CSCI 331:
Introduction to Computer Security

Lecture 16: More shellcode

Instructor: Dan Barowy

Williams

Announcements

David Jensen, UMass Amherst

- Class of 60's talk:
  What's So Important About Explanation? Science, Machine Learning, and Large Language Models
  Thu at 7:30pm in Wege Auditorium
- Friday's colloquium:
  Explanation, Causation, and Mechanism in AI systems
  Fri at 2:35pm in Wege Auditorium

Topics

Writing assembly programs

Removing NULL bytes

Your to-dos

1. Reading *Preventing Privilege Escalation* for **Thu 11/9**.
2. Lab 7, **due Sunday 11/19**.

Assembly programming

As usual, let's start with **"Hello world!"**

```c
#include <stdio.h>

int main() {
  printf("Hello world!\n");
  return 0;
}
```

How do we write the **equivalent** in assembly?

Let's use a **C program as inspiration**.

Assembly programming

```
$ gcc -S helloworld.c
```

```
 .arch armv6
 .eabi_attribute 28, 1
 .eabi_attribute 20, 1
 .eabi_attribute 21, 1
 .eabi_attribute 23, 3
 .eabi_attribute 24, 1
 .eabi_attribute 25, 1
 .eabi_attribute 26, 2
 .eabi_attribute 30, 6
 .eabi_attribute 34, 1
 .eabi_attribute 18, 4
 .file  "helloworld.c"
 .text
 .section  .rodata
 .align 2
.LC0:
 .ascii "Hello world!\000"
 .text
 .align 2
 .global   main
 .arch armv6
 .syntax unified
 .arm
 .fpu vfp
 .type  main, %function
main:
 @ args = 0, pretend = 0, frame = 0
 @ frame_needed = 1, uses_anonymous_args = 0
 push   {fp, lr}
 add fp, sp, #4
 ldr r0, .L3
 bl  puts
 mov r3, #0
 mov r0, r3
 pop {fp, pc}
.L4:
 .align 2
.L3:
 .word  .LC0
 .size  main, .-main
 .ident "GCC: (Raspbian 8.3.0-6+rpi1) 8.3.0"
 .section  .note.GNU-stack,"",%progbits
```

Eek!

What's really necessary?

This can all be removed

```
~~ .arch armv6~~
~~ .eabi_attribute 28, 1~~
~~ .eabi_attribute 20, 1~~
~~ .eabi_attribute 21, 1~~
~~ .eabi_attribute 23, 3~~
~~ .eabi_attribute 24, 1~~
~~ .eabi_attribute 25, 1~~
~~ .eabi_attribute 26, 2~~
~~ .eabi_attribute 30, 6~~
~~ .eabi_attribute 34, 1~~
~~ .eabi_attribute 18, 4~~
~~ .file  "helloworld.c"~~
~~ .text~~
~~ .section  .rodata~~
~~ .align 2~~
.LC0:
 .ascii "Hello world!\000"
~~ .text~~
 .align 2
 .global   main
~~ .arch armv6~~
~~ .syntax unified~~
~~ .arm~~
~~ .fpu vfp~~
~~ .type  main, %function~~
main:
~~ @ args = 0, pretend = 0, frame = 0~~
~~ @ frame_needed = 1, uses_anonymous_args = 0~~
 push   {fp, lr}
 add fp, sp, #4
 ldr r0, .L3
 bl  puts
 mov r3, #0
 mov r0, r3
 pop {fp, pc}
~~.L4:~~
~~ .align 2~~
.L3:
 .word  .LC0
~~ .size  main, .-main~~
~~ .ident "GCC: (Raspbian 8.3.0-6+rpi1) 8.3.0"~~
~~ .section  .note.GNU-stack,"",%progbits~~
```

Why?

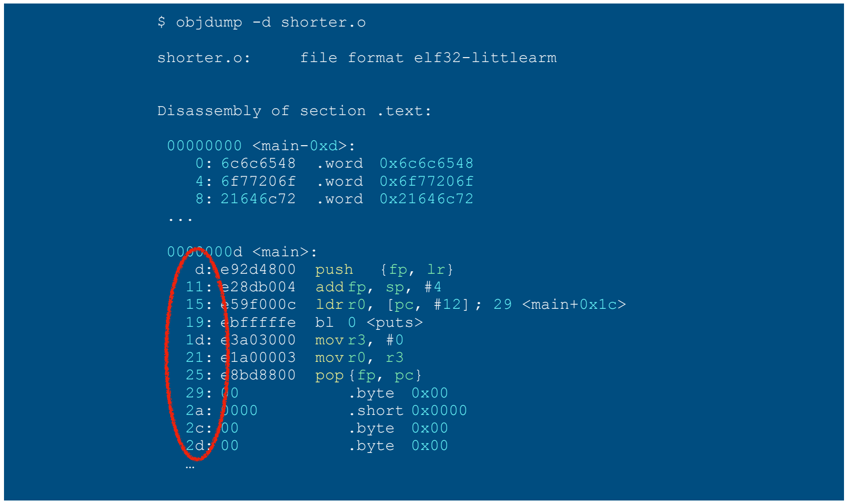
Much better

```
.LC0:
 .ascii "Hello world!\000"
 .align 2
 .global   main
main:
 push   {fp, lr}
 add fp, sp, #4
 ldr r0, .L3
 bl  puts
 mov r3, #0
 mov r0, r3
 pop {fp, pc}
.L3:
 .word  .LC0
```

Can we make this shorter?

Can we remove `.align 2`?   Not directly.

Can you spot the problem?



```
$ objdump -d shorter.o

shorter.o:     file format elf32-littlearm


Disassembly of section .text:

 00000000 <main-0xd>:
    0: 6c6c6548  .word  0x6c6c6548
    4: 6f77206f  .word  0x6f77206f
    8: 21646c72  .word  0x21646c72
 ...

 0000000d <main>:
    d: e92d4800  push   {fp, lr}
   11: e28db004  add fp, sp, #4
   15: e59f000c  ldr r0, [pc, #12] ; 29 <main+0x1c>
   19: ebfffffe  bl  0 <puts>
   1d: e3a03000  mov r3, #0
   21: e1a00003  mov r0, r3
   25: e8bd8800  pop {fp, pc}
   29: 00            .byte  0x00
   2a: 0000          .short 0x0000
   2c: 00            .byte  0x00
   2d: 00            .byte  0x00
   …
```

ARM instructions *must* be 4-byte aligned.

`ldr` vs `adr`

Pointers are supported in hardware!

Meaning	C	ARM
address of x	`&x`	`adr r7, x`
dereference x	`*x`	`ldr r7, x`

(variable names and register numbers chosen arbitrarily)

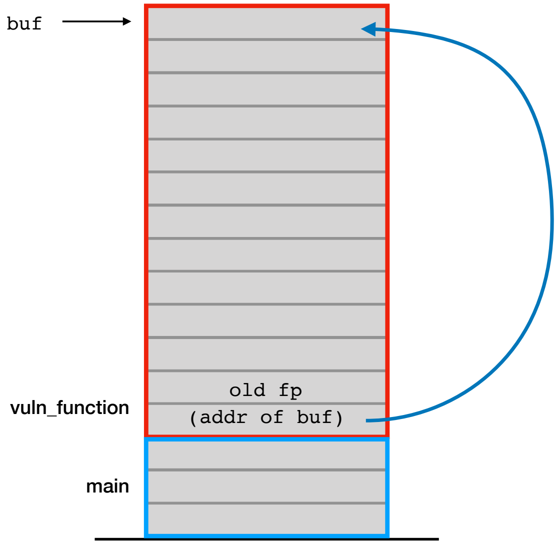
A nice, short program

```
 .global  main
main:
 push {fp, lr}
 add  fp, sp, #4
 adr  r0, hello
 bl   puts
 mov  r0, #0
 pop  {fp, pc}
hello:
 .ascii "Hello world!\000"
```

Now suppose we want to turn this into shellcode…

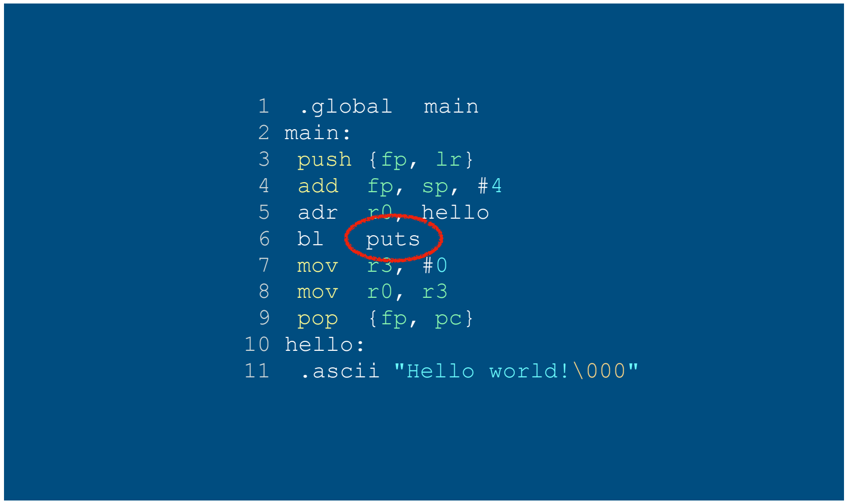
Recall how this works

```
eor r2, r2
adr r1, shell
push {r1, fp, lr}
pop {r0, fp, lr}
strb r2, [r1, #7]
push {r1, fp, lr}
add fp, sp, #4
mov r7, #11
…
```



Shellcode is written **independently** of the target.

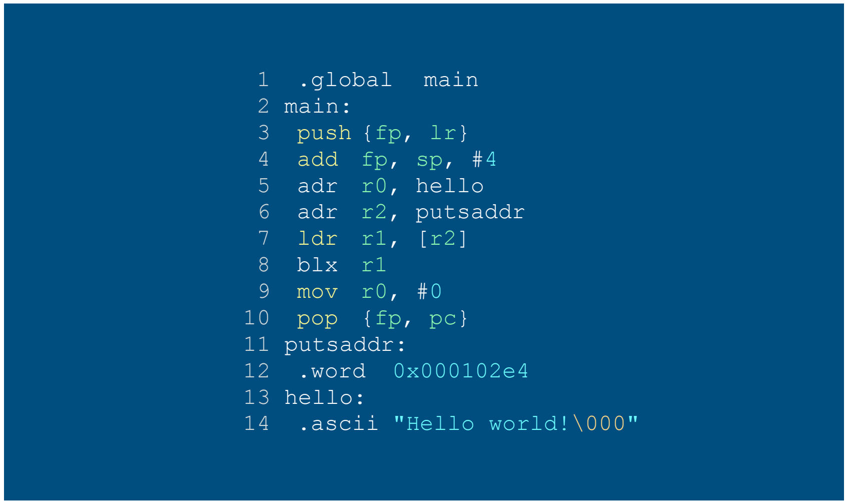
Can’t refer to all symbol names in target



```
 .global  main
main:
 push {fp, lr}
 add  fp, sp, #4
 adr  r0, hello
 bl   puts
 mov  r3, #0
 mov  r0, r3
 pop  {fp, pc}
hello:
 .ascii "Hello world!\000"
```

Symbols in **target** need to be translated into **addrs**

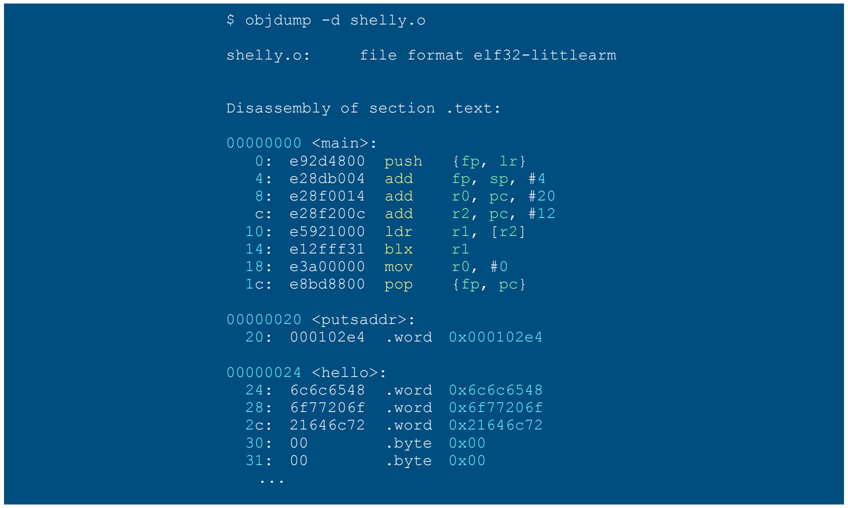
Suppose `puts` is `0x102e4` in target



```
 .global  main
main:
 push {fp, lr}
 add  fp, sp, #4
 adr  r0, hello
 adr  r2, putsaddr
 ldr  r1, [r2]
 blx  r1
 mov  r0, #0
 pop  {fp, pc}
putsaddr:
 .word  0x000102e4
hello:
 .ascii "Hello world!\000"
```

Better. But we have one more problem…

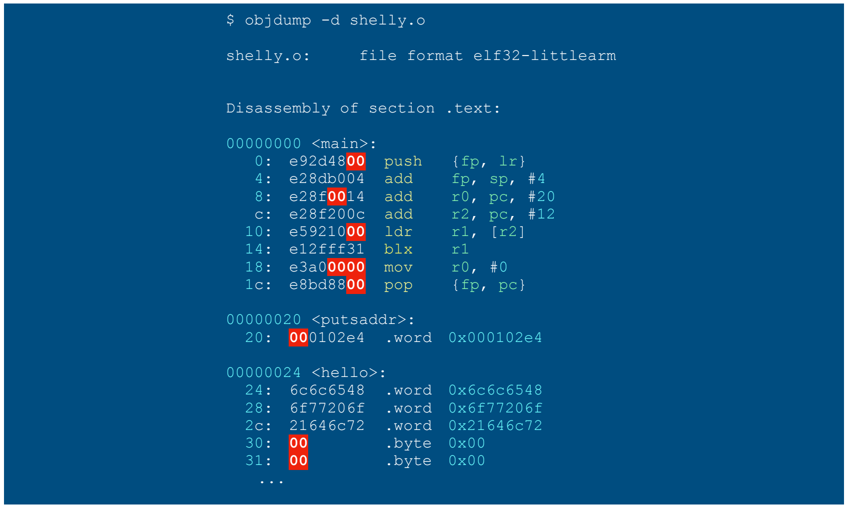
NULL bytes



```
$ objdump -d shelly.o

shelly.o:     file format elf32-littlearm


Disassembly of section .text:

00000000 <main>:
   0:  e92d4800  push   {fp, lr}
   4:  e28db004  add    fp, sp, #4
   8:  e28f0014  add    r0, pc, #20
   c:  e28f200c  add    r2, pc, #12
  10:  e5921000  ldr    r1, [r2]
  14:  e12fff31  blx    r1
  18:  e3a00000  mov    r0, #0
  1c:  e8bd8800  pop    {fp, pc}

00000020 <putsaddr>:
  20:  000102e4  .word  0x000102e4

00000024 <hello>:
  24:  6c6c6548  .word  0x6c6c6548
  28:  6f77206f  .word  0x6f77206f
  2c:  21646c72  .word  0x21646c72
  30:  00        .byte  0x00
  31:  00        .byte  0x00
    ...
```

Can you spot them?

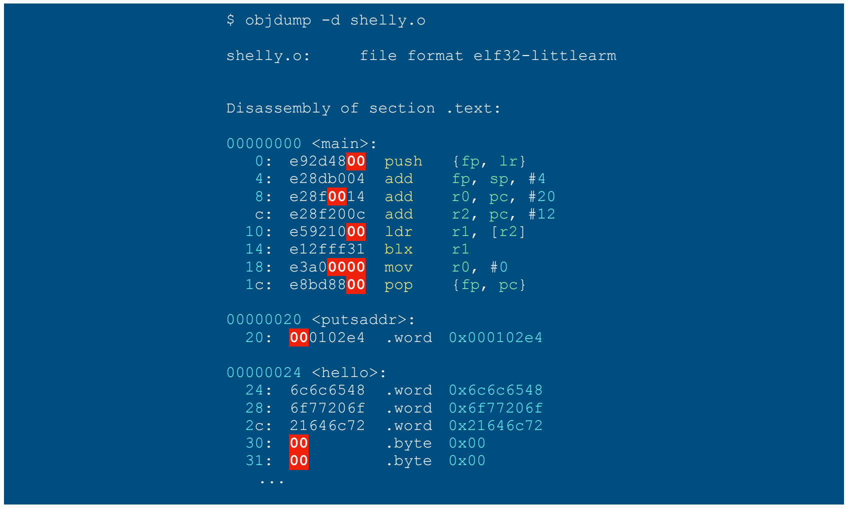
NULL bytes



```
$ objdump -d shelly.o

shelly.o:     file format elf32-littlearm


Disassembly of section .text:

00000000 <main>:
   0:  e92d4800  push   {fp, lr}
   4:  e28db004  add    fp, sp, #4
   8:  e28f0014  add    r0, pc, #20
   c:  e28f200c  add    r2, pc, #12
  10:  e5921000  ldr    r1, [r2]
  14:  e12fff31  blx    r1
  18:  e3a00000  mov    r0, #0
  1c:  e8bd8800  pop    {fp, pc}

00000020 <putsaddr>:
  20:  000102e4  .word  0x000102e4

00000024 <hello>:
  24:  6c6c6548  .word  0x6c6c6548
  28:  6f77206f  .word  0x6f77206f
  2c:  21646c72  .word  0x21646c72
  30:  00        .byte  0x00
  31:  00        .byte  0x00
   ...
```

Most C string handling functions will stop copying.

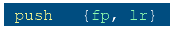
NULL bytes



```
$ objdump -d shelly.o

shelly.o:     file format elf32-littlearm


Disassembly of section .text:

00000000 <main>:
   0:  e92d4800  push   {fp, lr}
   4:  e28db004  add    fp, sp, #4
   8:  e28f0014  add    r0, pc, #20
   c:  e28f200c  add    r2, pc, #12
  10:  e5921000  ldr    r1, [r2]
  14:  e12fff31  blx    r1
  18:  e3a00000  mov    r0, #0
  1c:  e8bd8800  pop    {fp, pc}

00000020 <putsaddr>:
  20:  000102e4  .word  0x000102e4

00000024 <hello>:
  24:  6c6c6548  .word  0x6c6c6548
  28:  6f77206f  .word  0x6f77206f
  2c:  21646c72  .word  0x21646c72
  30:  00        .byte  0x00
  31:  00        .byte  0x00
   ...
```

We need to be creative to remove these.

Experiment using tiny examples



```
push   {fp, lr}
```

Experiment using tiny examples

experiment1.s

```
push   {fp, lr}
```

```
$ gcc -c experiment1.s

$ objdump -d experiment1.o

Disassembly of section .text:

00000000 <.text>:
   0:e92d4800   push {fp, lr}
```

experiment2.s

```
push   {r1, fp, lr}
```

```
$ gcc -c experiment2.s

$ objdump -d experiment2.o

Disassembly of section .text:

00000000 <.text>:
   0:e92d4802   push {r1, fp, lr}
```

If you do this, don’t forget that you have more to `pop` later.

Some tips

- Use `disas <fnname>` to find function in GDB (note: program must be loaded)
- Be careful where you put your stack!
- Use `.word` for 4-byte constants
- Use `.ascii` for NULL-free string literals
- Use `adr` to load the “address of” a value
- Use `ldr` to “dereference” a value
- Use `blx` to branch to a register (make sure MSB is zero!)
- `eor` a register to itself to generate zero values at runtime.
- Write self-modifying code!

Recap & Next Class

Today we learned:

NULL byte removal

Next class:

Social engineering