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 $\rightarrow$ Code-reuse (for example, ROP)

Stack canaries $\rightarrow$ Focus on and exploit heap overflows

ASLR $\rightarrow$ 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

??

Kernel space
User code CANNOT read from nor write to these addresses, doing so results in a Segmentation Fault

Stack (grows down)

Memory Mapping Segment
File mappings (including dynamic libraries) and anonymous mappings. Example: /lib/libc.so

Heap

BSS segment
Uninitialized static variables, filled with zeros. Example: static char *userName;

Data segment
Static variables initialized by the programmer. Example: static char *gonzo = "God’s own prototype";

Text segment (ELF)
Stores the binary image of the process (e.g., /bin/gonzo)
Enter Stack Pivoting

Make the stack pointer point to user data

- **Kernel space**
  - User code CANNOT read from nor write to these addresses, doing so results in a Segmentation Fault

- **Stack (grows down)**

- **Memory Mapping Segment**
  - File mappings (including dynamic libraries) and anonymous mappings. Example: /lib/libc.so

- **RSP**

- **Heap**

- **BSS segment**
  - Uninitialized static variables, filled with zeros. Example: static char *userName;

- **Data segment**
  - Static variables initialized by the programmer. Example: static char *gonzo = "God's own prototype";

- **Text segment (ELF)**
  - Stores the binary image of the process (e.g., /bin/gonzo)
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

```
movl  %ebp, %esp
pop   %ebp
leave
ret
```
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:

```assembly
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()

memcpy(dst, src, N)

RDI  RSI  RDX

Attacker data

Local data

retaddr

Memcpy frame