

CS356: Return Fun

W. Michael Petullo

University of Wisconsin–La Crosse

As of September 29, 2021



Exploitation: AMD64 Call Stack

```
void foo(long c1, long c2, long c3, long c4,
         long c5, long c6, long c7) {
    long c8 = 8;
    bar(c7, c8); ← RIP
}

foo(1, 2, 3, 4, 5, 6, 7);
```

RDI: 1 RSI: 2 RDX: 3 RCX: 4 R8: 5 R9: 6

RSP	Low addr.	0x0000000000000000	Padding
		0x0000000000000008	Local c8
		0x00007fffffff00000000	Saved RBP
		0x0000000000004011bd	Return addr.
	High addr.	0x0000000000000007	Seventh arg.
		...	

- ▶ Stack grows from high to low addresses (down)
- ▶ First six args. in registers RDI, RSI, RDX, RCX, R8, and R9
- ▶ Subsequent args. on stack
- ▶ Return value on stack
- ▶ Local variables on stack (i.e., array bytes go towards return address)



Review Aquinas' smash



Exploitation: service-smash.c's authenticate

```
char buf[8];  
...  
for (;;) {  
    c = getchar();  
    if (EOF == c) { break; }  
    buf[i++] = c;  
    if ('\n' == c) { break; }  
}  
... ← RIP, after user entered "1234567"
```

RSP	0x7fffffff d1e0	0x0a37363534333231	buf bytes 7-0
	0x7fffffff d1e8	0x000000002322b8ea	(16 byte align)
RBP	0x7fffffff d1f0	0x00007fffffff d200	Saved RBP
	0x7fffffff d1f8	0x0000000000400363	Return addr.
		...	



Review Aquinas' shellcode



Exploitation: Injecting Arbitrary Code, 1990s style

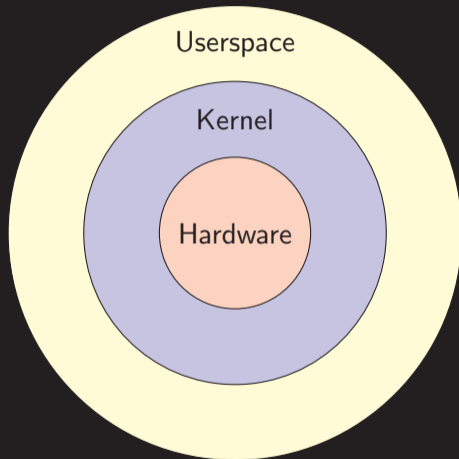
What if the program you attack does not have a `reveal` function? What if you want to execute code not already in the program?

- 1 Identify the address of `buffer` (sometimes this takes another bug).
- 2 Write instructions into `buffer` before overwriting return address.
- 3 Set return address to return into `buffer`.

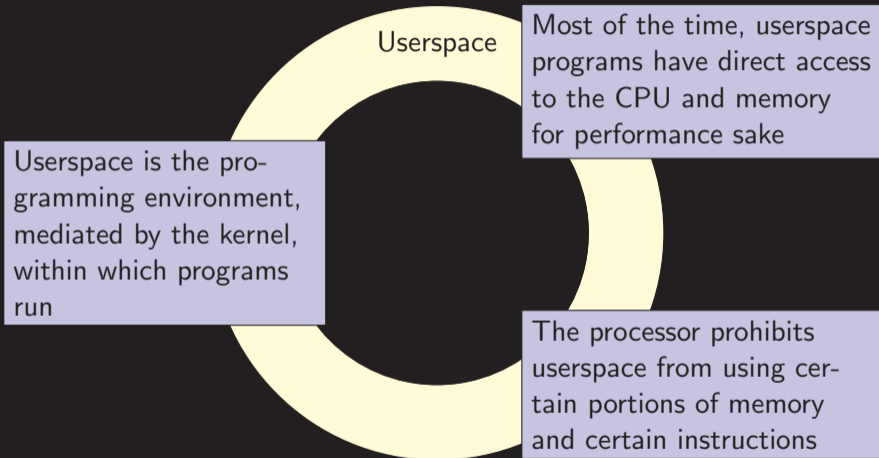
RSP	0x7fffffff d1e0	0xf7e65048bf2f6269	buf bytes 7-0
	0x7fffffff d1e8	0x6e2f2f7368574889	(16 byte align)
RBP	0x7fffffff d1f0	0xe7b03b0f05000000	Saved RBP
	0x7fffffff d1f8	0x00007fffffff d1e0	Return addr.
		...	

The instructions written are referred to as `shellcode` because a common aim is to execute a shell.

Return Fun: Just Enough OSES

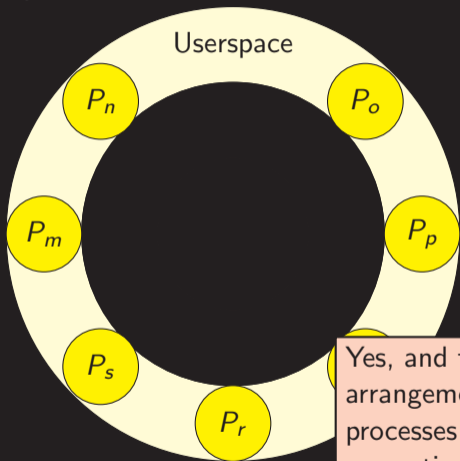


Return Fun: Just Enough OSes



Return Fun: Just Enough OSes

So easy even a Ph.D. can do it!

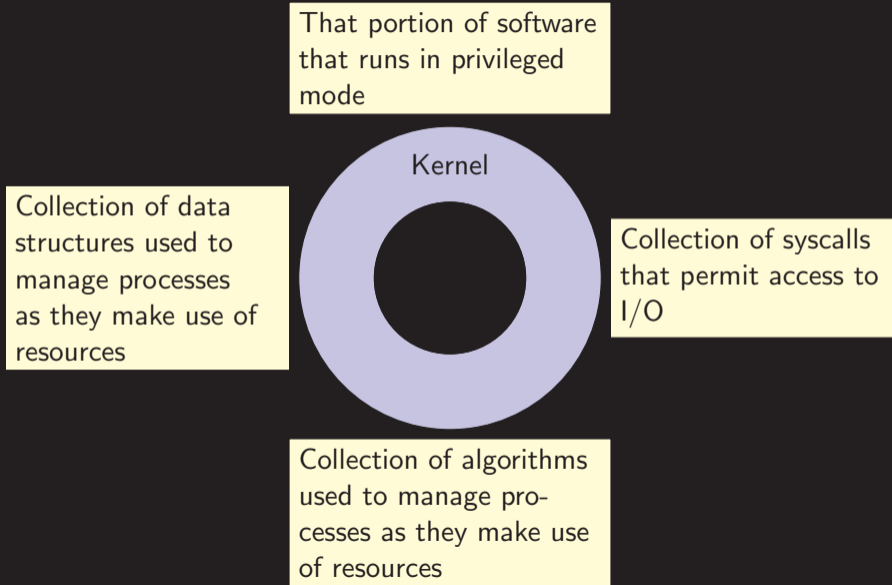


So, each process appears to have its own address space and direct access to most instructions? Wow!

Yes, and the arrangement prevents processes from corrupting each other or the system.



Return Fun: Just Enough OSEs



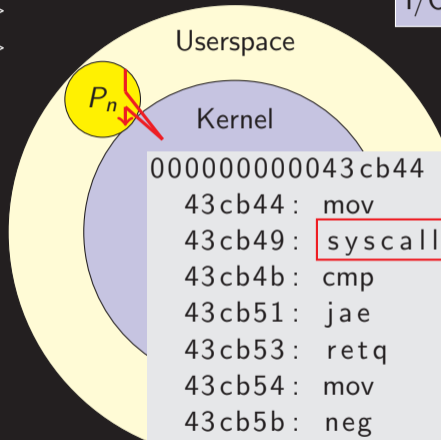
Return Fun: Just Enough OSEs



Collection of syscalls
that permit access to
I/O

```
#include <stdlib.h>
#include <unistd.h>
```

```
int main(void)
{
    alarm(1);
}
```



```
000000000043cb44 <alarm>:
43cb44: mov     $0x25,%eax
43cb49: syscall
43cb4b: cmp     $0xff...ff001,%rax
43cb51: jae    43cb54 <alarm+0x14>
43cb53: retq
43cb54: mov     $0xff...ffc0,%rcx
43cb5b: neg     %eax
43cb5d: mov     %eax,%fs:(%rcx)
43cb60: or     $0xff...ff,%rax
43cb64: retq
```

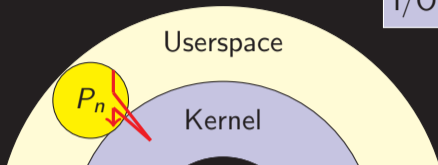
Return Fun: Just Enough OSEs



Collection of syscalls that permit access to I/O

```
#include <stdlib.h>
#include <unistd.h>
```

```
int main(void)
{
```



*Intel 64 and ia-32 Architectures
Software developer's Manual:
Volume 2. 2016. p. 4-668*

SYSCALL—Fast System Call

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
OF 05	SYSCALL	NP	Valid	Invalid	Fast call to privilege level 0 system procedures.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
NP	NA	NA	NA	NA

Description

SYSCALL invokes an OS system-call handler at privilege level 0. It does so by loading RIP from the IA32_LSTAR MSR (after saving the address of the instruction following SYSCALL into RCX). (The WRMSR instruction ensures

Return Fun: Just Enough OSes



- 1 Prepare arguments per convention
- 2 Set system call identifier (`/usr/include/asm/unistd_64.h`)
- 3 Invoke `syscall` instruction
 - Processor stores PC
 - Processor loads `syscall` handler into PC
 - Execution context set to privileged
 - Continue fetch-execute cycle:
- 4 Save additional process state
- 5 Check parameters (e.g., is `buf [0-- n]` legal?)
- 6 Check permissions (e.g., may process access file descriptor n ?)
- 7 Satisfy system call
- 8 Restore process state and set return value per convention
- 9 Return to unprivileged context and restore PC





Fragment	Meaning
hello:	Create a string with contents
.string "A string\n"	"A string\n" and label with hello.
.globl _start	Declare the entry point label _start
_start:	as global, and define (like C's main).
mov \$1, %rax	Move the literal value 1 (write) into register rax.
mov \$1, %rdi	Move the literal value 1 (stdout) into register rdi.
mov \$hello, %rsi	Move the address of the string hello into register rsi.
mov \$9, %rdx	Move length of hello into register rcx.
syscall	Invoke the system call instruction.

To compile example.S:

```
gcc -s -nostartfiles -nostdlib -o example example.S
```

Return Fun: Intel Manuals



Intel® 64 and IA-32 Architectures Software Developer's Manual

Volume 1:
Basic Architecture

NOTE: The Intel® 64 and IA-32 Architectures Software Developer's Manual consists of ten volumes:
Basic Architecture, Order Number 253685; Instruction Set Reference, A-Z, Order Number 253690;
Instruction Set Reference, M16, Order Number 253687; Instruction Set Reference, IA-32, Order Number
253692; Instruction Set Reference, IA-64, Order Number 253689; System Programming Guide, Part 1, Order
Number 253686; System Programming Guide, Part 2, Order Number 253693; System Programming
Guide, Part 3, Order Number 253679; System Programming Guide, Part 4, Order Number 253677;
Model-Specific Registers, Order Number 253695. Refer to all ten volumes when evaluating your design
needs.

Order Number: 253685-079US
June 2011



Intel® 64 and IA-32 Architectures Software Developer's Manual

Volume 2 (2A, 2B, 2C & 2D):
Instruction Set Reference, A-Z

NOTE: The Intel 64 and IA-32 Architectures Software Developer's Manual consists of four volumes:
Basic Architecture, Order Number 253685; Instruction Set Reference, A-Z, Order Number 253690;
System Programming Guide, Order Number 253696; Model-Specific Registers, Order Number
253695. Refer to all four volumes when evaluating your design needs.

Order Number: 253690-079US
June 2011



Intel® 64 and IA-32 Architectures Software Developer's Manual

Volume 3 (3A, 3B, 3C & 3D):
System Programming Guide

NOTE: The Intel 64 and IA-32 Architectures Software Developer's Manual consists of four volumes:
Basic Architecture, Order Number 253685; Instruction Set Reference, A-Z, Order Number 253690;
System Programming Guide, Order Number 253696; Model-Specific Registers, Order Number 253695.
Refer to all four volumes when evaluating your design needs.

Order Number: 253696-079US
June 2011



Intel® 64 and IA-32 Architectures Software Developer's Manual

Volume 4:
Model-Specific Registers

NOTE: The Intel® 64 and IA-32 Architectures Software Developer's Manual consists of ten volumes:
Basic Architecture, Order Number 253685; Instruction Set Reference, A-Z, Order Number 253690;
Instruction Set Reference, M16, Order Number 253687; Instruction Set Reference, IA-32, Order Number
253692; Instruction Set Reference, IA-64, Order Number 253689; System Programming Guide, Part 1, Order
Number 253686; System Programming Guide, Part 2, Order Number 253693; System Programming
Guide, Part 3, Order Number 253679; System Programming Guide, Part 4, Order Number 253677;
Model-Specific Registers, Order Number 253695. Refer to all ten volumes when evaluating your design
needs.

Order Number: 253695-079US
June 2011



Return Fun: Shellcode Demonstration

Cause `service-shellcode` (sanitized) to terminate with a exit code of 42.

Important: GDB disables a feature called ASLR to aid in reproducibility. Your “live” code will need to make use of the leaked buffer and bp.

- 1 Determine stack layout using `gdb`.
- 2 Write shellcode in assembly (examples and `/usr/include/asm/unistd_64.h`).
- 3 Compile with `gcc -nostartfiles -nostdlib -o shellcode shellcode.S`, and view with `objdump`; gather shellcode bytes.
- 4 Run `./shellcode` to test.
- 5 Build: `shellcode | spacer | bp | &buffer` with `echo "\x48..." >exploit`

Important: your shellcode must make use of a string (input file path). Refer to the project instructions for why this causes a problem and to find a solution.



Graded Homework Aquinas: smash, syscall, and shellcode

Reading Read 0x321–0x359

<https://www.flyn.org/courses/cs356/schedule>