CS1632, LECTURE 20: Security Testing

Wonsun Ahn

Writing Secure Software Is Difficult; So Is Testing It!

- **Heartbleed**: A defect in OpenSSL
  - Caused ~ 66% of servers connected to the Internet to be vulnerable
  - Allowed for untraceable eavesdropping on data in memory
  - Discovered in 2014, vulnerability introduced in 2012
- **Shellshock**: A defect in bash (default shell for OS X and most Linux)
  - Millions of attacks recorded in the days following discovery
  - Allowed arbitrary code execution stored in environment variables
  - Discovered in 2014, vulnerability introduced in 1989

Even Testing Secure *Hardware* is Difficult

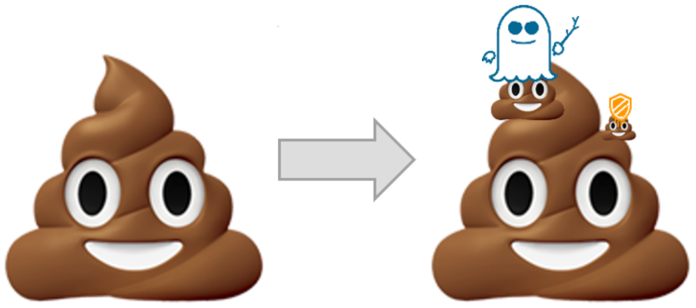
- **Spectre / Meltdown**: A vulnerability in CPU design
  - Impacts all CPUs in wide-use today (Intel, AMD, ARM, IBM …)
  - Allowed arbitrary access to private data in a process (Spectre)
  - Allowed arbitrary access to private data in an OS (Meltdown)
  - Discovered in 2017, vulnerability introduced in 1995
  - OS / Web Browser patches issued but some Spectre vulnerabilities still open

SPECTRE MELTDOWN

A Slide from a 2018 Hardware Design Conference

Risk in context

Because of software bugs, computer security was in a dire situation



Spectre doesn't change the magnitude of the risk, but adds to the mess

- Poor mitigation options (fixes -> new risks)

17

* From ISCA 2018 Panel "**On the Implications of the Meltdown & Spectre Design Flaws**"

Why is it so Difficult?

1. Adversaries are actively seeking to defeat security
2. Information about security vulnerabilities spreads quickly
3. You need to protect all doors;
   They only need to find one they can open
4. Even minor vulnerabilities can have outsized consequences

History

- Security was not a big deal in the early computing world
  - Usually required physical access to a system to do anything
  - Few people had necessary skills even if they did
    (So called “security through obscurity”)

- Hacker culture 1960-80s exemplified in ITS Operating System
  - OS did not use passwords; anyone could use it and do anything
  - There was a flaw where clever users could crash the OS
    - Solution? A “crash” command was created that could be run by anyone
    - Crashing the OS was not challenging or fun anymore → nobody did it

History

- Now the stakes are much higher
  - "Estimating the Global Cost of Cyber Risk", RAND Corp., 2018 https://www.rand.org/pubs/research_reports/RR2299.html
  - Global cost of cyber crime: $799 billion to $22.5 trillion (1.1% to 32.4% of global GDP)

- And there are many more actors …

Actors in the Security Sphere

- White hat hackers (Ethical hackers)
  - Employed by companies to check their own systems, or by a security firm
  - Performs penetration testing and vulnerabilities testing for client
- Black hat hackers (Crackers)
  - Violates system security for personal gain or other malicious purpose
- Red hat hackers (Hacktivists)
  - Works to spread a political / ideological / religious message
- Organized crime (works in conjunction with black hat hackers)
- Nation states (e.g. Stuxnet, Equation Group)

The InfoSec (CIA) Triad

- No, it has nothing to do with *that* CIA
- CIA as in:
  - **C**onfidentiality
  - **I**ntegrity
  - **A**vailability
- A secure system needs to provide these qualities

Confidentiality

No unauthorized users may read data.

Integrity

No unauthorized users may write data.

Availability

System is available for authorized parties to read from and write to.

Terminology: Kinds of Security Attacks

1. *Interruption* (attack on availability)
    - e.g. Pulling plug from network switch, DDoS
2. *Interception* (attack on confidentiality)
    - e.g. Eavesdropping, keylogger
3. *Modification* (attack on integrity)
    - Modifying or deleting data
4. *Fabrication* (attack on integrity)
    - Making up or inserting data

Terminology: Passive vs Active Attacks

- *Passive:* Do not modify system in any way
  - Eavesdropping
  - Monitoring
  - Traffic Analysis
- *Active:* Modify the system in some way
  - Fill up database with garbage data
  - Modify bank account information

Terminology: Vulnerability vs Exploit

- Vulnerability: identified weakness of a system

- Exploit: Mechanism for compromising a system using a vulnerability

Terminology: Kinds of Malicious Code

- **Malware** – General term for malicious code (includes all kinds below)
- **Bacteria** - program that consumes system resources (e.g. fork bomb)
- **Logic bomb** - code within a program which executes an unauthorized function
- **Trapdoor** - secret undocumented access to a system or app
- **Trojan horse** – program that pretends to be another program
- **Virus** - replicates itself WITH human intervention
- **Worm** - replicates itself WITHOUT human intervention
- **Zombie** – A computer or program being run by an unauthorized controller
- **Bot network** – collection of zombies controlled by master
- **Spyware** – surreptitiously monitors your actions
- **Adware** – Shows you more ads
- **DOS** (Denial of service) attacks (e.g. via LOIC)

Protections

- Firewalls
- Operating System Permissions
- CDNs (Content Delivery Networks) – e.g. Akamai
- Cryptography – e.g. HTTPS protocol
- Well-written code
- User training

Some security testing tools

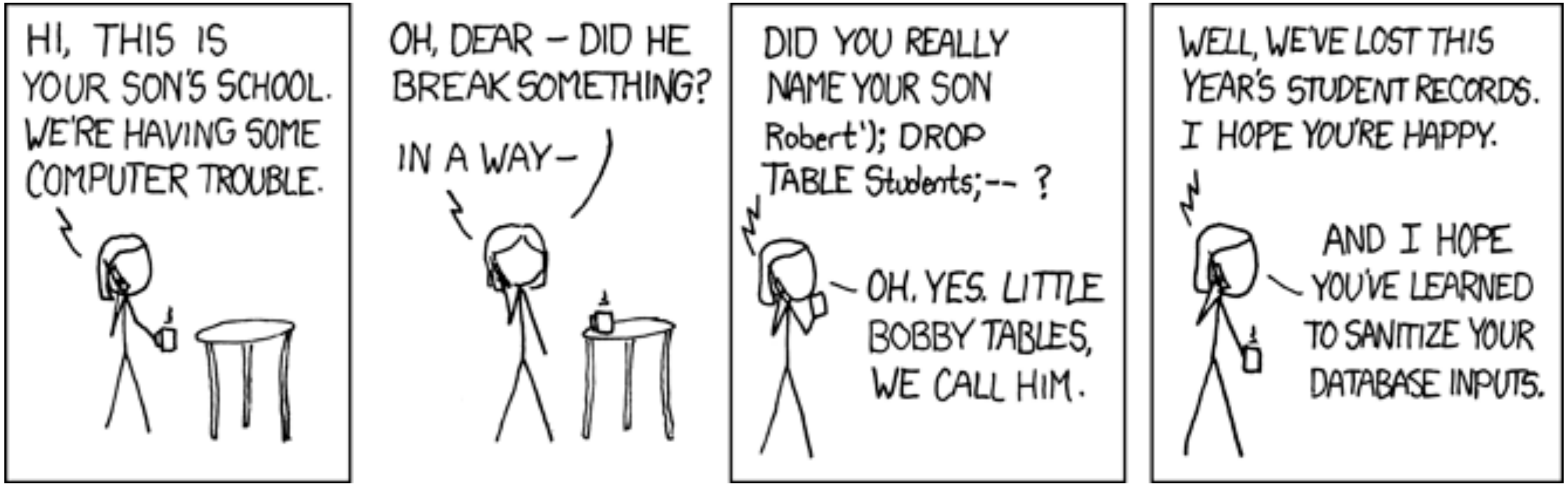
- Nmap – Network Mapper
  - Audits the network for open ports and open services
  - Audits version numbers for all OSes and services
- Metasploit – A penetration testing tool using actual exploits
  - Over 900 exploits for various Oses
  - Includes fuzzing technology to look for unknown software vulnerabilities
- Wireshark – A packet sniffer that displays all network traffic

- Incidentally, these are the same tools attackers use
  - You might as well use them yourselves to test your systems

Common Attacks

- Injection Attacks
- Broken Authentication
- Cross-Site Scripting (XSS)
- Insecure Object References
- Security Misconfiguration
- Insecure Storage
- Buffer overruns
- Social Engineering

Injection Attacks



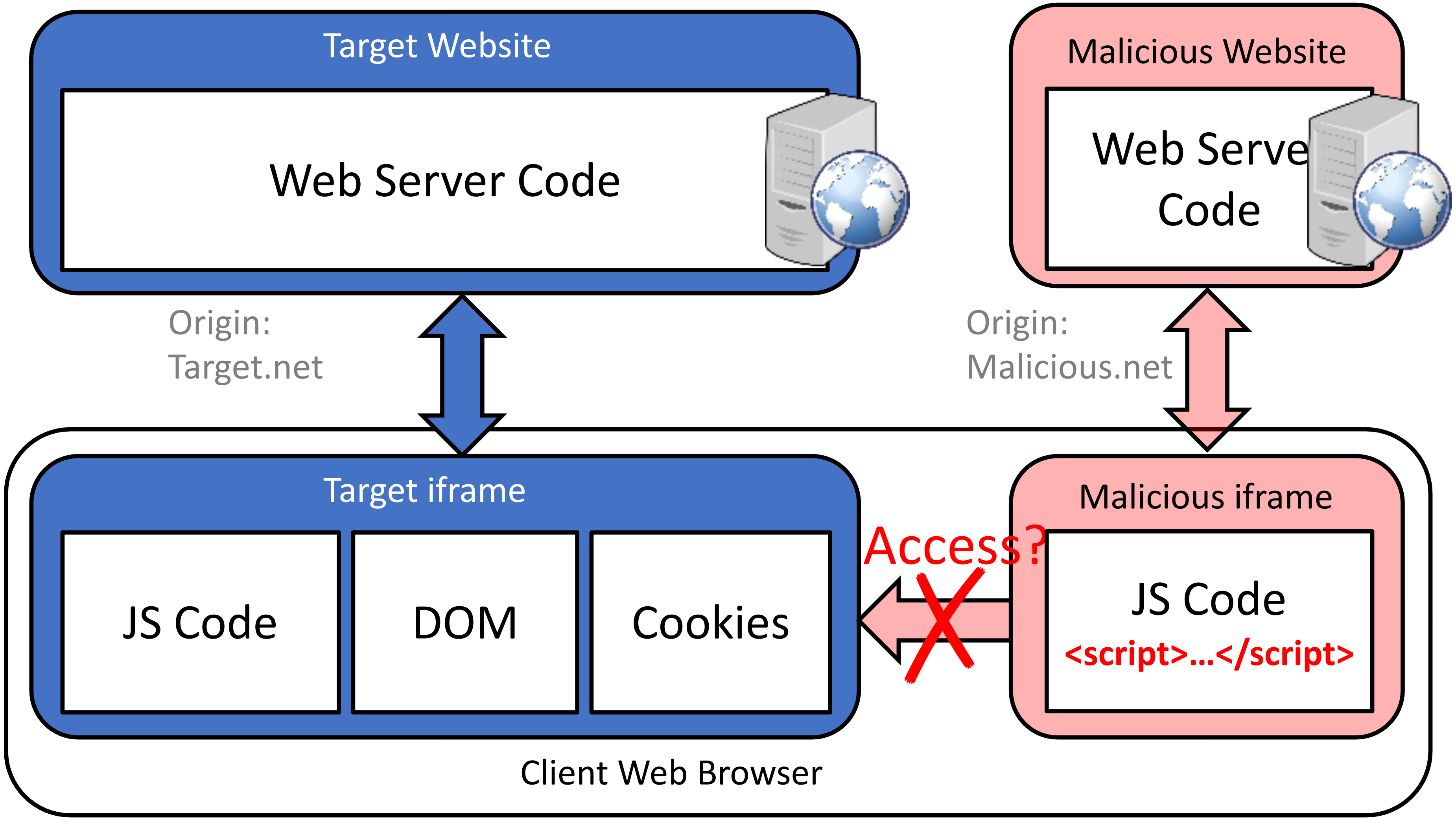
Broken Authentication

- One user pretends to be another
- How?
  - Guess or crack passwords
  - “Password reset”
  - Unencrypted session IDs
- Apple iCloud leak was suspected of being this
  - iCloud API allowed unlimited attempts – allowing a brute force attack
- Sarah Palin email hack was definitely this
  - All he needed to know, he learned from Wikipedia
  - Answered security questions, reset password

Cross-Site Scripting

- 2019 CWE (Common Weakness Enumeration) Top 25: 2^{nd} place
  - The most popular exploit for web apps for over a decade
- To fully understand, need to first understand Same Origin Policy
- Same Origin Policy (SOP): Web browser sandboxing architecture
  - A webpage can access data in another webpage only if from same URL origin
  - Your reddit.com webpage cannot access your onlinebanking.com webpage
  - Same rule applies for frames even if on same webpage
    (e.g. an advertisement in a frame cannot access data in rest of webpage)

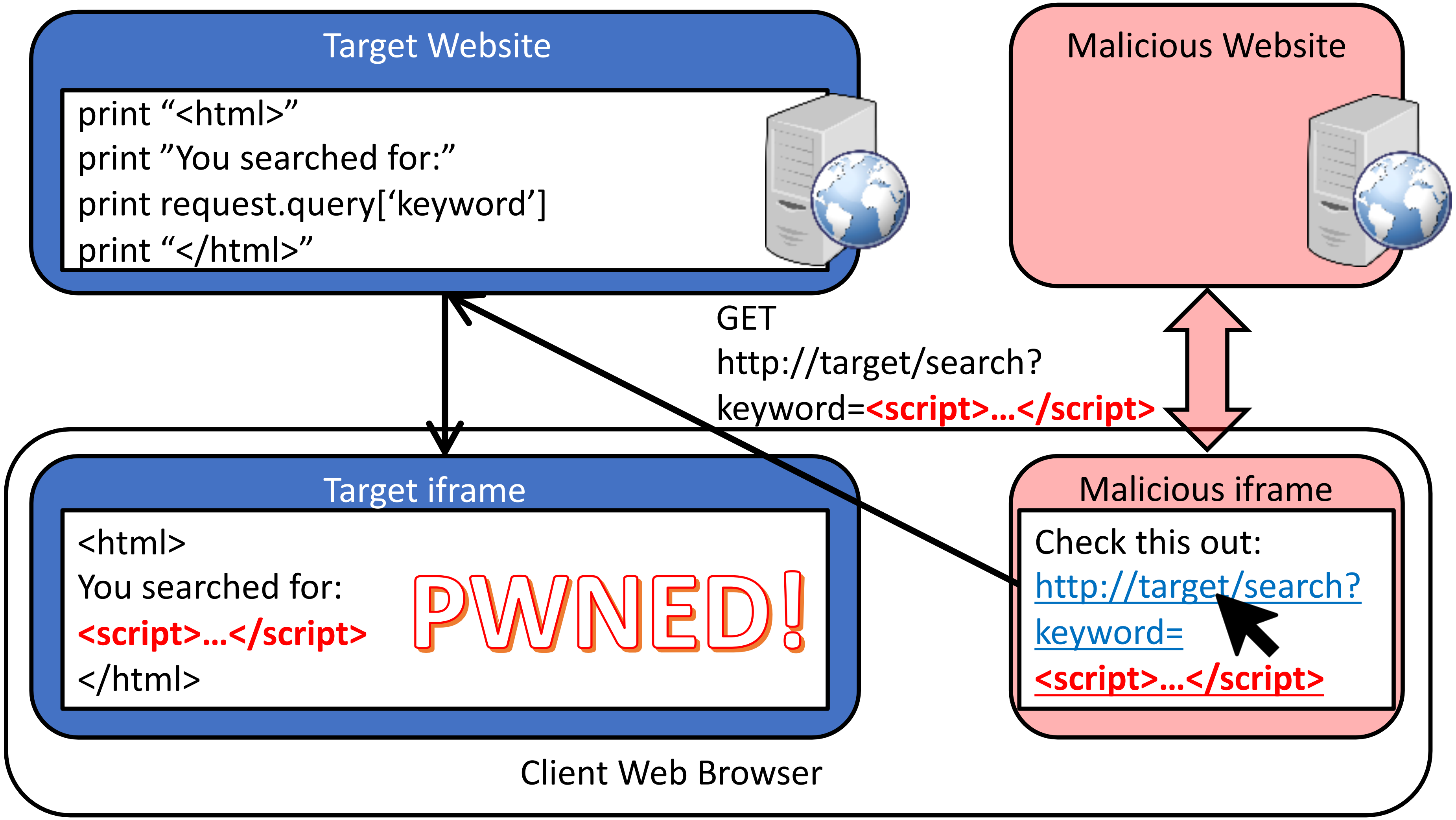
Browser Sandboxing – Same Origin Policy



Cross-Site Scripting

- Allows malicious website to execute (Java)script code
  - Across site boundaries
  - Ignoring SOP protections

Cross Site Scripting



Insecure Object References

- Someone can access something by knowing where it is, despite not having proper security credentials
  - http://bank.com/?account=9844
  - http://bank.com/?account=9845

Security Misconfiguration

- Proper security is available, it’s just not set up correctly!
- Default passwords
- Firewalls with dangerous exceptions
- IPS (Intrusion Prevention System), packet filtering, etc. not running
- Directory listing in web server not disabled
- Web server error log display left on
    - Displays to user Java exception stack trace
    - Displays to user SQL error message

Insecure Storage

- Secure data is stored in an unsafe way
- Credit card numbers being stored in a /tmp or logging directory as part of logging all transactions
- Database being stored with incorrect file permissions, allowing the DB file to be copied wholesale
- Passwords in database not encrypted, or encrypted without salting

Buffer Overrun

- 2019 CWE (Common Weakness Enumeration) Top 25: Winner
  - Consistently within the top 3 for all years since 2009

- Reading or writing past the end of memory allocated for a buffer
  - Doesn’t happen in Java (results in a IndexOutofbounds exception)
  - Doesn’t happen in JavaScript or Python (results in silent expansion of buffer)
  - Only happens in C / C++ / Assembly – allows direct access to memory
  - But a lot of critical system code is written in C / C++, unfortunately

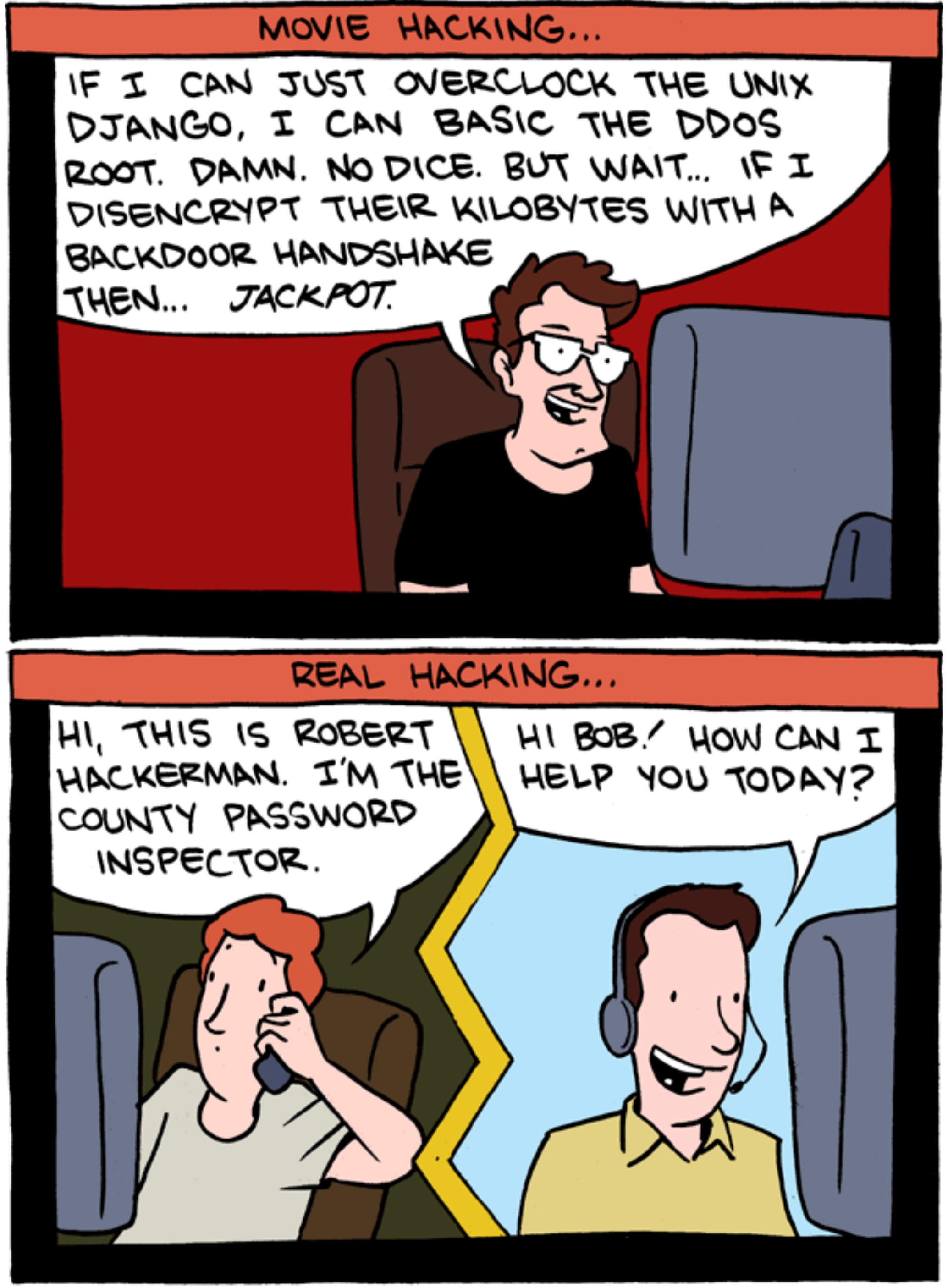
- What Heartbleed was – see heartbleed.c in sample_code directory

heartbleed.c

```c
void bad(int len) {
  char* notSecret = "open data";
  char* secret = "SECRET DATA HERE! NOBODY SHOULD SEE THIS!";
  printf("Sending data:\n");
  for (int j=0; j < len; j++) {
    printf("%c", notSecret[j]);
  }
}
int main() {
  int l;
  puts("Enter length of data:");
  scanf("%d", &l);
  bad(l);
}
```

```
-bash-4.1$ gcc heartbleed.c -o heartbleed
-bash-4.1$ ./heartbleed
Enter length of data:
100
Sending data:
open dataSECRET DATA HERE! NOBODY SHOULD SEE THIS!
Sending data:Enter length of data:%d
```

Social Engineering



For a More Comprehensive List …

- CWE (Common Weakness Enumeration) Top 25:
  - https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html
  - By MITRE Corp. which maintains CVE (Common Vulnerabilities and Exposures) DB

- OWASP (Open Web Applications Security Project) Top 10 Project:
  - https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
  - Top 10 security vulnerabilities for web applications over the years

- OWASP attacks page:
  - https://www.owasp.org/index.php/Category:Attack
  - Contains testing guides on how to test for those vulnerabilities

Pittsburgh – A Great City To Learn About Security!

- Many security researchers here at Pitt and CMU
  - LERSAIS at Pitt SCI – Laboratory for Education & Research on Security-Assured Information System
  - Pitt Cyber Institute
  - CyLab at CMU
- SEI (Software Engineering Institute)
- CERT (Computer Emergency Response Team)
- Many security engineering positions (esp. at banks)

Now Please Read Textbook Chapter 20

… and this ends all official lectures.

- Are you sad? Here are some extra slides for you:
  - Note: Slides following will not appear in the exam

- Monday (Dec. 2), we will have an exam review

Supplemental Security Slides

How far are exploits willing to go?

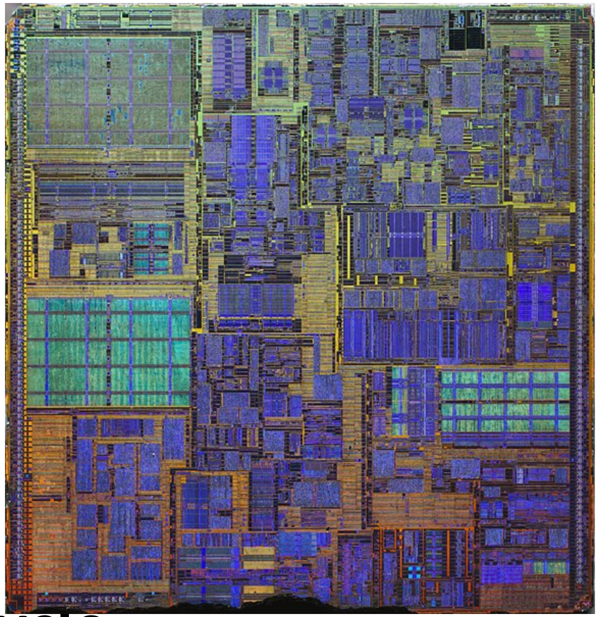
Spectre & Its Root Causes

Paul Kocher
(paul@paulkocher.com)

ISCA
June 4, 2018

If the surgery proves unnecessary, we'll revert your architectural state at no charge.

Addicted to speed



Performance goal

- Lowest time to reach the result same as running program in-order

Single-thread speed gains require getting **more done per clock cycle**

- Memory latency is slow and not improving much
- Clock rates are maxed out: Pentium 4 reached 3.8 GHz in 2004

- How to do more per clock?
  - Reducing memory delays → Caches
  - Working during delays → Speculative execution

Public domain image of Pentium 4 die by Ritzchens Fritz

Memory caches for dummies

- Caches hold local (fast) copy of recently-accessed 64-byte chunks of memory

CPU
Sends address,
Receives data

Addr: 2A1C0700
Data: 9E C3 DA EE B7 D3..

Addr: 132E1340
Data: AC 99 17 8F 44 09..

Addr: 132E1340
Data: AC 99 17 8F 44 09..

Fast
Slow
Fast

MEMORY CACHE

h(addr) to map to cache set

Set	Addr	Cached Data ~64B
0	F0016280	B5 F5 80 21 E3 2C..
	31C6F4C0	9A DA 59 11 48 F2..
	339DD740	C7 D7 A0 86 67 18..
	614F8480	17 4C 59 B8 58 A7..
1	71685100	27 BD 5D 2E 84 29..
	132A4880	30 B2 8F 27 05 9C..
	2A1C0700	9E C3 DA EE B7 D9..
	C017E9C0	D1 76 16 54 51 5B..
2	311956C0	0A 55 47 82 86 4E..
	002D47C0	C4 15 4D 78 B5 C4..
	91507E80	60 D0 2C DD 78 14..
	55194040	DF 66 E9 D0 11 43..
3	9B27F8C0	84 A0 7F C7 4E BC..
	8E771100	3B 0B 20 0C DB 58..
	A001FB40	29 D9 F5 6A 72 50..
	132E1340	AC 99 17 8F 44 09..
4	6618E980	B0 FC 5A 20 D0 7F..
	BA0CDB40	1C 50 A4 F8 EB 6F..
	89E92C00	1C 50 A4 F8 EB 6F..
	090F9C40	BB 71 ED 16 07 1F..

Address: 132E1340

Data: AC 99 17 8F 44 09..

MAIN MEMORY

Big, slow
e.g. 16GB SDRAM

Reads change system state:
- Next read to newly-cached location is faster
- Next read to evicted location is slower

Speculative execution

Example of speculative execution:

```
if (uncached_value_usually_1 == 1)
    foo()
```

- Branch predictor: if() will probably be 'true' (based on prior history)
- CPU starts foo() speculatively -- but doesn't commit changes
- When value returns, changes committed only when value is actually '1'

Violates software security requirement that the CPUs runs instructions correctly.

Regular execution

Set up the conditions so the processor will make a desired mistake

Fetch the sensitive data from the covert channel

Erroneous speculative execution

Mistake leaks sensitive data into a covert channel (e.g. state of the cache)

Conditional branch (Variant 1) attack

```
if (x < array1_size)
   y = array2[array1[x]*4096];
```

Assume code in kernel API, where unsigned int `x` comes from untrusted caller

Execution <u>without</u> speculation is safe

- CPU will not evaluate `array2[array1[x]*4096]` unless `x < array1_size`

What about with speculative execution?

Conditional branch (Variant 1) attack

```
if (x < array1_size)
   y = array2[array1[x]*4096];
```

Before attack:

- Train branch predictor to expect if() is true (e.g. call with `x` < `array1_size`)
- Evict `array1_size` and `array2[]` from cache

Memory & Cache Status

`array1_size = 00000008`

Memory at `array1` base address:

8 bytes of data (value doesn't matter)

[... lots of memory up to `array1` base+N...]

`09 F1 98 CC 90...` (something secret)

```
array2[ 0*4096]
array2[ 1*4096]
array2[ 2*4096]
array2[ 3*4096]
array2[ 4*4096]
array2[ 5*4096]
array2[ 6*4096]
array2[ 7*4096]
array2[ 8*4096]
array2[ 9*4096]
array2[10*4096]
array2[11*4096]
...
```

Contents don't matter only care about cache ***status***

Uncached

Cached

Conditional branch (Variant 1) attack

```
if (x < array1_size)
    y = array2[array1[x]*4096];
```

Attacker calls victim with x=N (where N > 8)

- Speculative exec while waiting for array1_size
  - Predict that if() is true
  - Read address (array1 base + x) w/ out-of-bounds x
  - Read returns secret byte = **09**
  - Request memory at (array2 base + **09***4096)
  - Brings array2[**09***4096] into the cache
  - Realize if() is false: discard speculative work
- Finish operation & return to caller

Attacker measures read time for array2[i*4096]

- Read for i=**09** is fast (cached), revealing secret byte
- Repeat with many x (eg ~10KB/s)

Memory & Cache Status

array1_size = 00000008

Memory at array1 base address:
8 bytes of data (value doesn't matter)
[... lots of memory up to array1 base+N...]
09 F1 98 CC 90... (something secret)

array2[0*4096]
array2[1*4096]
array2[2*4096]
array2[3*4096]
array2[4*4096]
array2[5*4096]
array2[6*4096]
array2[7*4096]
array2[8*4096]
array2[9*4096]
array2[10*4096]
array2[11*4096]
...

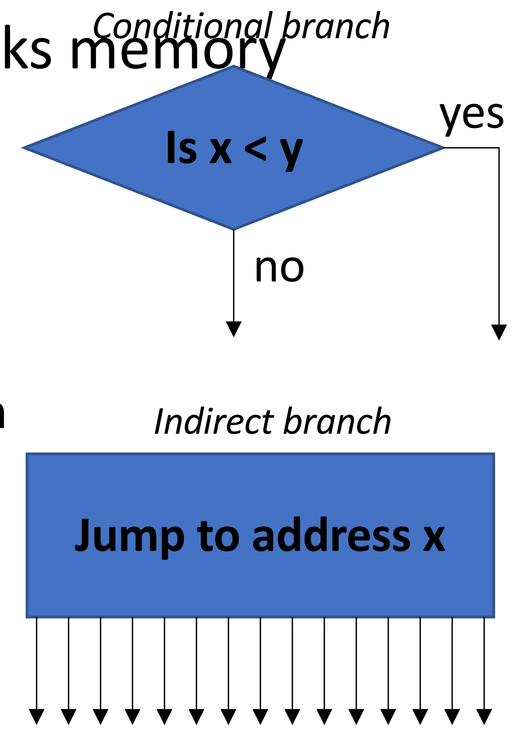
Contents don't matter only care about cache ***status***

Uncached   Cached

Indirect branches (Variant 2)

Can go anywhere instantly (“jmp [rax]”)

- Poison predictor so victim speculative executes a ‘gadget’ that leaks memory



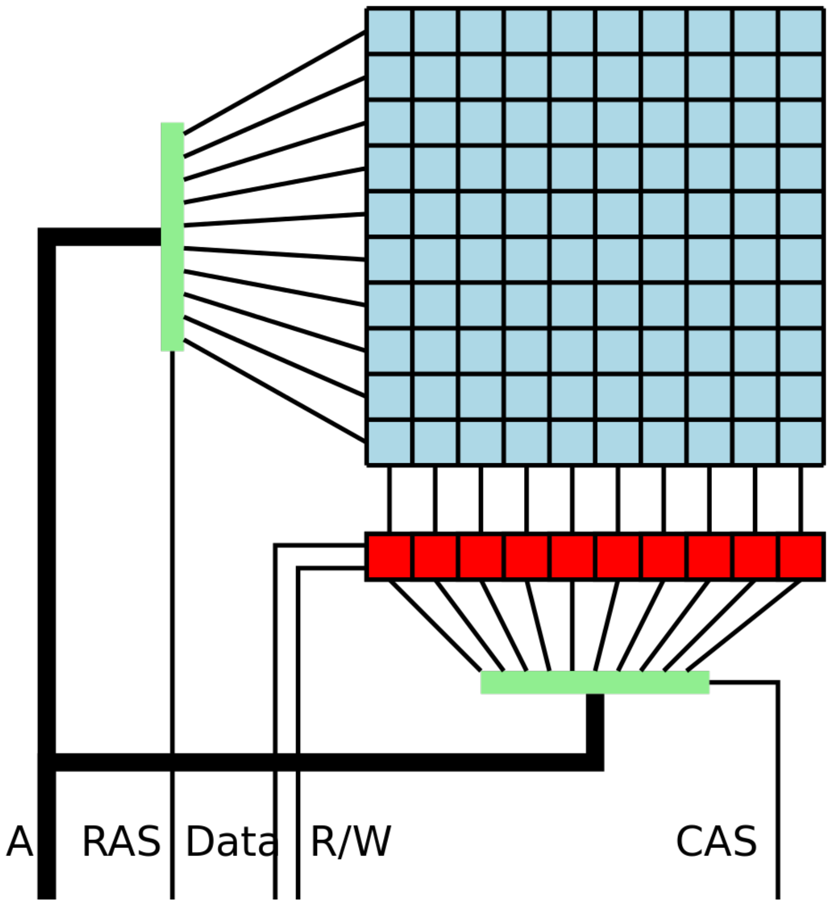
- Attack steps
  - **<u>Poison</u>** branch predictor/BTB so speculative execution will go to gadget
  - **<u>Evict</u>** from the cache or do other setup to encourage speculative execution
  - **<u>Execute</u>** victim so it runs gadget speculatively
  - **<u>Read</u>** sensitive data from covert channel
  - **<u>Repeat</u>**

Row Hammer Exploit

Discovered by Google Project Zero, 2015

DRAM Organization

- Each square in matrix is a memory cell
  - Stores one bit of memory
  - Basically a capacitor holding a charge
- RAS (Row Address Strobe) selects row
  - Row is stored in row buffer (in red)
- CAS selects column, the specific bit
- Memory cell leaks charge over time
  - Needs refresh every 200 ms or so
  - Refresh recharges capacitors



Row Hammer Exploit

- Suppose purple line contains a password, the target of the exploit
- Keep hammering the neighboring yellow lines with reads
- Rapid voltage fluctuations of RAS lines cause purple line to lose charge faster
- Cells in purple line become zeroed out before getting a chance for refresh
- Password is now 00000000