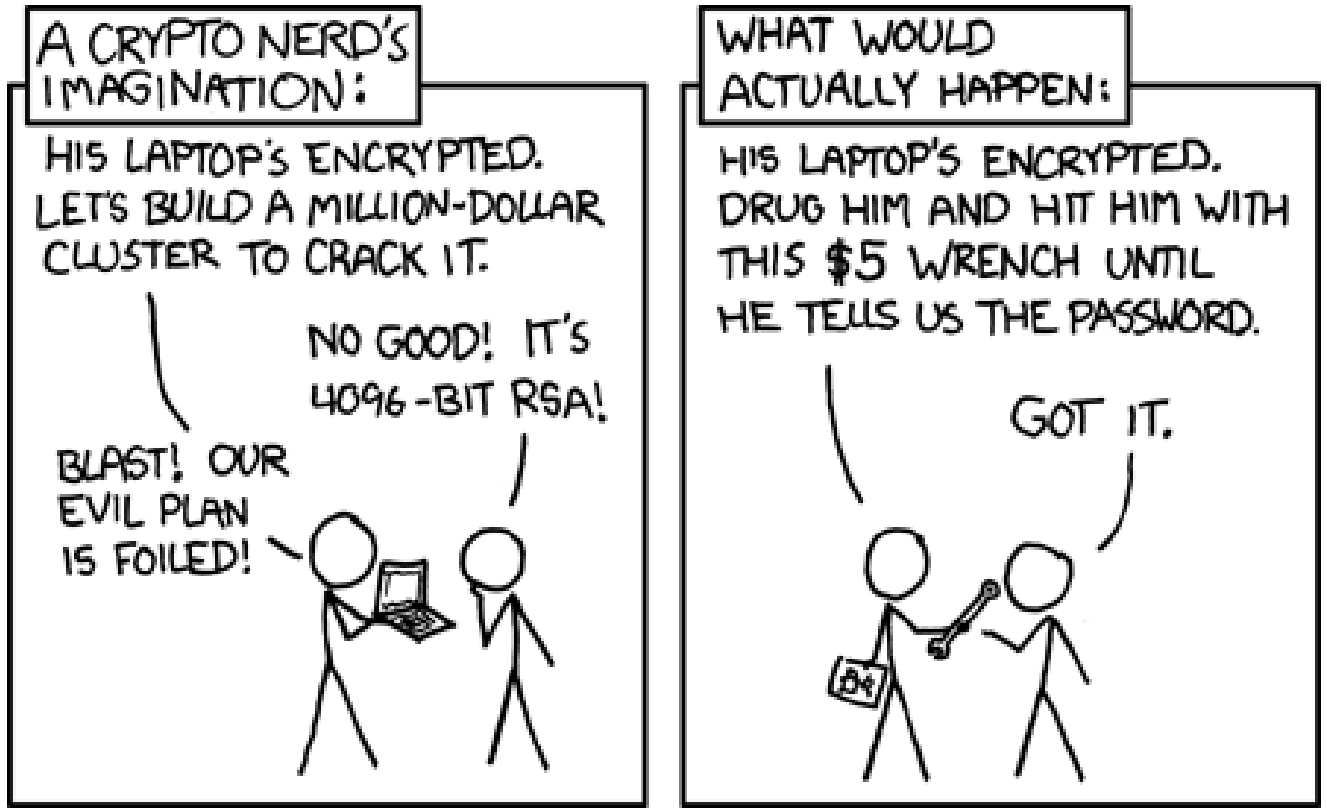
Security

Kevin Webb

Swarthmore College

April 23, 2020

xkcd #538



Actual actual reality: nobody cares about his secrets.
(Also, I would be hard-pressed to find that wrench for $5.)

Today's Goals

- What security means, and the challenges of working in this area.
- Broad categories of attacks.
- Specific attacks and ways to defend against them.
- (This is intended to be a fun topic. It's related to OS, but not necessarily "CORE OS".)

Security

- Perpetual arms race between attackers and defenders.
  - Attacker: “How can I cheat the system? I’m willing to lie/cheat/steal…”
  - Defender: “How can I prevent cheating, or at least make it not worth it…”
- Security is both interesting and difficult: often very clever solutions.
- Today: brief introduction into some of the most impactful ideas.

Security Research

- Theoretical security: how can we design an algorithm / system that can't be cheated?
  - Example: cryptography
- Focus today: applied security
  - How can we compromise the integrity of a real system?
  - Often attack the implementation, not the idea.
- Promise me you won't use anything here for evil!

“Threat Model”

- Thinking about security, it helps to characterize the potential attacker(s), their capabilities, and their motivations.

- Where are your system’s weaknesses?

- Defenses look VERY different when:
  - Attackers are bored teenagers who are looking for any old website to deface for laughs.
  - Attackers are a well-funded and trained foreign government looking to steal specific state secrets.

Broad Classes of Attacks

1. Social engineering

2. Denial of service

3. Information leakage
   - Due to coding bugs
   - Side-channel attacks

4. Arbitrary / remote code execution

Social Engineering

- Idea: humans in the loop are often the weak point, target them…
- Real world: fancy way of saying “tricking people”.
- Famous example: Kevin Mitnick

Social Engineering

- https://en.wikipedia.org/wiki/Kevin_Mitnick

Social Engineering

- Other examples: phishing, robo calls, “spear phishing”, “smishing” / “cat phishing”

- Mitigation solutions:
  - Better training / education
  - Law enforcement

    …
  - Warning users about suspicious emails / web pages (perhaps w/ crypto keys)

Denial of Service (DoS)

- Goals:
  - Prevent victim from doing business.
  - Send some sort of political / ideological message.
  - Ruin someone's day for "fun".

- Real-world analogy: picketing outside a building / blocking entry.

Cat performing the classic "denial of pizza" attack.

DoS Examples

1. Attacker discovers a bug that causes a crash, triggers it intentionally.

Suppose you wanted to attack your Lab 5 file system. Are there vulnerabilities you could exploit to trigger a crash?

DoS Examples

1. Attacker discovers a bug that causes a crash, triggers it intentionally.

- Hypothetical example: you're writing a file system, and you don't validate that read offsets are less than the size of the file.

- Real example: "ping of death" (1997ish)
  - IP uses 16 bits to indicate packet size. (65535 bytes max)
  - Due to weirdness of "IP fragmentation", it's possible to send larger packets (in violation of the specification!)
  - Nearly every OS that received such a message would crash.

DoS Examples

2. Intentionally overloading resource capacity.

- Classic “fork bomb”:

```
while (1)
    fork();
```

- Distributed denial of service:
  - Flood a victim with so much traffic from all over the Internet, they can’t do any real work.

Distributed Denial of Service (DDoS)

- Possibility 1: Control a “bot net”.
  - Use remote execution attack to take control over lots of computers.
  - Tell them all to flood victim with traffic.
  - Laugh at and/or extort victim for ransom money

- Possibility 2: Exploit other resources / protocols on the Internet.
  - Example: DNS Amplification Attack
  - Normal behavior: make a DNS name query (e.g., swarthmore.edu), get back IP address or error message explaining what went wrong.
  - Exploit: lie about who you are, make queries that fail intentionally.

Notable DDoS Example: “Lizard Squad”

- https://en.wikipedia.org/wiki/Lizard_Squad

Notable DDoS Example: “Lizard Squad”

- https://en.wikipedia.org/wiki/Lizard_Squad

- Aftermath: https://www.engadget.com/2017/12/22/lizard-squad-hacker-founder-guilty/

DoS Examples

3. “Ransomware”: Hold user’s files hostage.
   - Use remote execution attack to encrypt a user’s files.
   - Make them pay to get the files back.

- Notable example: WannaCry (May 2017)
  - Affected >230,000 machines, including UK’s NHS, FedEx, Honda, many others

- Similar attacks have been used against people on our campus!

DoS Mitigation

- Varies a lot depending on attack type…

- In general…
  - Write bug-free code! :D
  - Find and punish offenders.
  - Audit code for bugs, fix them, distribute updates.

Relevant:

https://www.google.com/search?q=windows+update+meme

Information Leakage

1. Buggy program reveals information that it’s not supposed to reveal.

2. Side channel: Attacker learns information by observing seemingly innocuous information.

Suppose you wanted to attack your Lab 5 file system. Are there vulnerabilities you could exploit to leak unauthorized information?

- (assume files had access control)

Information Leakage Examples

- Hypothetical example: you're writing a file system, and you don't validate that read offsets are less than the size of the file.
  - Accidentally read / retrieve for user data in neighboring file.

- Recent example:
  - http://www.cbc.ca/news/canada/nova-scotia/freedom-of-information-request-privacy-breach-teen-speaks-out-1.4621970

Side Channels

- Learn information you're not supposed to know, despite barriers in place.

- "Real (?) world" example: rent hotel room next to victim, attempt to listen in through walls.

- Simple computing example:
  - You want to know whether your victim is using their computer or not, but you can't log in. Point thermometer at it, measure heat output.

More Interesting Example…

- Threat model: determined attacker is targeting specific victim. Attacker knows target using public cloud resources, can use them too.

- Suppose victim is a web service, attacker can interact with them.

- Scenario: cloud service provider, with multiple users sharing a physical machine. Each user gets a private VM, with copy of OS.
  - Problem: library code (e.g., encryption code) is the same on both.

More Interesting Example

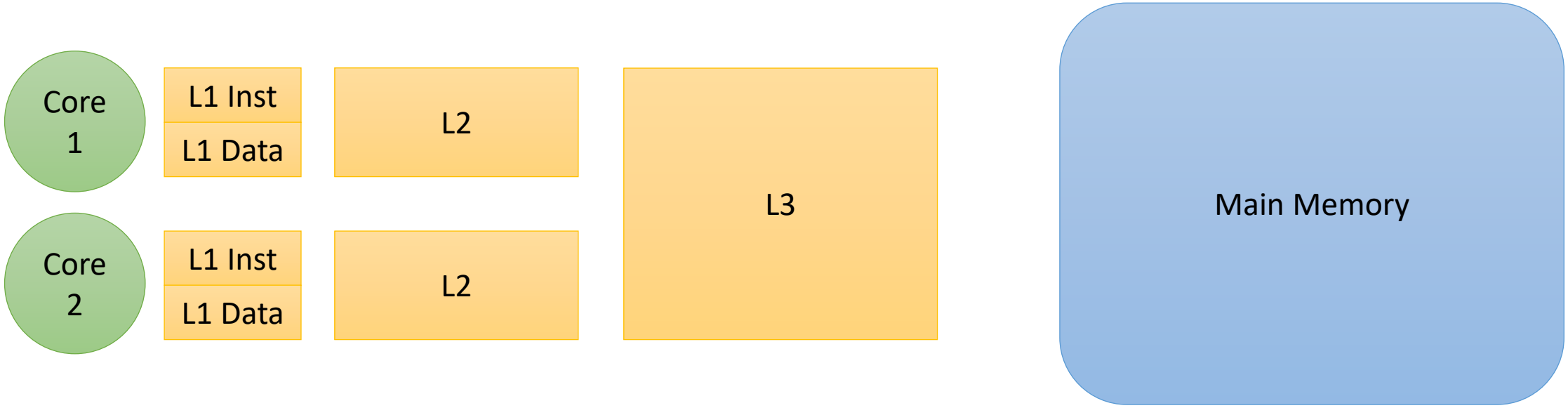
- Attacker: request many VMs, attempt to get one that is physically co-located with target victim.

- Attacker:
  - Load CPU cache with data.
  - Trigger victim to do something with encryption keys.
  - Read back original data, and time how long it takes to read.
    - Fast? => still in cache, victim didn't evict that data.
    - Slow? => victim used something that conflicted, evicted attacker data

More Interesting Example

- Attacker:
  - Load CPU cache with data.
  - Trigger victim to do something with encryption keys.
  - Read back original data, and time how long it takes to read.
    - Fast? => still in cache, victim didn't evict that data.
    - Slow? => victim used something that conflicted, evicted attacker data
- So what? How does cache usage tell us anything?

Cache Architecture



- If only certain data is slow, it might tell us whether or not victim executed certain *instructions* (e.g., did they take an *if* branch).

- Which *ifs* do we care about? The ones that interact with encryption keys!

Related: Spectre and Meltdown (2018)

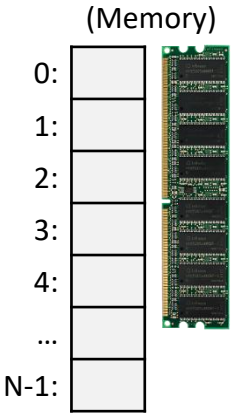
- Modern CPU: deeply pipelined execution
  - Multiple instructions in-flight at once, at different stages
  - In many cases, memory values not known yet
  - For example, during a branch (if), CPU might not know which side will be taken
  - Rather than wait, guess (“branch prediction”). If wrong, discard results.

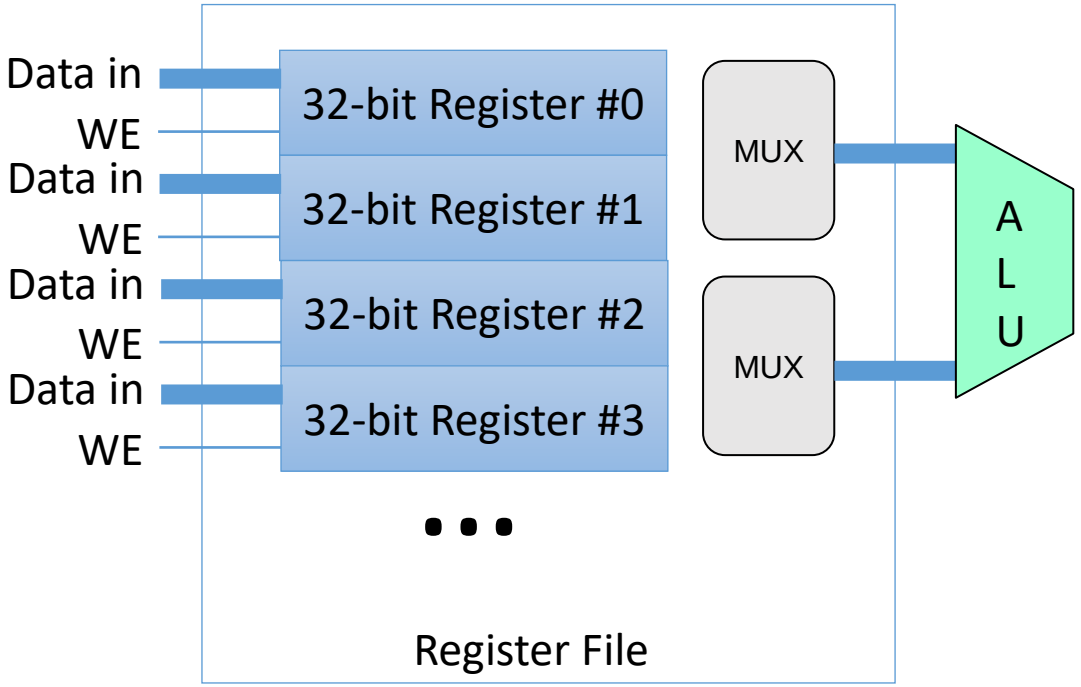
(CS 31) Recap CPU Model

Four stages: fetch instruction, decode instruction, execute, store result

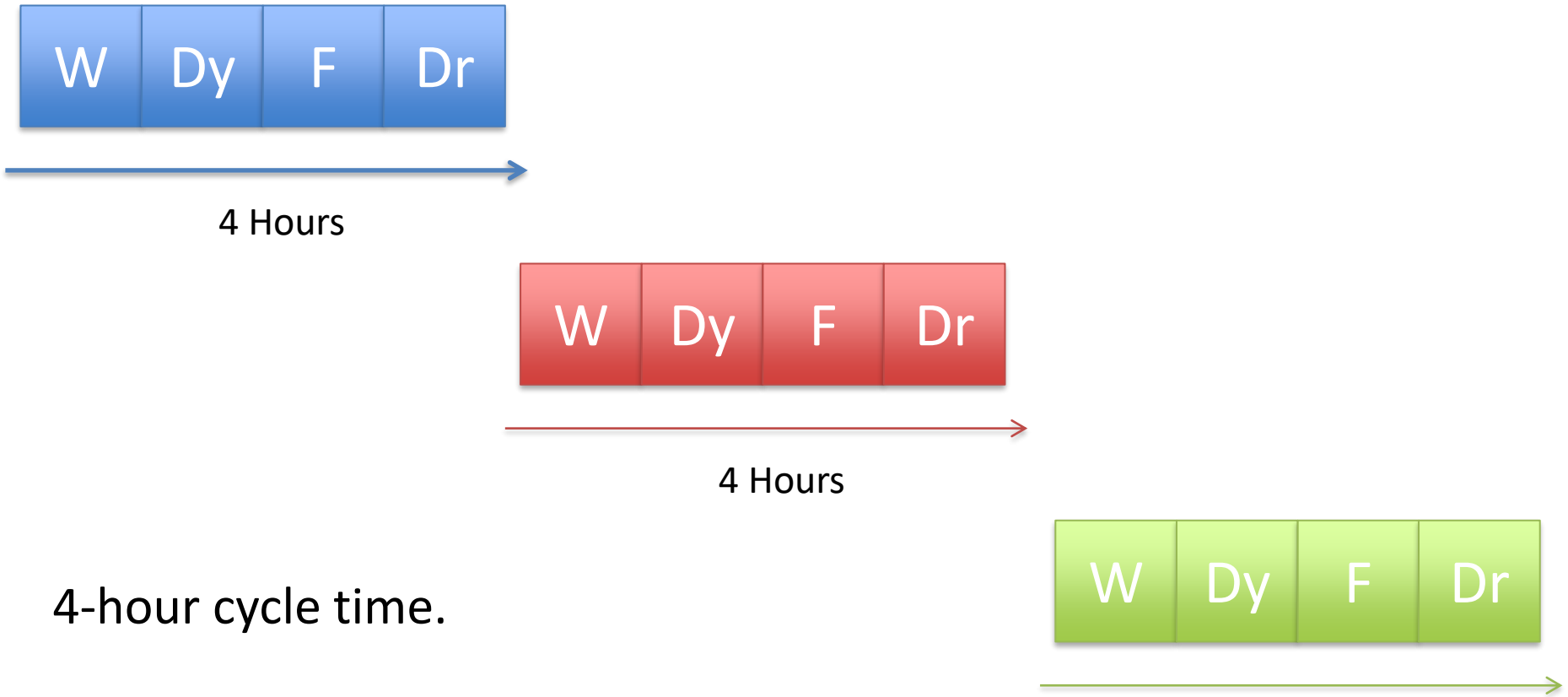
Program Counter (PC): **Memory address of next instr**

Instruction Register (IR): **Instruction contents (bits)**





Laundry

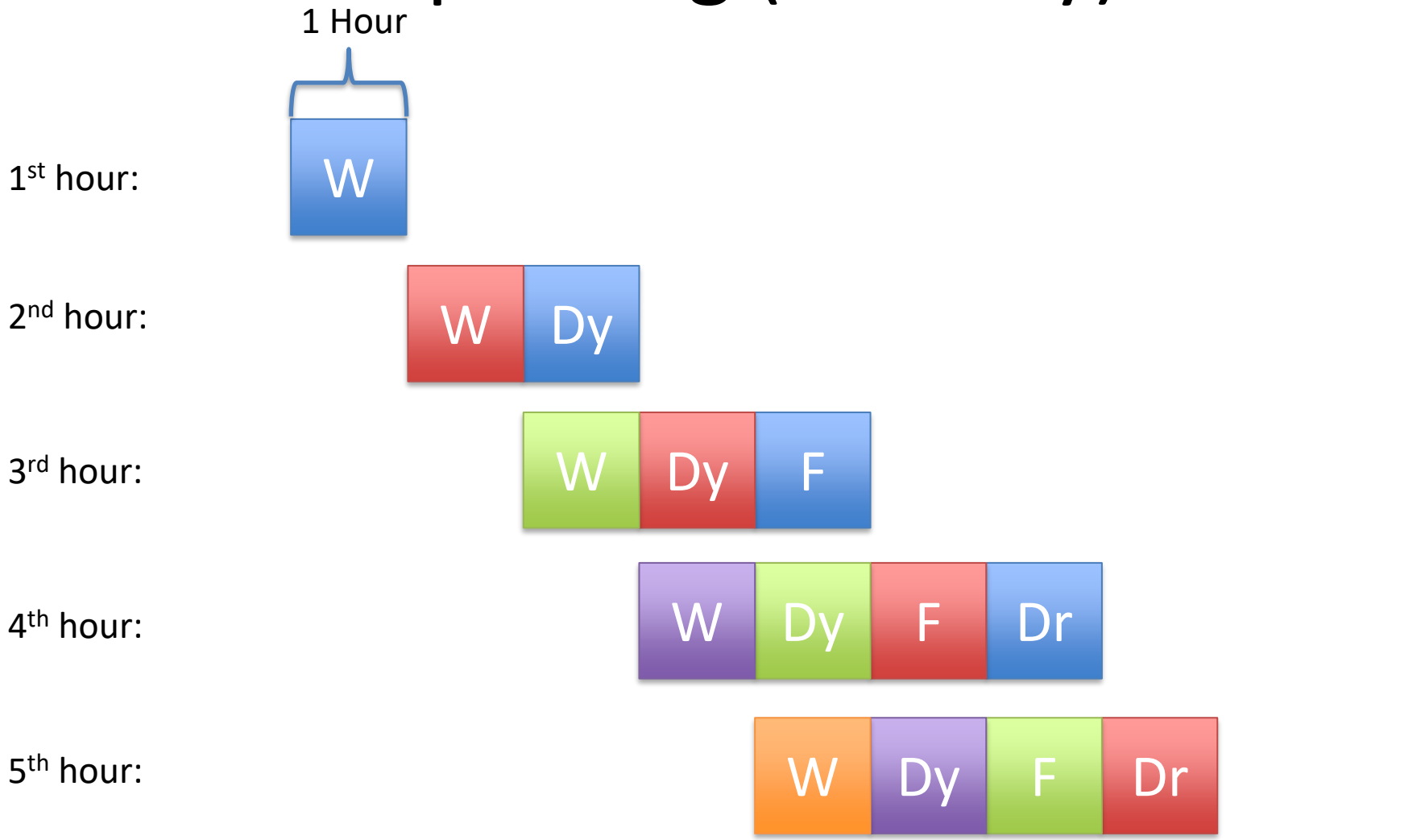


4-hour cycle time.

Finishes a laundry load every cycle.

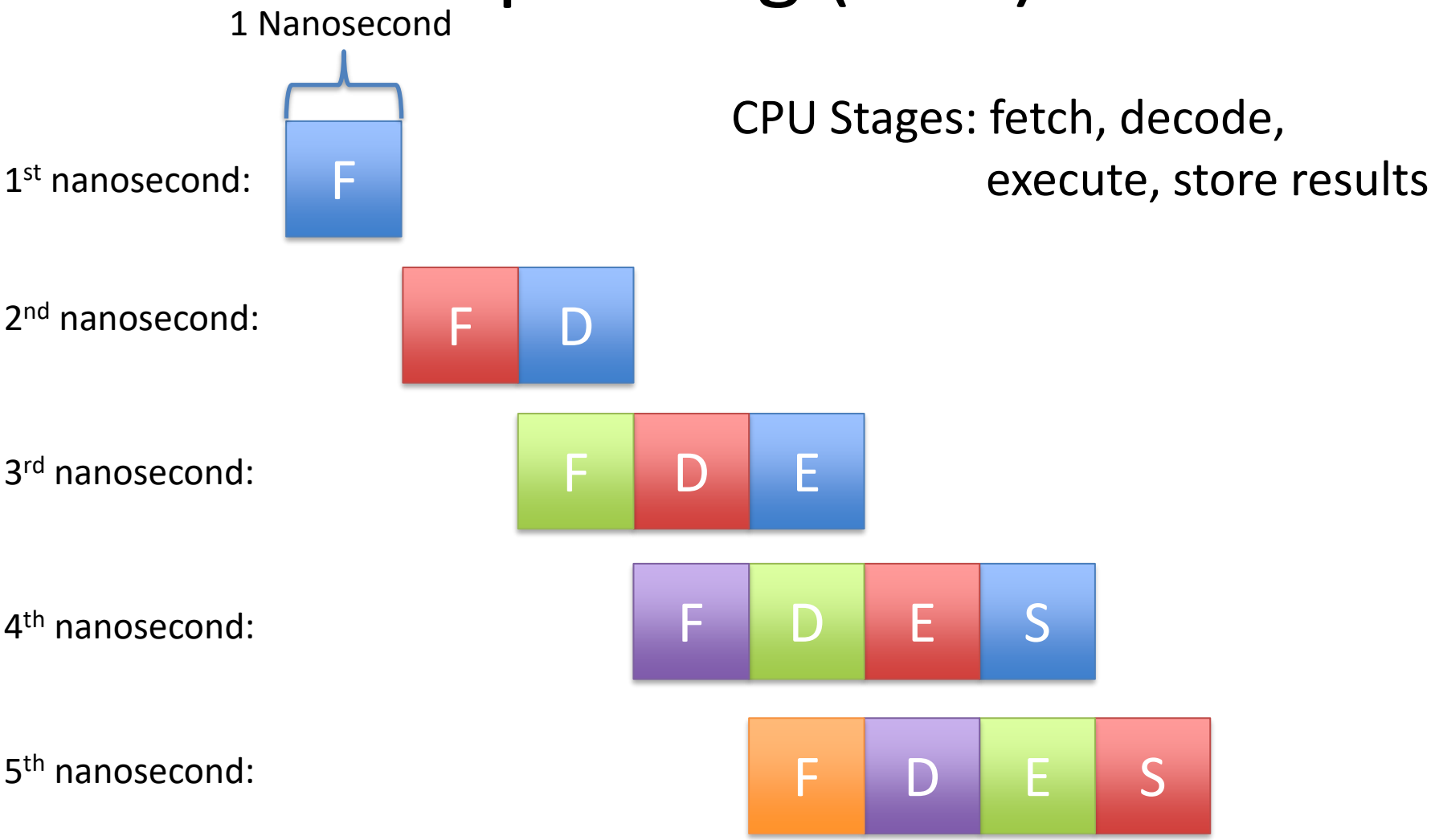
(6 laundry loads per day)

Pipelining (Laundry)



Steady state: One load finishes every hour!
(Not every four hours like before.)

Pipelining (CPU)



CPU Stages: fetch, decode, execute, store results

Steady state: One instruction finishes every nanosecond!
(Clock rate can be faster.)

CPU Pipelining - Reality

Year	Micro-architecture	Pipeline stages	max. Clock	Tech process
1989	486 (80486)	3	100 MHz	1000 nm
1993	P5 (Pentium)	5	300 MHz	600 nm
1995	P6 (Pentium Pro; later Pentium II)	14 (17 with load & store/retire)	450 MHz	350 nm
1999	P6 (Pentium III) (Copper Mine)	12 (15 with load & store/retire)	1400 MHz	250 nm
2000	NetBurst (Pentium 4) (Willamette)	20 unified with branch prediction	2000 MHz	180 nm
2002	NetBurst (Pentium 4) (Northwood, Gallatin)		3466 MHz	130 nm
2003	Pentium M	10 (12 with fetch/retire)	2133 MHz	130 nm
2004	NetBurst (Pentium 4) (Prescott)	31 unified with branch prediction	3800 MHz	90 nm
2006	Intel Core	12 (14 with fetch/retire)	3000 MHz	65 nm
2007	Penryn		3333 MHz	45 nm
2008	Nehalem	20 unified (14 without miss prediction)	3600 MHz	
	Bonnell	16 (20 with prediction miss)	2100 MHz	
2010	Westmere	20 unified (14 without miss prediction)	3730 MHz	

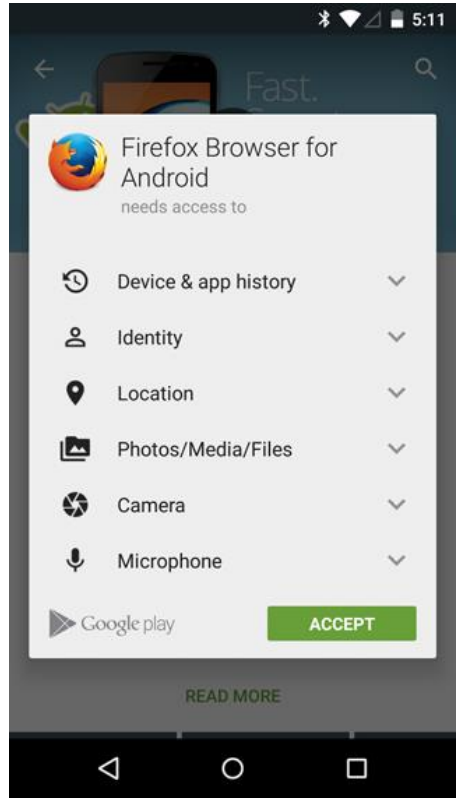
Source: https://en.wikipedia.org/wiki/List_of_Intel_CPU_microarchitectures

Related: Spectre and Meltdown (2018)

- Modern CPU: deeply pipelined execution
  - Multiple instructions in-flight at once, at different stages
  - In many cases, memory values not known yet
  - For example, during a branch (if), CPU might not know which side will be taken
  - Rather than wait, guess ("branch prediction"). If wrong, discard results.
- Hardware problem: discarded speculative executions were changing the state of the CPU (caches, branch target buffers)
- Use CPU "speculative execution" to learn about the state of the cache, even for memory you're not allowed to access.

Side Channels with Sensors

- Many of us have smart phones, which have lots of sensors…
- Some sensors (microphone, camera, GPS) clearly abusable.
  - User presented with choice to allow/deny apps access.
- Other sensors (gyroscope, accelerometer) require no user approval to use in applications.



Suppose a user installs your app. How can you use something like a phone's gyroscope for evil?

- Gyrophone: Recognizing Speech from Gyroscope Signals
- https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-michalevsky.pdf

"We achieve about 50% success rate for speaker identification from a set of 10 speakers. We also show that while limiting ourselves to a small vocabulary consisting solely of digit pronunciations ("one", "two", "three", ...) and achieve speech recognition success rate of 65% for the speaker dependent case and up to 26% recognition rate for the speaker independent case. This capability allows an attacker to substantially leak information about numbers spoken over or next to a phone (i.e. credit card numbers, social security numbers and the like)."

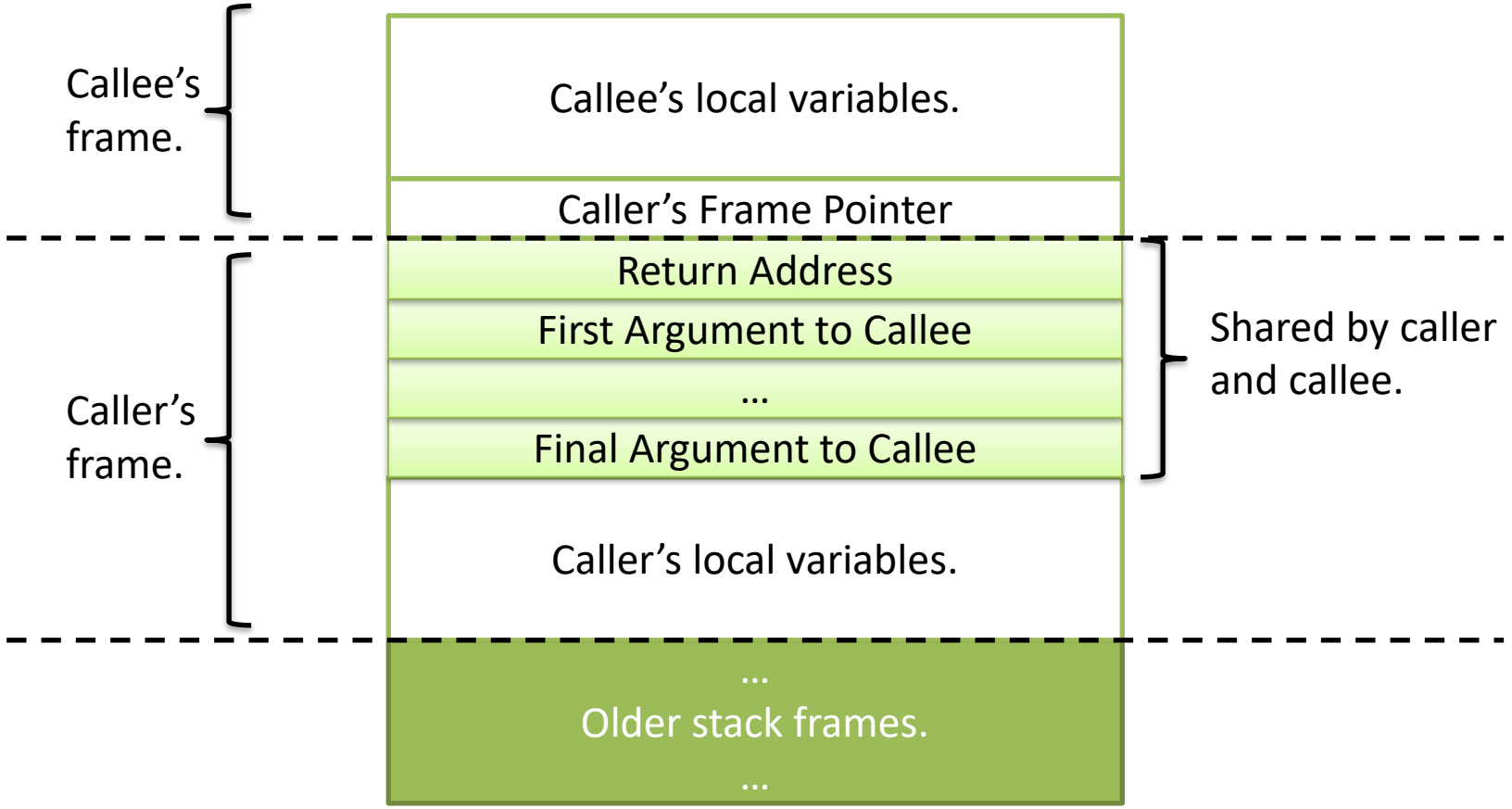
Arbitrary Execution

- Take over someone else's machine, make it execute your evil commands. (called "remote" code execution over network)

- "Real world": hypnosis? mind control?

- Historically, requires two major steps:
  1. Get evil instructions into target/victim program.
  2. Cause victim to execute those instructions (alter control flow).

Recall: The Stack



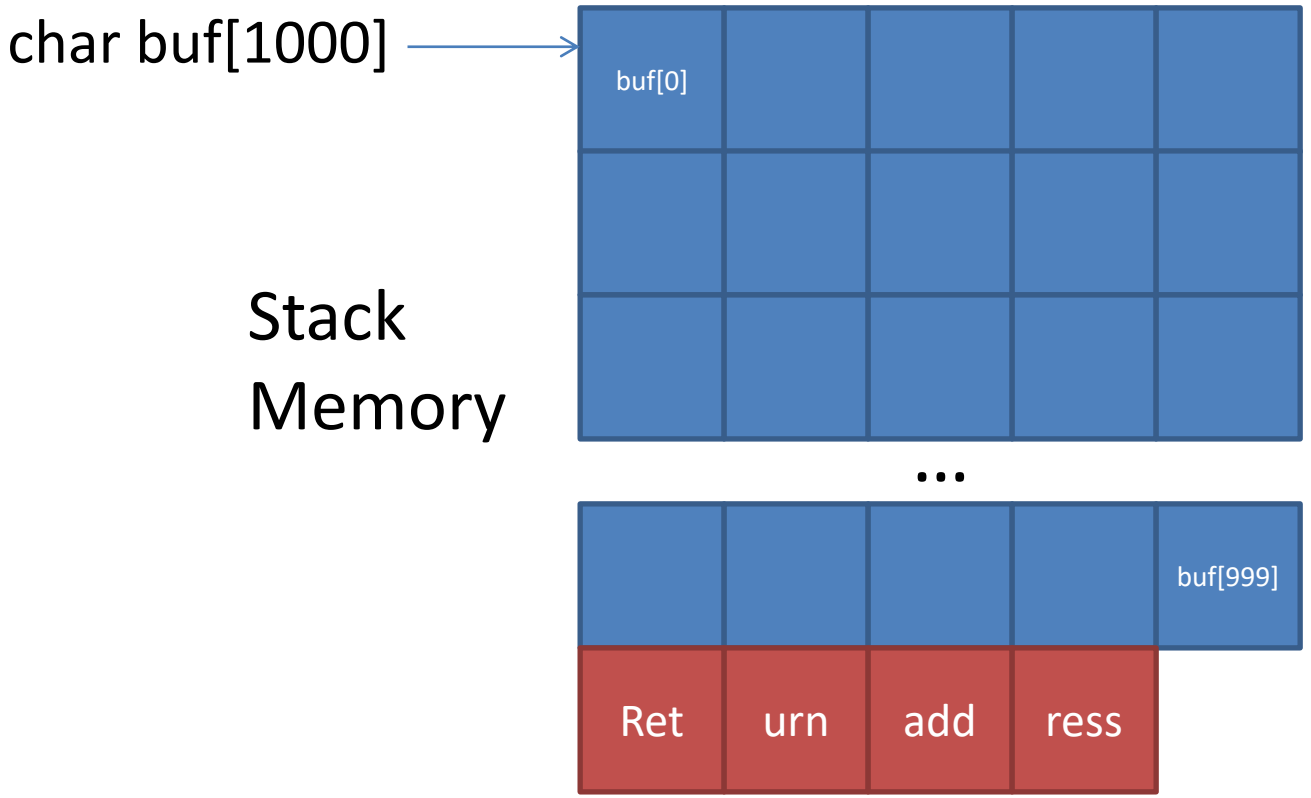
Classic Buffer Overflow

- See: “Smashing the Stack for Fun and Profit”

- Goal: overwrite the return address to hijack control flow. Make PC jump to instructions from attacker input.

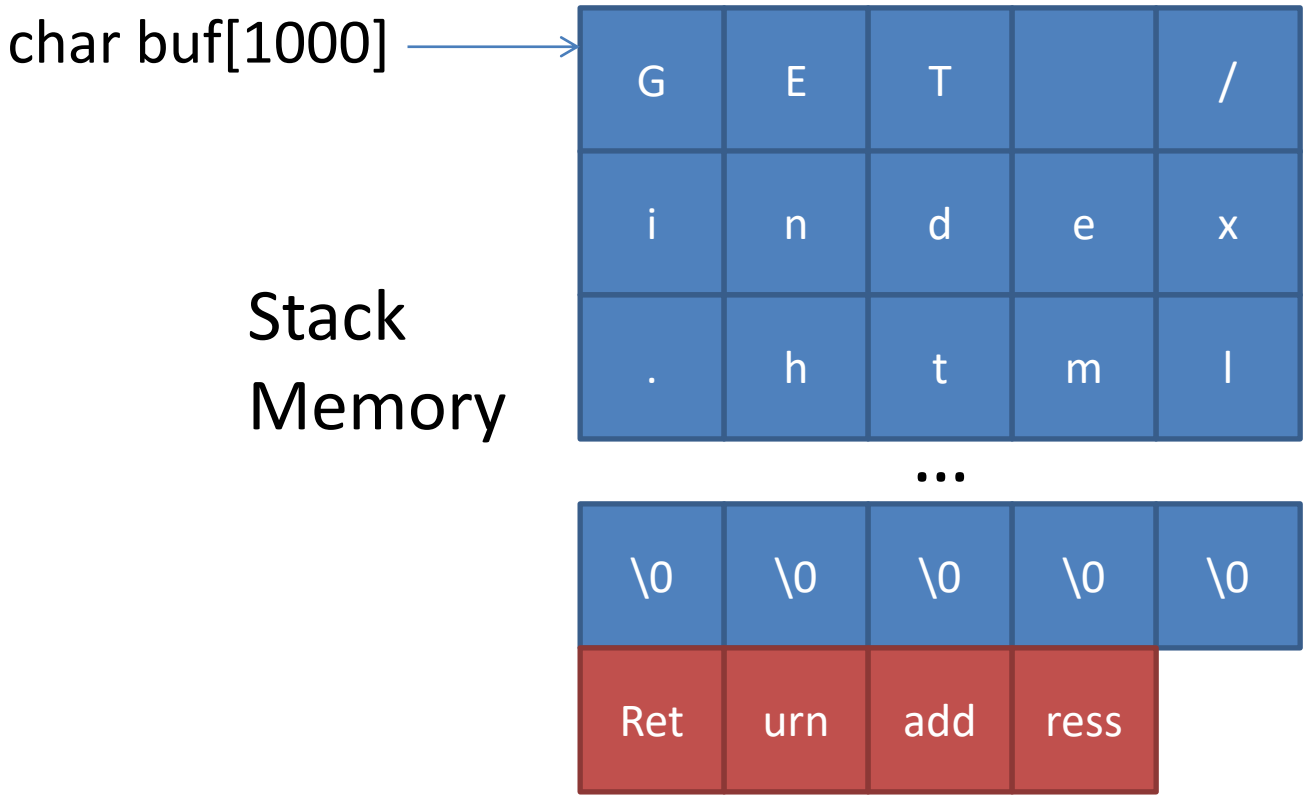
A well intentioned program…

- Suppose we have a protocol that does recv() until it finds \r\n\r\n. (or even just reading from stdin)



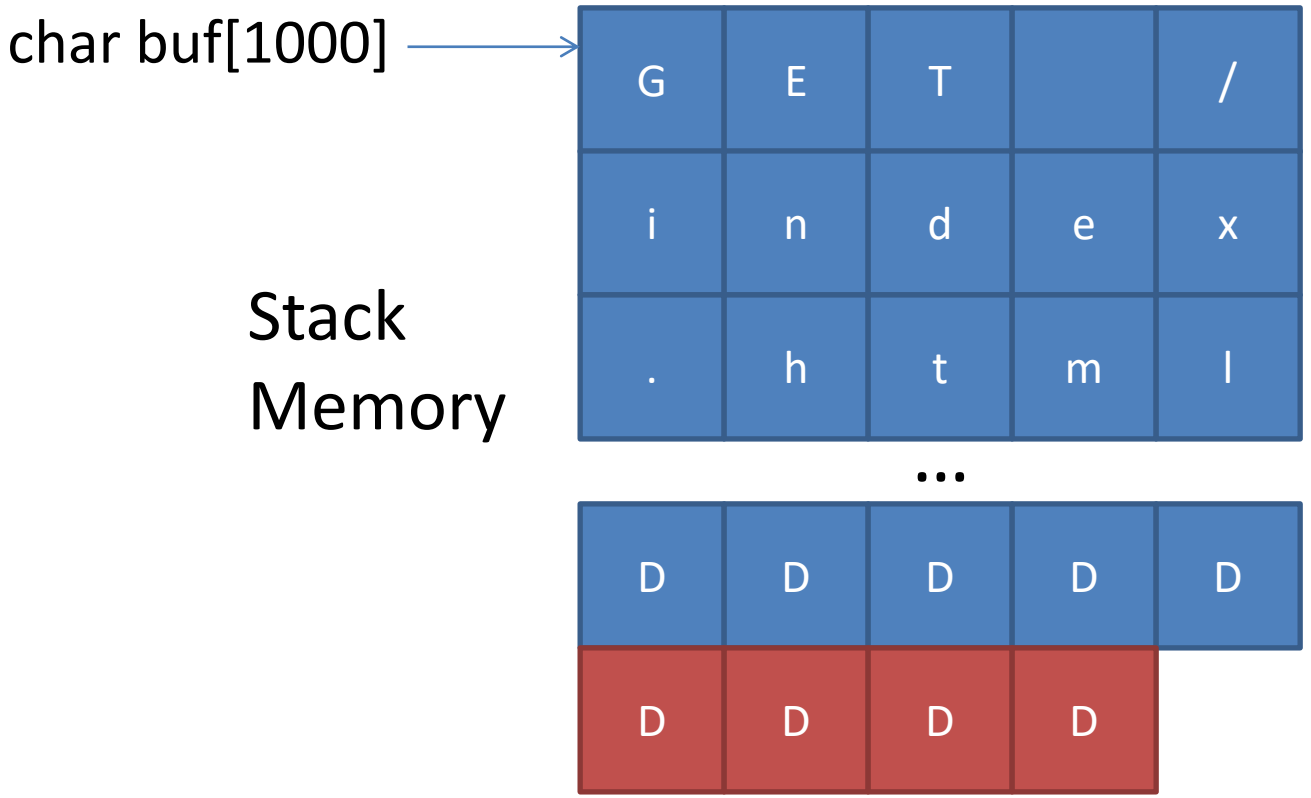
A well intentioned program…

- Suppose we have a protocol that does recv() until it finds \r\n\r\n.



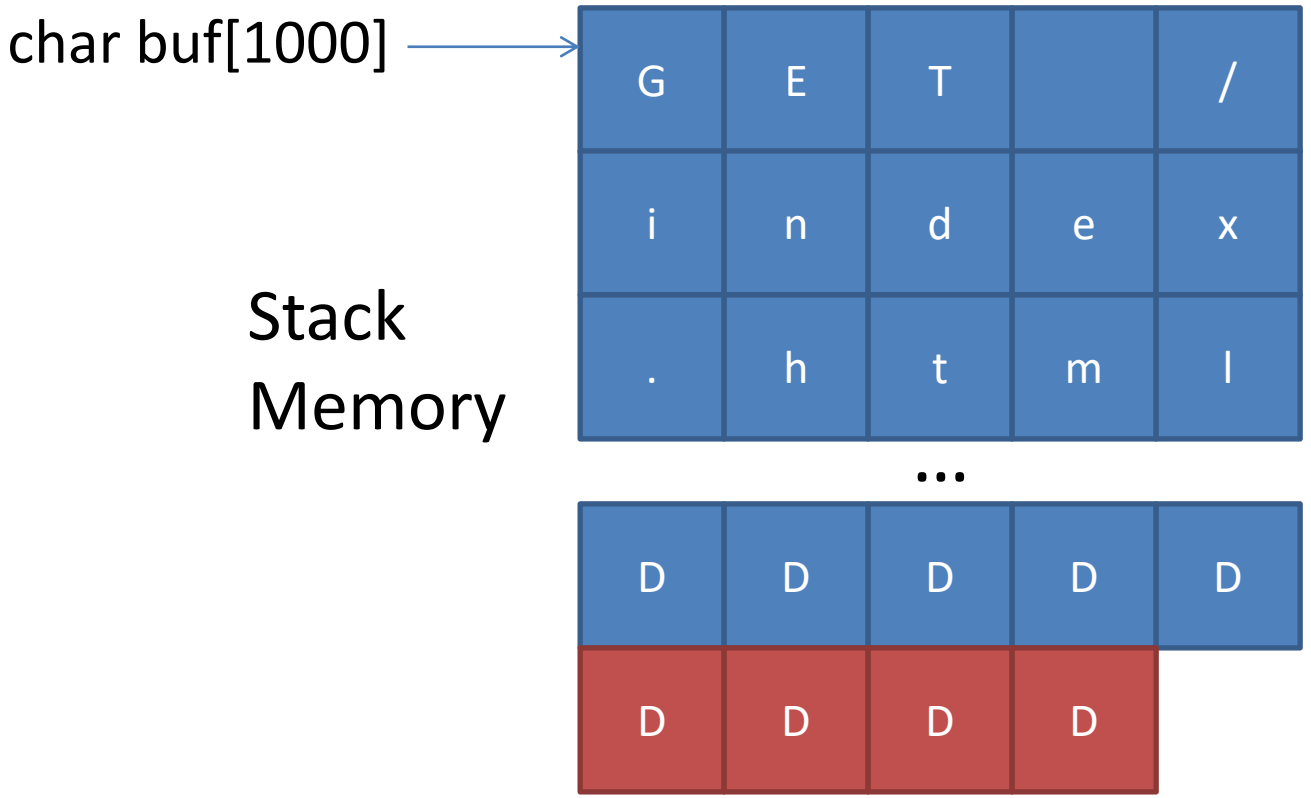
A well intentioned program…

- What happens if we're sent more than 1000 bytes before we see \r\n\r\n? Keep writing…



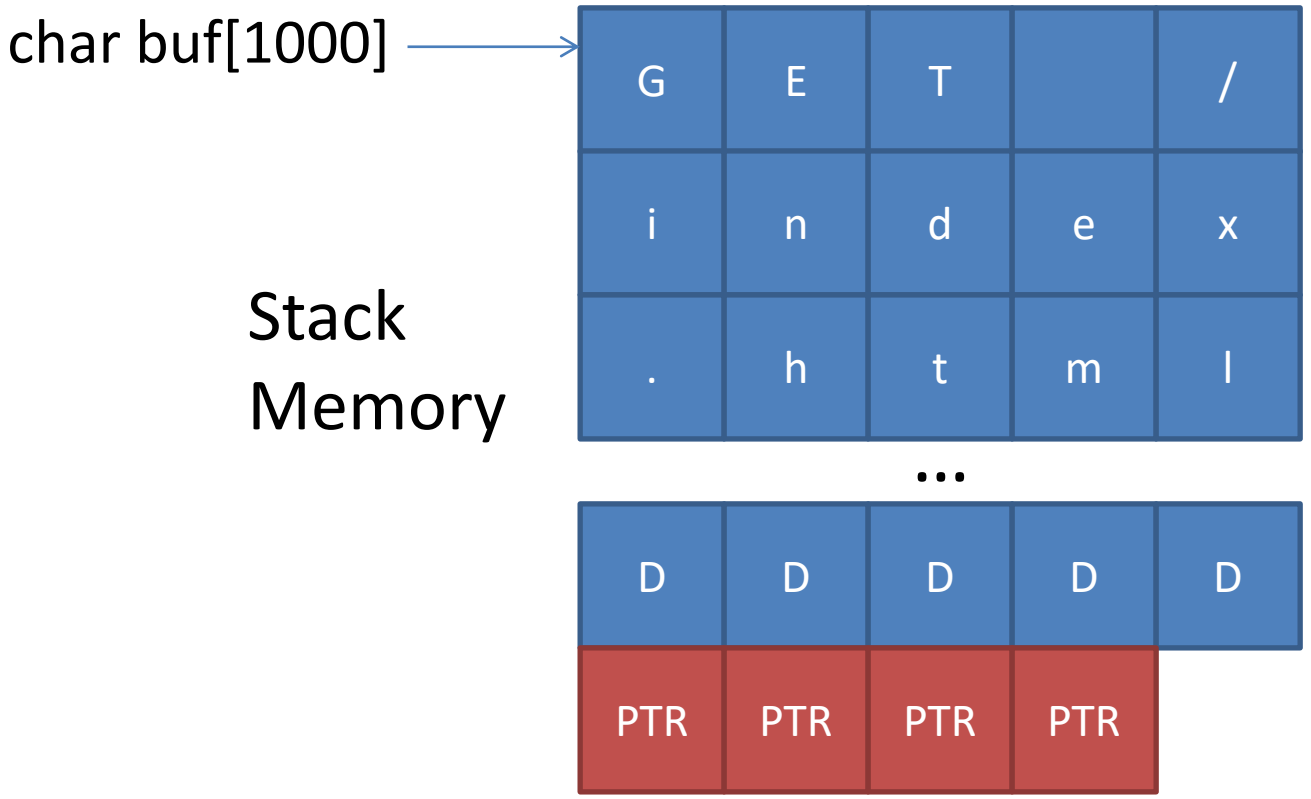
A well intentioned program…

- Uh, if we can overwrite the return address…
- We can control execution on return.



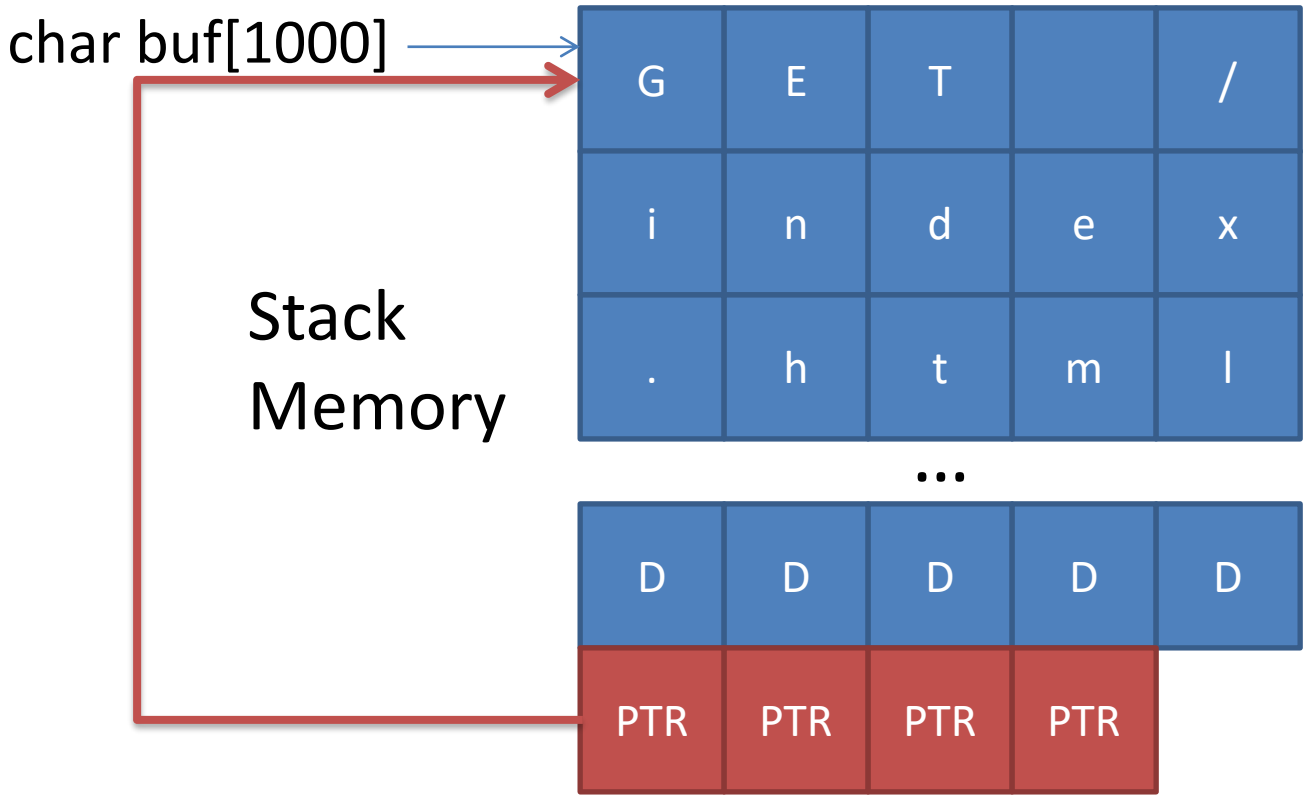
A well intentioned program…

- Let’s send malicious data that contains a ptr.



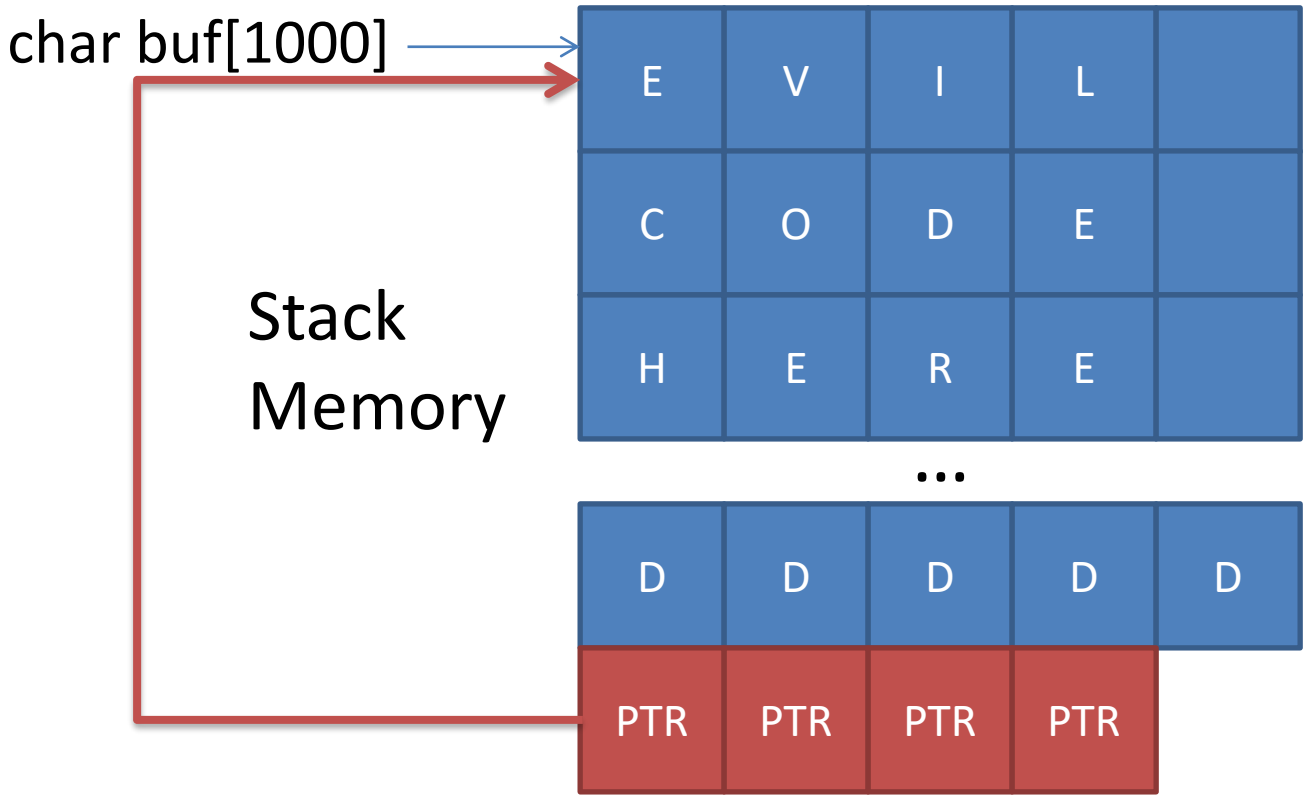
A well intentioned program…

- Let’s send malicious data that contains a ptr.



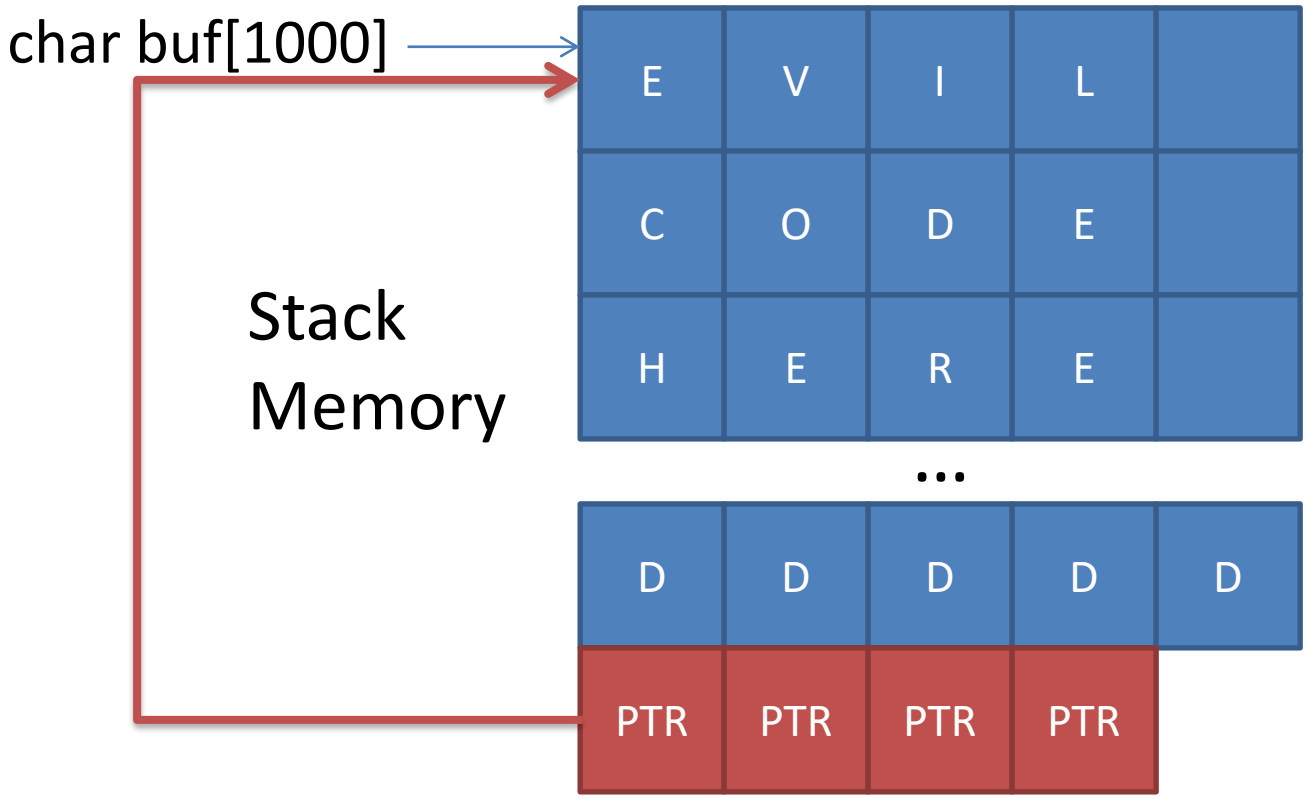
A well intentioned program…

- Oh, and also some commands up here…



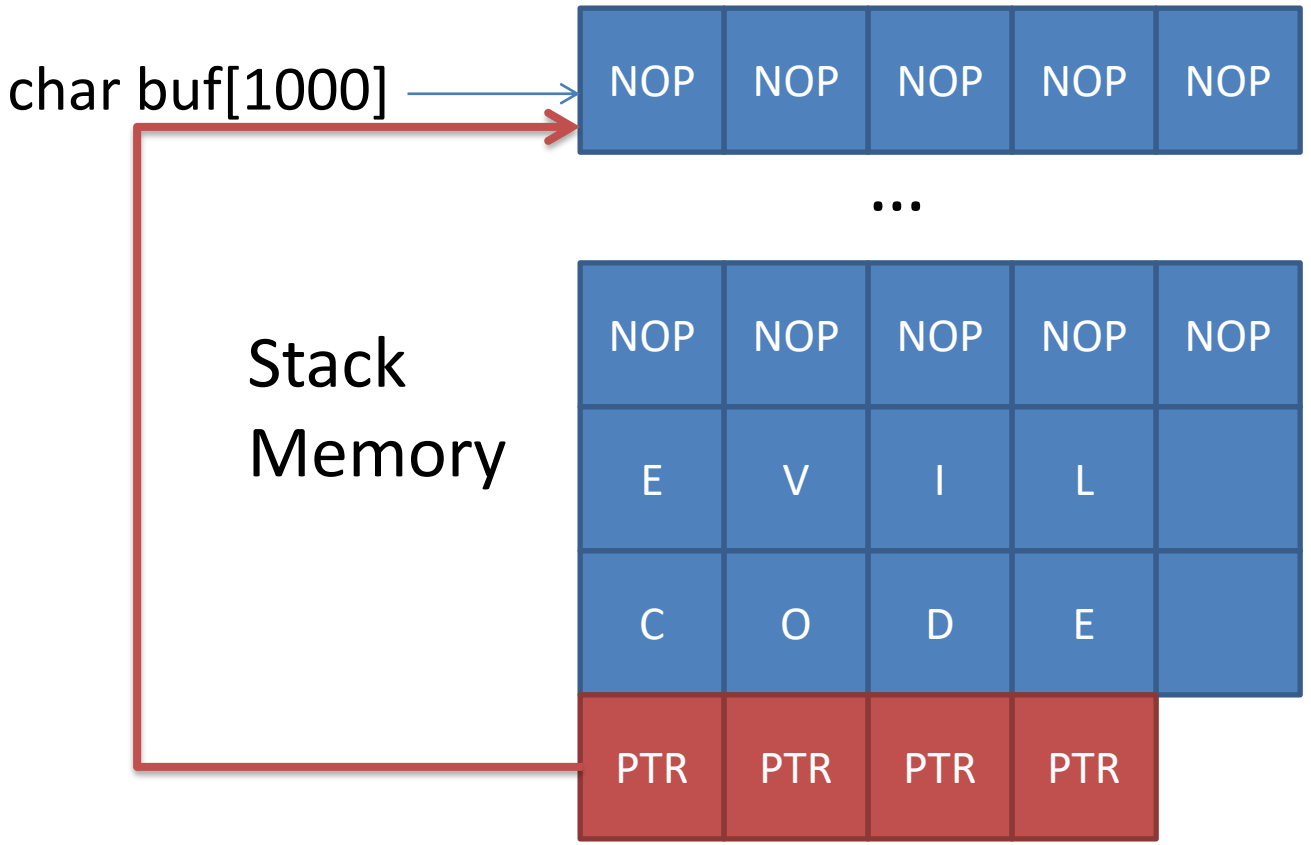
A well intentioned program…

- Function returns, executes evil code.



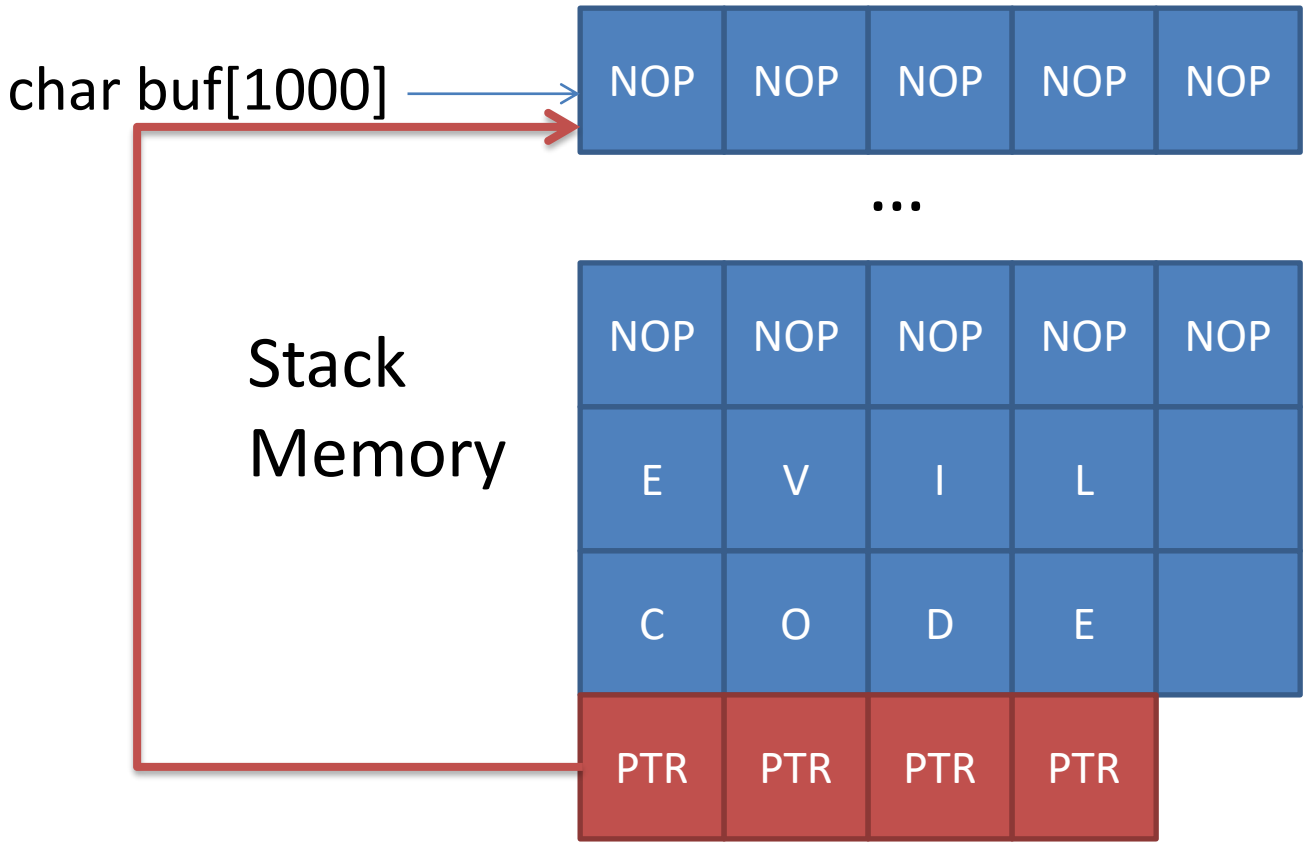
A well intentioned program…

- Improve chances: “NO OP sled”



A well intentioned program…

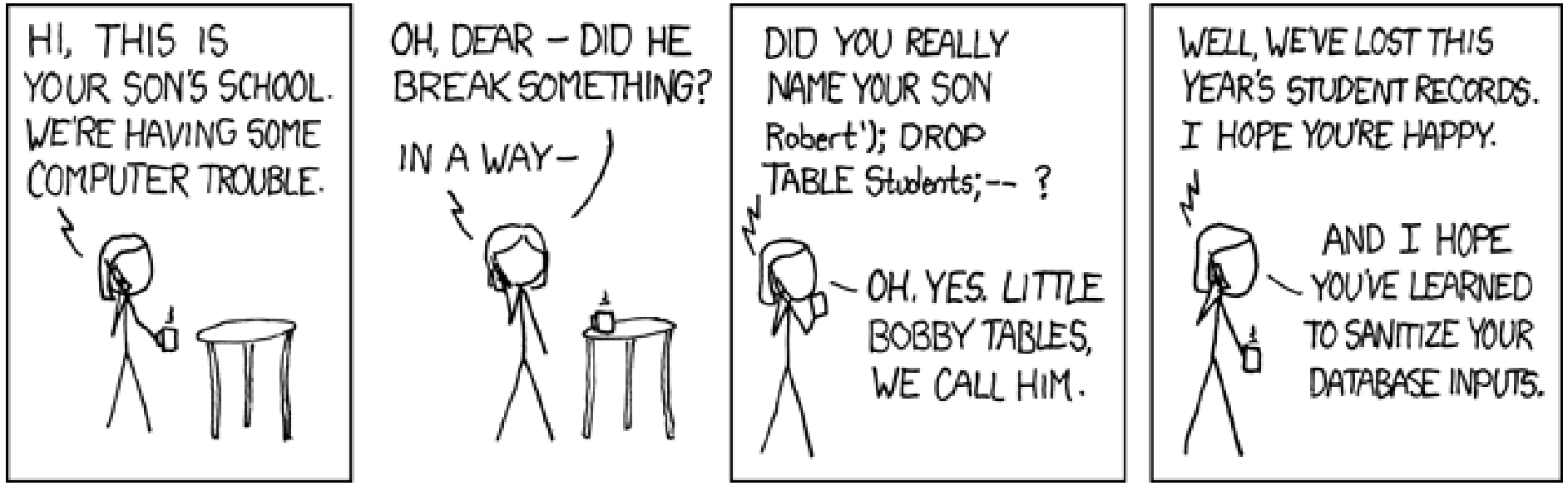
- See: “Smashing the Stack for Fun and Profit”



Input from Users

- Programs that receive user input are susceptible to buffer overflow (& more) attacks.
- Potentially much more problematic to receive input from the Internet!
- If attackers can take over program's control flow, they can execute *anything.*

Relevant XKCD #327



Alt text: Her daughter is named Help I'm trapped in a driver's license factory.

Bottom line: be careful about what you're accepting from users!
Make sure the memory you're using is bounded and that the data is valid!

1988: The Morris Worm

- Cornell student Robert Morris
- Exploited buffer overflow in fingerd
  - It had a 512-byte buffer, he exploited it to execute /bin/sh, giving him shell access
- Told compromised host to download his worm code, it self-replicated by exploiting others
- Claimed “wanted to gauge size of Internet”

1988: The Morris Worm

- Worm did a check to see if it needed to replicate itself
  - If machine already compromised (process running) don't infect again.
- Worried about admins putting up fake process
  - Replicate anyway, at random, 1/7 times.
- This effectively shut down LOTS of machines.

1988: The Morris Worm

- Robert Morris:
  - First person convicted under Computer Fraud and Abuse Act
  - Sentenced to three years probation, 400 hours community service, $10,000
- Where is he now?

Another Famous Buffer Overflow

- https://en.wikipedia.org/wiki/Blaster_(computer_worm)

- There are MANY similar examples from ~2000 – 2005.

My Console for a Horse…

- Nintendo Wii game: The Legend of Zelda: Twilight Princess
- You get to name your horse (Epona).
- Data saved to memory card, loaded on restart of game.
- Horse name length NOT checked!

See: http://wiibrew.org/wiki/Twilight_Hack

Final Example…

- More recently:

- https://arstechnica.com/gaming/2018/04/the-unpatchable-exploit-that-makes-every-current-nintendo-switch-hackable/

What can we do to prevent buffer overflow attacks?

- As the programmer?
- As the OS developer?
- In the compiler or hardware?

Buffer Overflow Mitigation

- Option 1: Stack Canary
  - Generate a random “canary” value at program start (unknown to potential attackers). Put it on the stack before each return address.
  - Before returning, always verify that canary is intact.
  - If canary is changed, terminate the process.

- Pros: no special help from OS or hardware, only compiler
- Cons: arbitrary execution => denial of service, requires recompile

Buffer Overflow Mitigation

- Option 2: W^X (“W xor X”)
  - Enforce in OS & hardware: a page in memory can be writable or executable, but not both (at the same time).

- Pros: enforced by system, prevents stack execution
- Cons:
  - Requires help from hardware (mostly available now)
  - There are *some* legit cases where we want writable, executable memory (e.g., JIT javascript compilers in browsers).

How can we *still* be evil?

- Historically, arbitrary execution requires two major steps:
  1. Get evil instructions into target/victim program.
  2. Cause victim to execute those instructions (alter control flow).

- Let's assume we're all using W^X, which makes #1 (mostly) impossible.

How can we *still* be evil?

- Historically, arbitrary execution requires two major steps:
  1. Get evil instructions into target/victim program.
  2. Cause victim to execute those instructions (alter control flow).

- Let's assume we're all using W^X, which makes #1 (mostly) impossible.

- Is this game over, or can we still perform arbitrary execution?

Return-Oriented Programming

- Proof of concept in 1997, built practically in 2007.
- Super clever idea: all the code you could possibly need is *already available in the target application*.
- Most common variant: “return into libc”: the C library is, by itself, large enough to have any code attacker might want.

In other words, we don’t need to inject evil instructions. All we really need is to hijack the control flow and find the existing instructions we need to perform our evil work.

Recall: Functions in Assembly

```
function:
    pushl %ebp
    movl %esp, %ebp

    # Function body here

    leave
    ret
```

Set up the stack frame for this function.

In libc, the instructions in the body implement library calls…

Restore caller's %esp, %ebp.

Return from function

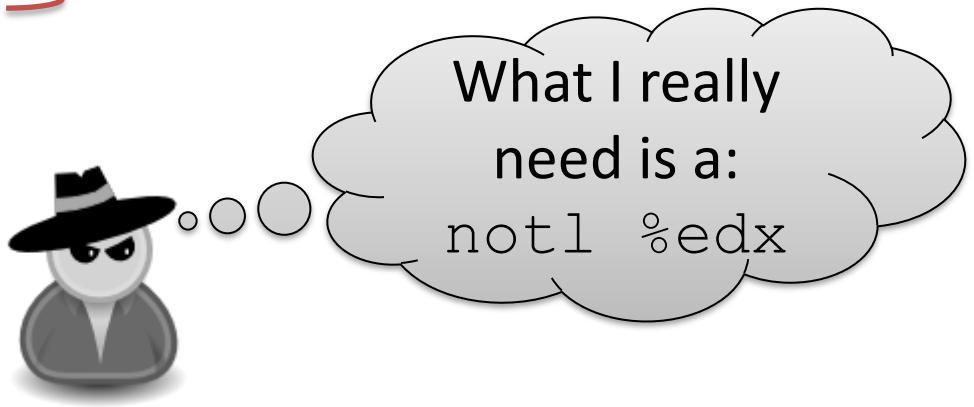
Return-Oriented Programming

```
function:
        pushl %ebp
        movl %esp, %ebp
        incl %ecx
        …
        movl (%ecx), %eax
        notl %edx
        addl %ecx, %eax
        leave
        ret
```

Force jump to here:

Quickly return:

Set up the stack frame for this function.



Restore caller's %esp, %ebp.

Return from function

Return-Oriented Programming

```
function:
    pushl %ebp
    movl %esp, %ebp
    incl %ecx
    …
    movl (%ecx), %eax
    notl %edx
    addl %ecx, %eax
    leave
    ret
```

- In general: find all the ‘ret’ instructions in C library, and what happens before each one.
- Assumption: we know or can determine version of victim’s C library code.
- Determine “gadgets”: ways of stringing together small pieces of functions to achieve arbitrary goals.

Return-Oriented Programming

```
function:
    pushl %ebp
    movl %esp, %ebp
    incl %ecx
    …
    movl (%ecx), %eax
    notl %edx
    addl %ecx, %eax
    leave
    ret
```

- Hijack control flow, execute gadgets.
- There's enough code in libc to build a Turing-complete set of gadgets:
  - conditionals
  - loops
  - variables
  - etc.

How can we defend against ROP?

ROP Mitigation

- Option 1: Address Space Layout Randomization (ASLR):
  - Don’t load library code into known locations, mix up the code region of the virtual address space.
  - Goal: Attacker won’t know which addresses to jump to for gadgets.

- Requires OS support (to randomize when process starts up).
- Requires complier support (to build code that can be place anywhere in memory, at random). (position-independence)

ROP Mitigation

- Option 2: Control Flow Integrity (CFI)
  - Intuition: only allow jumps to pre-approved locations (e.g., the start of a legit function).
  - Keep a structure to record valid jump locations. On each function call and return, check structure for jump target location.
  - If target not valid, terminate process.

Control Flow Integrity

- Pro: no control flow hijacking!

- Cons:
  - performance hit from checking every call/return
  - turns arbitrary execution into denial of service
  - must recompile applications

State of the Art

- Buffer overflows are still out there, despite mitigations.
- The arms race continues between attackers, defenders.
- ASLR is becoming common, but not as much as you'd like.
- CFI gaining some traction, but not widely adopted…

State of the Art

- Buffer overflows are still out there, despite mitigations.

- In the meantime, *many* applications still vulnerable, and we're not just talking about video game horses…

“Comprehensive Experimental Analyses of Automotive Attack Surfaces”

- Modern cars have a network (“CAN Bus”) that connects basically every device in the vehicle.
  - engines, lights, stereo, locks, brakes, bluetooth, cellular (onstar)

http://www.autosec.org/pubs/cars-usenixsec2011.pdf

"Comprehensive Experimental Analyses of Automotive Attack Surfaces"

Vulnerability Class	Channel	Implemented Capability	Visible to User	Scale	Full Control	Cost	Section
Direct physical	OBD-II port	Plug attack hardware directly into car OBD-II port	Yes	Small	Yes	Low	Prior work [14]
Indirect physical	CD	CD-based firmware update	Yes	Small	Yes	Medium	Section 4.2
	CD	Special song (WMA)	Yes*	Medium	Yes	Medium-High	Section 4.2
	PassThru	WiFi or wired control connection to advertised PassThru devices	No	Small	Yes	Low	Section 4.2
	PassThru	WiFi or wired shell injection	No	Viral	Yes	Low	Section 4.2
Short-range wireless	Bluetooth	Buffer overflow with paired Android phone and Trojan app	No	Large	Yes	Low-Medium	Section 4.3
	Bluetooth	Sniff MAC address, brute force PIN, buffer overflow	No	Small	Yes	Low-Medium	Section 4.3
Long-range wireless	Cellular	Call car, authentication exploit, buffer overflow (using laptop)	No	Large	Yes	Medium-High	Section 4.4
	Cellular	Call car, authentication exploit, buffer overflow (using iPod with exploit audio file, earphones, and a telephone)	No	Large	Yes	Medium-High	Section 4.4

http://www.autosec.org/pubs/cars-usenixsec2011.pdf

Detailed Examples:

- Stefan Savage, one of the authors:
- https://www.youtube.com/watch?v=oiFnjuOYz3k&t=629

Other Common Remote Execution Exploits

- Trojans (trick user into downloading, running)
- Browser exploits (drive-by downloads)

Other Common Remote Execution Exploits

- Trojans (trick user into downloading, running)
- Browser exploits (drive-by downloads)

- Great, so you've figured out how to execute arbitrary code on lots of computers out there. How do you benefit from that?

BotNets

- Having access to 1000’s of machines is lucrative!

- Flood target with traffic (DDoS).
- Hold files hostage with ransomeware.
- Steal data (CC #’s, state secrets, etc.).
- Mine bitcoins.
- Send Spam.

Spam “Affiliate Programs”

- https://www.cs.uic.edu/~ckanich/papers/spamalytics.pdf
- https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final204.pdf

Security and Trust

- “Reflections on Trusting Trust” by Ken Thompson
  - https://dl.acm.org/citation.cfm?id=358210

- Turing award acceptance speech in 1984.
  - Hid a “back door” in Unix login program and compiler.

Summary

- Many different common classes of attacks:
  - social engineering, denial of service, information leakage, arbitrary code execution
- Security: battle of wits between attackers and defenders.
  - Defenders make attacking difficult, but systems often deployed slowly.
- Evil can be lucrative, but it makes you a criminal…