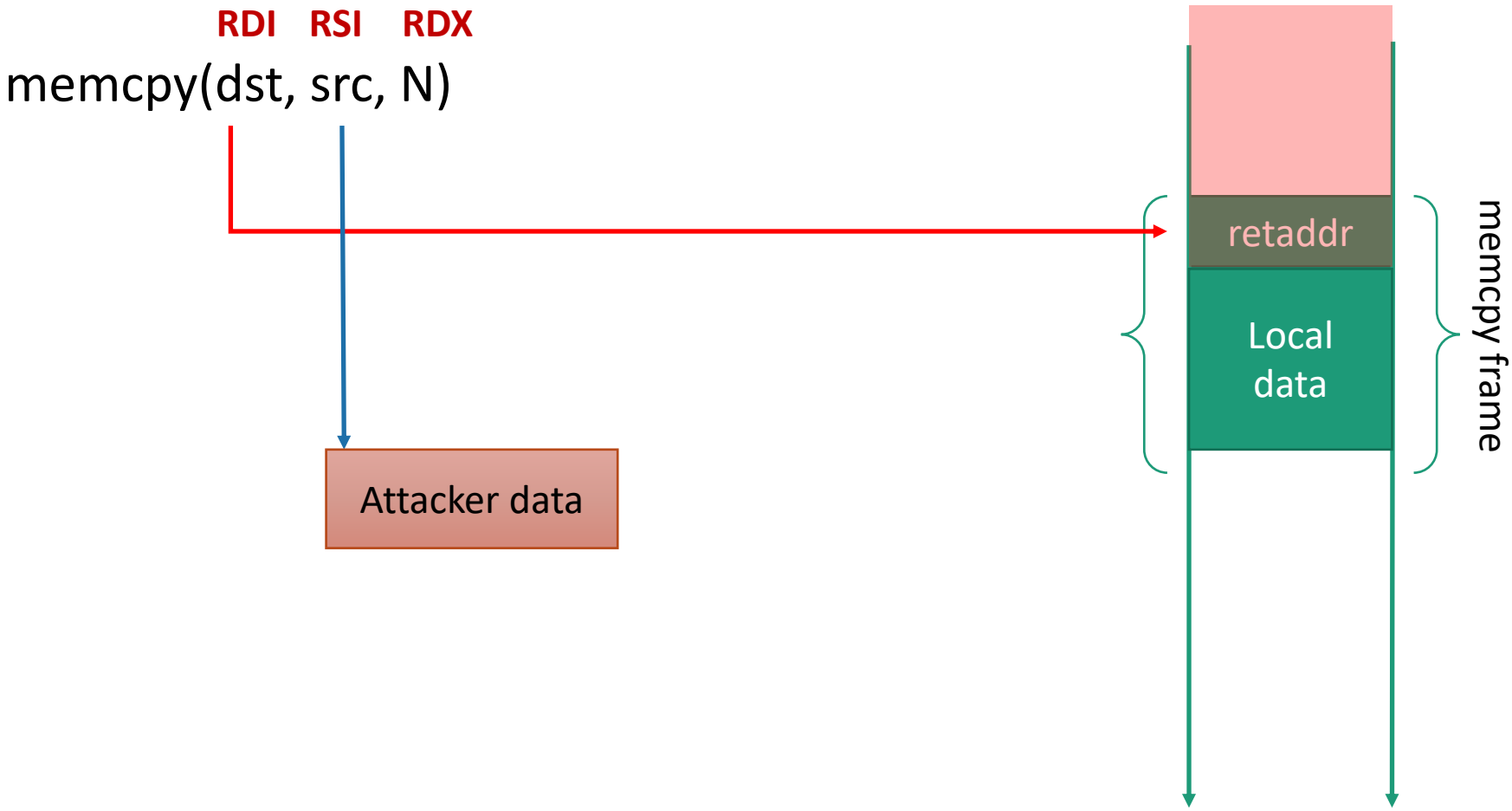
- Potentially expensive (how often should I check the RSP?)
- Can be potentially subverted (where are the stack boundaries saved?)

Actually Moving to the Stack

Find a gadget that copies your buffer into the stack

- For example, find a gadget that calls memcpy()

Memcpy()


RDI RSI RDX
memcpy(dst, src, N)
Attacker data
retaddr
Local data
memcpy frame