

Backdoors, Endianness

CS 2130: Computer Systems and Organization 1

Xinyao Yi Ph.D.
Assistant Professor

Announcements

- Homework 4 due Friday at 11:59pm on Gradescope
 - Note the earlier deadline!
 - You have written most of this code already
 - Lab 6 may provide a fast way to get started

Backdoors

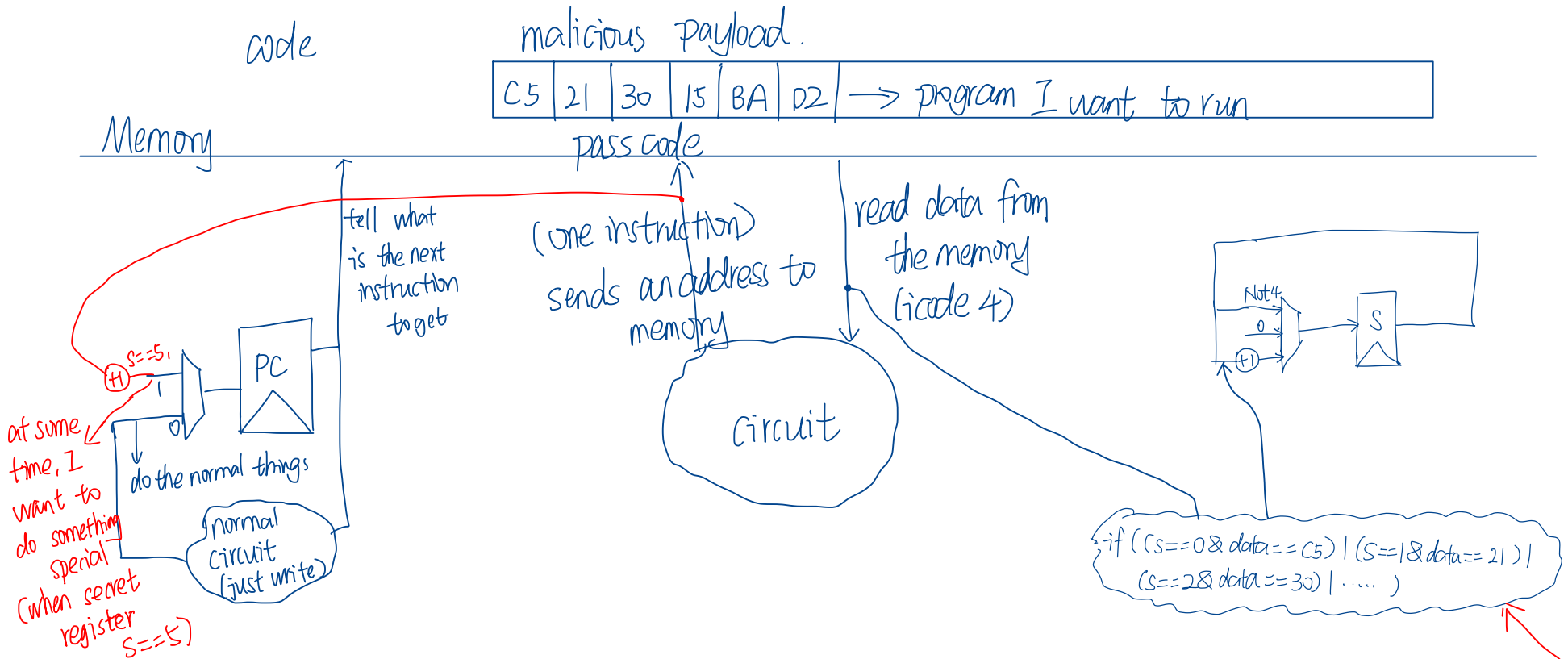
Backdoor: secret way in to do new unexpected things

- Get around the normal barriers of behavior
- Ex: a way in to allow me to take complete control of your computer

Exploit - a way to use a vulnerability or backdoor that has been created

- Our exploit today: a **malicious payload**
 - A passcode and program
 - If it ever gets in memory, run my program regardless of what you want to do

Our Hardware Backdoor



Something not icode 4: keep the original value
 if it is icode4: ①. reset counter if that's not part of my passcode
 ②. increasmentor: under special condition

Our Hardware Backdoor

Will you notice this on your chip?

Our Hardware Backdoor

Will you notice this on your chip?

- Modern chips have **billions** of transistors
- We're talking adding a few hundred transistors

Our Hardware Backdoor

Will you notice this on your chip?

- Modern chips have **billions** of transistors
- We're talking adding a few hundred transistors
- Maybe with a microscope? But you'd need to know where to look!

If I had a microscope and I knew exactly where to look, I probably can find it.

Or, maybe, I'm very, very lucky.

But - Most exploits are going to be found after somebody tries to use them.

Our Hardware Backdoor

Have you heard about something like this before?

- Sounds like something from the movies
- People claim this might be happening
- To the best of my knowledge, no one has ever admitted to falling in this trap

Ethics, Business, Tech

Are there reasons to do this? Not to do this?

- No technical reason not to, it's easy to do!

Ethics, Business, Tech

Are there reasons to do this? Not to do this?

- No technical reason not to, it's easy to do!
- Ethical implications
- Business implications (lawsuits, PR, etc)

Ethics, Business, Tech

Are there reasons to do this? Not to do this?

- No technical reason not to, it's easy to do!
- Ethical implications
- Business implications (lawsuits, PR, etc)

Can we make a system where one bad actor can't break it?

Ethics, Business, Tech

Are there reasons to do this? Not to do this?

- No technical reason not to, it's easy to do!
- Ethical implications
- Business implications (lawsuits, PR, etc)

Can we make a system where one bad actor can't break it?

- Code reviews, double checks, verification systems, automated verification systems, ...

Why does this work?

Why?

Why does this work?

- **It's all bytes!**
- Everything we store in computers are bytes
- We store code and data in the same place: memory

Von Neumann model

It's all bytes

Memory, Code, Data... It's all bytes!

- **Enumerate** - pick the meaning for each possible byte
- **Adjacency** - store bigger values together (sequentially)
- **Pointers** - a value treated as address of thing we are interested in

Enumerate

Enumerate - pick the meaning for each possible byte

Assign meaning to this byte.

What is 8-bit 0x54?

Unsigned integer

Signed integer

Floating point w/ 4-bit exponent

ASCII

Bitvector sets

Our example ISA

eighty-four

positive eighty-four

twelve

capital letter T: T

The set {2, 3, 5}

Flip all bits of value in r1

Adjacency

Adjacency - store bigger values together (sequentially)

- An array: build bigger values out of many copies of the same type of small values

- Store them next to each other in memory

- Arithmetic to find any given value based on index

We know: ①. The address of the first element. (addr)

②. The index of that element (i)

Then the address of that element : $addr + (i * \text{size of each element})$.

Adjacency

Adjacency - store bigger values together (sequentially)

One row in a database table, like one line in a CSV file.

- Records, structures, classes

- Classes have fields! Store them adjacently

- Know how to access (add offsets from base address) *addr + offset_of - π*

- If you tell me where object is, I can find fields

Pointers

Pointers - a value treated as address of thing we are interested in

- A value that really points to another value
- Easy to describe, hard to use properly
- We'll be talking about these a lot in this class!

Those 3 things, we combine them all together. And this is kind of how we're storing the data in the memory.

Pointers

Pointers - a value treated as address of thing we are interested in

- Give us strange new powers (represent more complicated things), e.g.,
 - Variable-sized lists
 - Values that we don't know their type without looking
 - Dictionaries, maps

Programs Use These!

How do our programs use these?

- Enumerated icode, numbers
- Adjacent stored instructions (PC+1)
- Pointers of where to jump/goto (addresses in memory)

ToyISA Instructions

So far, only dealing with 8-bit machine!

→ 8-bit instructions/values/memory addresses

icode	b	meaning
0		$rA = rB$
1		$rA \&= rB$
2		$rA += rB$
3	0	$rA = \sim rA$
	1	$rA = !rA$
	2	$rA = -rA$
	3	$rA = pc$
4		$rA = \text{read from memory at address } rB$
5		write rA to memory at address rB
6	0	$rA = \text{read from memory at } pc + 1$
	1	$rA \&= \text{read from memory at } pc + 1$
	2	$rA += \text{read from memory at } pc + 1$
	3	$rA = \text{read from memory at the address stored at } pc + 1$
		For icode 6, increase pc by 2 at end of instruction
7		Compare rA as 8-bit 2's-complement to 0 if $rA \leq 0$ set $pc = rB$ else increment pc as normal

64-bit Machines

64-bit machine: The registers are 64-bits

- i.e., r0, but also PC

Important to have large values. Why?

64-bit Machines

64-bit machine: The registers are 64-bits

- i.e., r0, but also PC

Important to have large values. Why?

- Most important: PC ^{PC is wider} and memory addresses *We need space to do things.*
- How much memory could our 8-bit machine access?

$2^8 = 256$ bytes. 00 - FF.

64-bit Machines

64-bit machine: The registers are 64-bits

- i.e., r0, but also PC

Important to have large values. Why?

- Most important: PC and memory addresses
- How much memory could our 8-bit machine access? 256 Bytes

64-bit Machines

64-bit machine: The registers are 64-bits

- i.e., r0, but also PC

Important to have large values. Why?

- Most important: PC and memory addresses
- How much memory could our 8-bit machine access? 256 Bytes
- Late 70s - 16 bits:

64-bit Machines

64-bit machine: The registers are 64-bits

- i.e., r0, but also PC

Important to have large values. Why?

- Most important: PC and memory addresses
- How much memory could our 8-bit machine access? **256 Bytes**
- Late 70s - 16 bits: **65536 Bytes** (2^{16})

64-bit Machines

64-bit machine: The registers are 64-bits

- i.e., r0, but also PC

Important to have large values. Why?

- Most important: PC and memory addresses
- How much memory could our 8-bit machine access? **256 Bytes**
- Late 70s - 16 bits: **65536 Bytes** (*roughly 65 kilo bytes*).
- 80s - 32 bits:

64-bit Machines

64-bit machine: The registers are 64-bits

- i.e., r0, but also PC

Important to have large values. Why?

- Most important: PC and memory addresses
- How much memory could our 8-bit machine access? **256 Bytes**
- Late 70s - 16 bits: **65536 Bytes**
- 80s - 32 bits: $\approx$ **4 billion bytes**

64-bit Machines

64-bit machine: The registers are 64-bits

- i.e., r0, but also PC

Important to have large values. Why?

- Most important: PC and memory addresses
- How much memory could our 8-bit machine access? **256 Bytes**
- Late 70s - 16 bits: **65536 Bytes**
- 80s - 32 bits: ≈ 4 billion bytes (4 GiB)
- Today's processors - 64 bits:

64-bit Machines

64-bit machine: The registers are 64-bits

- i.e., r0, but also PC

Important to have large values. Why?

- Most important: PC and memory addresses
- How much memory could our 8-bit machine access? **256 Bytes**
- Late 70s - 16 bits: **65536 Bytes**
- 80s - 32 bits: ≈ 4 billion bytes
- Today's processors - 64 bits: 2^{64} addresses (2^{64} indices to memory)

Aside: Powers of Two

Value	base-10	Short form	Pronounced
2^{10}	(10^3) 1024	Ki	Kilo
2^{20}	1,048,576	Mi	Mega
2^{30}	1,073,741,824	Gi	Giga
2^{40}	(10^{12}) 1,099,511,627,776	Ti	Tera
2^{50}	1,125,899,906,842,624	Pi	Peta
2^{60}	1,152,921,504,606,846,976	Ei	Exa

Example: 2^{27} bytes

If I have 2 to some power, it works out to be roughly equivalent to a power of 10.

Aside: Powers of Two

Value	base-10	Short form	Pronounced
2^{10}	1024	Ki	Kilo
2^{20}	1,048,576	Mi	Mega
2^{30}	1,073,741,824	Gi	Giga
2^{40}	1,099,511,627,776	Ti	Tera
2^{50}	1,125,899,906,842,624	Pi	Peta
2^{60}	1,152,921,504,606,846,976	Ei	Exa

Example: 2^{27} bytes = $2^7 \times 2^{20}$ bytes

$$2^m \times 2^n = 2^{m+n}$$

Aside: Powers of Two

Value	base-10	Short form	Pronounced
2^{10}	1024	Ki	Kilo
2^{20}	1,048,576	Mi	Mega
2^{30}	1,073,741,824	Gi	Giga
2^{40}	1,099,511,627,776	Ti	Tera
2^{50}	1,125,899,906,842,624	Pi	Peta
2^{60}	1,152,921,504,606,846,976	Ei	Exa

Example: 2^{27} bytes = $2^7 \times 2^{20}$ bytes = 2^7 MiB = 128 MiB

64-bit Machines

How much can we address with 64-bits?

64-bit Machines

How much can we address with 64-bits?

- 16 EiB (2^{64} addresses = $2^4 \times \underbrace{2^{60}}_{\text{Exa}}$)

64-bit Machines

How much can we address with 64-bits?

- 16 EiB (2^{64} addresses = $2^4 \times 2^{60}$)
- But I only have 8 GiB of RAM

*I have 16 EiB of addresses. We can address more space we actually have.
But it could be used for virtual memory.
We will talk about it in CS0-2.*