Class

7

CSC 472 Software Security

Shellcode: From Assembly to Bytes

Dr. Si Chen (schen@wcupa.edu)

Tue / Thu · 2:00–3:15 PM · 25 University Ave., Room 158
Fall 2026 · Week 4 · Thursday, September 17, 2026
WolfCTF + GDB / GEF · wcu.ninja/csc472

One program, two ways to execute it

Start from Tuesday

A Linux i386 write syscall prints a message.

Remove the ELF loader's help

Make code and data work after copying the bytes to a new address.

Test what we built

Extract .text, inspect the bytes, and step through them in GEF.

Explain the execution conditions

Architecture, data addresses, memory permissions, and entry point.

What do we mean by shellcode?

Machine-code bytes with a small purpose

Historically, launching a shell; today's first payload just prints hello.

A compact execution context

The bytes may run without an ordinary ELF loader arranging their data.

An environment-specific program

Instruction set, ABI, permissions, and entry conditions must all match.

Our experiment

Run the bytes deliberately inside our own WolfCTF process.

An ELF file is more than its instructions

Linked executable	Raw payload used here
ELF headers describe program mappings	Only the extracted .text bytes are copied
Linker resolves data addresses	The bytes must find their own data
Entry is described by the executable	Our runner calls the first copied byte
Separate .text and .data can work	Needed bytes must be present at runtime

Copying instructions is not enough if they still refer to an old address.

The same hello.asm from ss2024

hello.asm — all bytes in .text

```
_start:
    jmp short ender
starter:
    xor eax, eax
    xor ebx, ebx
    xor edx, edx
    xor ecx, ecx
    mov al, 4
    mov bl, 1
    pop ecx
    mov dl, 5
    int 0x80
```

Finish and inline data

```
    xor eax, eax
    mov al, 1
    xor ebx, ebx
    int 0x80
ender:
    call starter
    db 'hello'
```

The call pushes the address of the next byte: h in hello.

Jump–call–pop: find the data at runtime

Step	Control flow / stack effect
1 · jmp ender	Skip the body and reach the call instruction.
2 · call starter	Push the address just after call; jump to starter.
3 · clear and set registers	Prepare write: EAX=4, EBX=1; EDX will be 5.
4 · pop ecx	Recover the runtime address of the inline hello.
5 · int 0x80	write(1, ECX, 5), then exit(0).

ECX is a runtime pointer. No fixed address of hello is encoded in the payload.

Why the copied bytes still work

Relative control flow

The jmp and call displacements describe distances inside the payload.

Runtime data address

call supplies the actual next-instruction address; pop retrieves it.

Self-contained bytes

The instructions and the 5 message bytes are all in .text.

No promise of universal portability

These are Linux i386 instructions and syscall conventions.

Assemble and extract only .text

In ~/csc472-week04/class07

```
make

# What make does for the payload:
nasm -f elf32 -g -F dwarf hello.asm -o hello.o
objcopy -O binary -j .text hello.o hello.bin

objdump -d -M intel hello.o
readelf -r hello.o
```

Check for .text address dependencies. Debug-section relocations are separate.

Check the actual byte sequence

od -An -tx1 hello.bin

```
eb 19 31 c0 31 db 31 d2 31 c9 b0 04 b3 01 59 b2
05 cd 80 31 c0 b0 01 31 db cd 80 e8 e2 ff ff ff
68 65 6c 6c 6f
```

37 total bytes

32 bytes of instructions + 5 bytes of inline message data.

The last five bytes

68 65 6c 6c 6f = h e l l o. They are data for write().

Count bytes, not visible characters in a hex string

Use wc -c hello.bin. The \x notation is just a representation.

Bad bytes depend on the input path

C-string input may stop at 0x00

That motivates many classic NUL-free encodings.

Other inputs have other constraints

A line parser may treat newline specially; an encoder may alter bytes.

Our runner uses an explicit length

memcpy can copy NUL bytes. strlen cannot measure arbitrary code.

Ask before optimizing

Which bytes are forbidden, by what input path, and at which stage?

Update the original test program

ss2024 example	This week's runner
Fixed address: 0x80000000	Let mmap choose a page-aligned address
strlen(shellcode) + 1	Use sizeof an explicit unsigned-byte array
RWX allocation	Copy while RW; switch that mapping to RX
Unchecked mapping / permissions	Check errors before calling the bytes

The teaching idea is the same: copy our bytes into memory, then call the entry.

Runner, part 1: allocate and copy

run_payload.c — excerpt

```
void *region = mmap(NULL, payload_len,
    PROT_READ | PROT_WRITE,
    MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);

if (region == MAP_FAILED) {
    perror("mmap");
    return 1;
}
memcpy(region, payload, payload_len);
```

At this point the new mapping is writable. Its address is not hard-coded.

Runner, part 2: execute with RX permissions

run_payload.c — excerpt

```
if (mprotect(region, payload_len,
             PROT_READ | PROT_EXEC) == -1) {
    perror("mprotect");
    munmap(region, payload_len);
    return 1;
}
ready_for_debug(region);
((void (*)(void))region)();
```

mprotect changes this mapping. It does not globally disable NX / DEP.

Build and run the raw hello payload

The supplied Makefile handles all steps

```
cd ~/csc472-week04/class07
make
file hello_runner
./hello_runner
printf '\n'

# Expected payload output: hello
```

The payload calls exit(0). It does not return to the C runner.

GEF: stop at the copied entry point

gdb -q ./hello_runner

```
gef config context.layout 'regs code stack'
break ready_for_debug
run
set $p = (unsigned char *)region
x/37bx $p
x/12i $p
vmmap
break *$p
continue
```

Use this run's region address. The copied mapping should be readable + executable.

GEF: follow jump–call–pop

At the payload entry breakpoint

```
si          # jmp ender
si          # call starter
x/wx $esp
x/5cb *(unsigned int *)$esp

# Continue stepping until after pop ecx:
si
# ... repeat as needed, then:
x/5cb $ecx
```

Explain both levels: ESP points to a stack word; that word points to hello.

A classic shell payload needs writable data

The historical ss2024 example

An execve shell payload patches its own inline pathname / pointer data.

Why an unchanged copy can fault

The updated runner gives the payload mapping read + execute access.

Separate data from executable bytes

Build the pathname, argv, and envp on the writable stack.

Keep the model precise

Writing stack data does not require executing instructions on the stack.

Optional: build execve data on the stack

execve_stack.asm — main path

```
xor eax, eax
push eax                ; string terminator
push dword 0x68732f2f    ; "//sh"
push dword 0x6e69622f    ; "/bin"
mov ebx, esp            ; pathname
push eax
mov edx, esp            ; envp = { NULL }
push eax
push ebx
mov ecx, esp            ; argv = { pathname, NULL }
mov al, 11
int 0x80
```

./shell_runner → id → exit. Same unprivileged lab account; local container only.

Diagnose by checking one assumption

Symptom	Check first
ELF will not run	file output: did you build Linux i386 in WolfCTF?
Wrong output / length	EDX count; exact extracted bytes; trailing newline
Crash at first instruction	Entry pointer, mapping permissions, architecture
Crash during a store	Did this payload try to modify its RX code mapping?
Wrong string address	Step through call and pop; inspect ESP and ECX

Practice and Lab 1 checkpoint

Practice · change the message

Replace hello with WCU!! (5 bytes), rebuild, and inspect the new bytes.

Explain · position independence

Why do the jump, call, and pop still work at a different mapping address?

Continue · Lab 1 on WolfCTF

Due Thursday, September 24, 2026 · 11:59 PM Eastern Time.

Submit · one PDF report

D2L → CSC 472 → Assignments → Lab 1
Required format: wcu.ninja/csc472/lab-report

References and what comes next

Original course materials

ss2024 Chapter 6: hello.asm, shell.asm, and the shellcode test program.

Current technical references

Linux syscall / mmap / mprotect / execve manuals; NASM and GNU objcopy.

All files

wcu.ninja/csc472/ → Class 7 slides and the Week 4 code bundle.

Next connection

Stack layout + control flow + payload bytes
How can memory corruption change where execution goes?