Speculative Execution Attacks: Technical Background of the Spectre and Meltdown Vulnerabilities

Dr. Robert Marmorstein
(marmorsteinrm@longwood.edu)

June 5th, 2018

About Me

- Associate Professor of Computer Science at Longwood University
- BA in Mathematics and Computer Science from Washington & Lee

- PhD and MS in Computer Science from William & Mary

Timeline[1]

- **June 2017:** KAISER patches[2] are introduced to Linux

November 2017: Linux community notices that something strange is going on with KAISER patches

December 2017: Apple pushes updates to Mac OS X and iOS

January 2018: Google‘s Project Zero announces the Spectre and Meltdown vulnerabilities

1. Jan Wildeboer, "How we got to #Spectre and #Meltdown", Jan 5. 2018 https://plus.google.com/+jwildeboer/posts/jj6a9JUaovP

2. Daniel Gruss, et. al, https://github.com/IAIK/KAISER

Why do we care?

- **Process Separation**
  - Passwords, PII, and other sensitive data can leak
  - Compromised web applications can leak database or file data
- **Virtual Machines and Containers**
  - Share physical memory
  - Provide isolation
  - Spectre and Meltdown break the sandbox

Page Tables

- Map virtual pages to physical pages

  paddr = PT[vpn]*PageSize + offset

- Stored per-process

- Typically stored as a hierarchy
  - Linux uses a three-level page table

- Cached by the TLB (translation look-aside buffer)

- Contain permission bits that identify which processes can access each page.

Side Channel Attacks

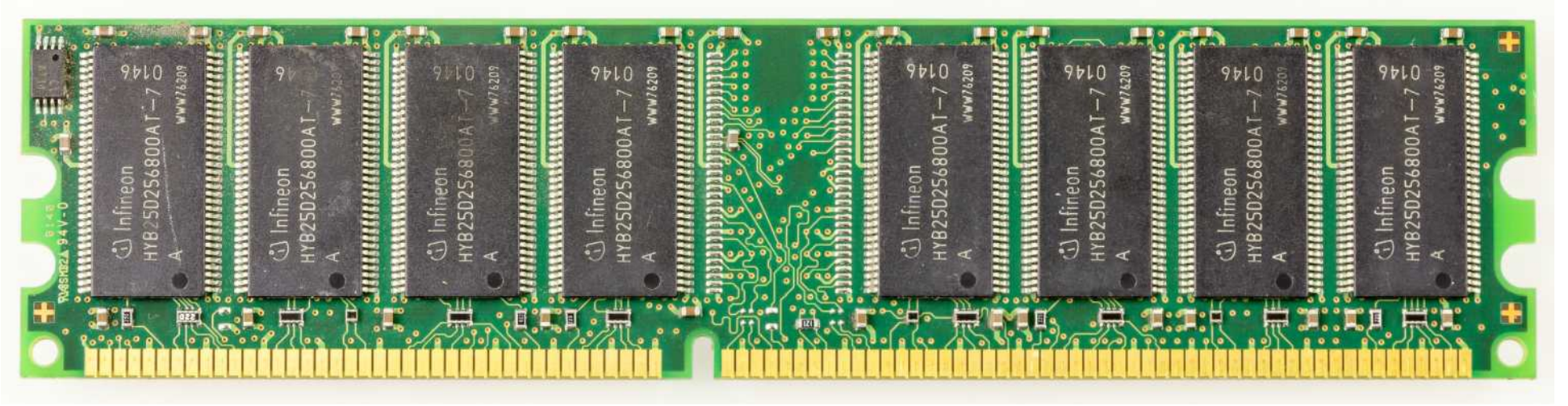
- What is a side-channel attack?
  - A means of circumventing software security protections using physical properties of the hardware
  - Power usage, temperature, timing, electromagnetic interference
- Example: Cracking RSA keys by timing how long the server processes each candidate key[1]

1. Paul Kocher, "Timing Attacks on the Implementations of Diffie-Hellman, RSA, DSS, and Other Systems“, CRYPTO 96, p. 104-113, August 1996

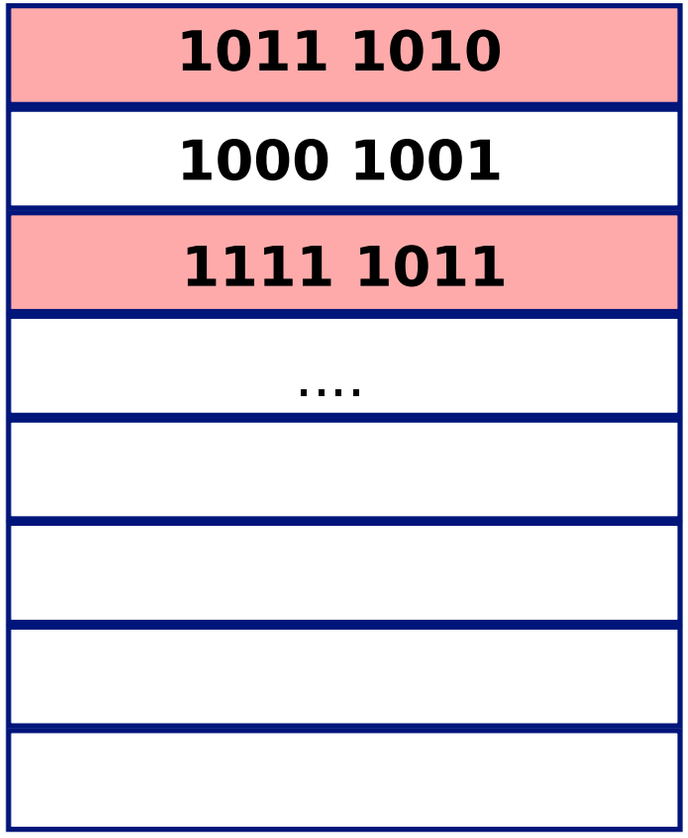
Row Hammer

- Electromagnetic interference can cause DRAM cells to flip from 0s to 1s (or 1s to 0s)
- Certain memory access patterns can exploit this to allow the contents of one row of cells to be modified by accessing another
- Affects only DDR3 and newer, not older hardware

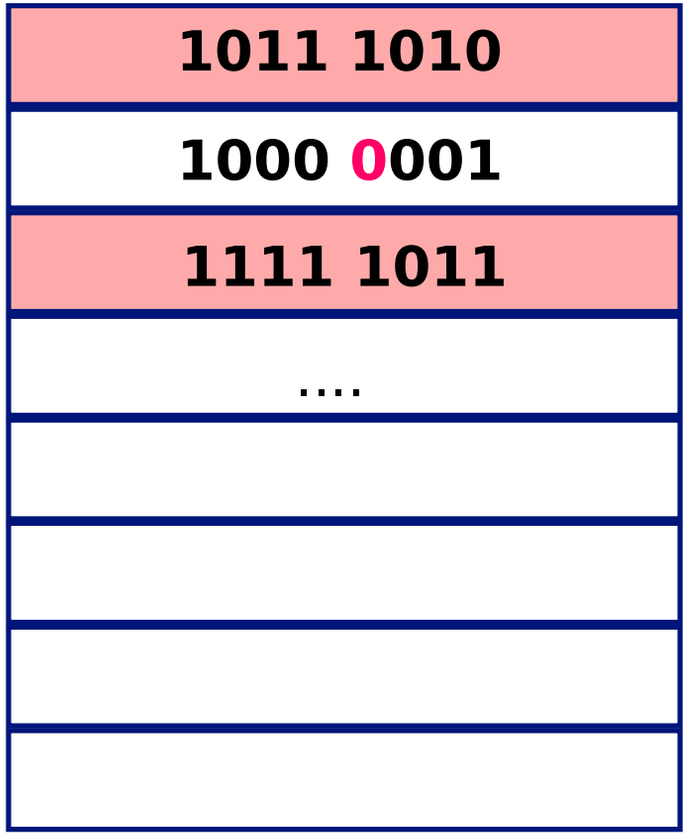
Row Hammer



Row Hammer



Row Hammer



Pipelining

- Allows processor to execute multiple instructions at once
  - Fetch, Decode, Execute, Memory, Writeback
  - Modern processor (Skylake): 20-24 stages

```
mov $4, %eax
    add %ebx, %esi
        mov %ecx, 0xbffffff0
            cmp %edx, %edi
                jne 0x08001020
```

Speculative Execution

- Pipelining produces huge speedups **if you can fetch the correct set of instructions**

- Instruction flow isn't always sequential:
  - Loops and conditional statements can cause branching

- Branch prediction allows a processor to execute a set of instructions and then revoke them if the branch is mispredicted

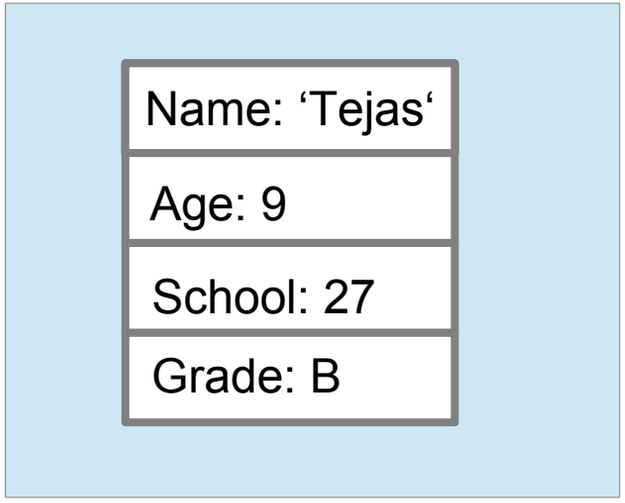
Flush+Reload

Flush+Reload:

1. Create an array of fixed size in process memory.
2. Flush the cache.
   – *clflush* instruction
3. Leak information from the kernel by accessing exactly ONE address of the array. This access is revoked, but only after it is cached.
4. Time access to each element of the array. The fastest one is probably the address we modified.
   – *rdtscp* instruction

Flush+Reload

Protected Data Structure



Flush Caches →

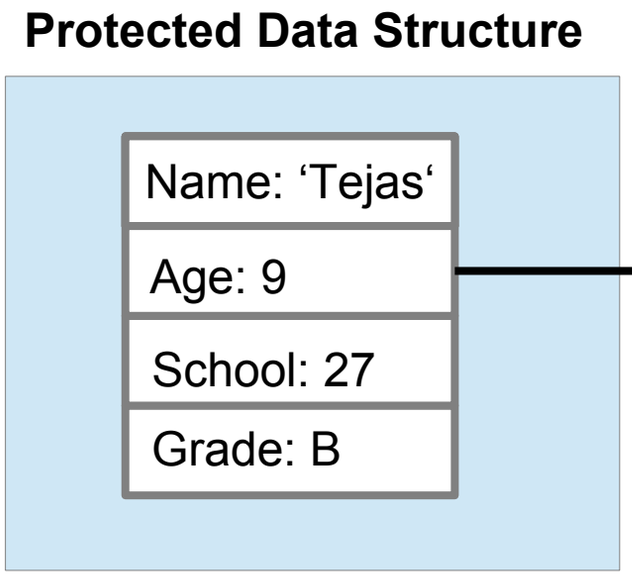
Cache

	Addr	Value	Valid
0			
1			
2			
3			

Unprotected Array

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Flush+Reload



Access A[protected value]

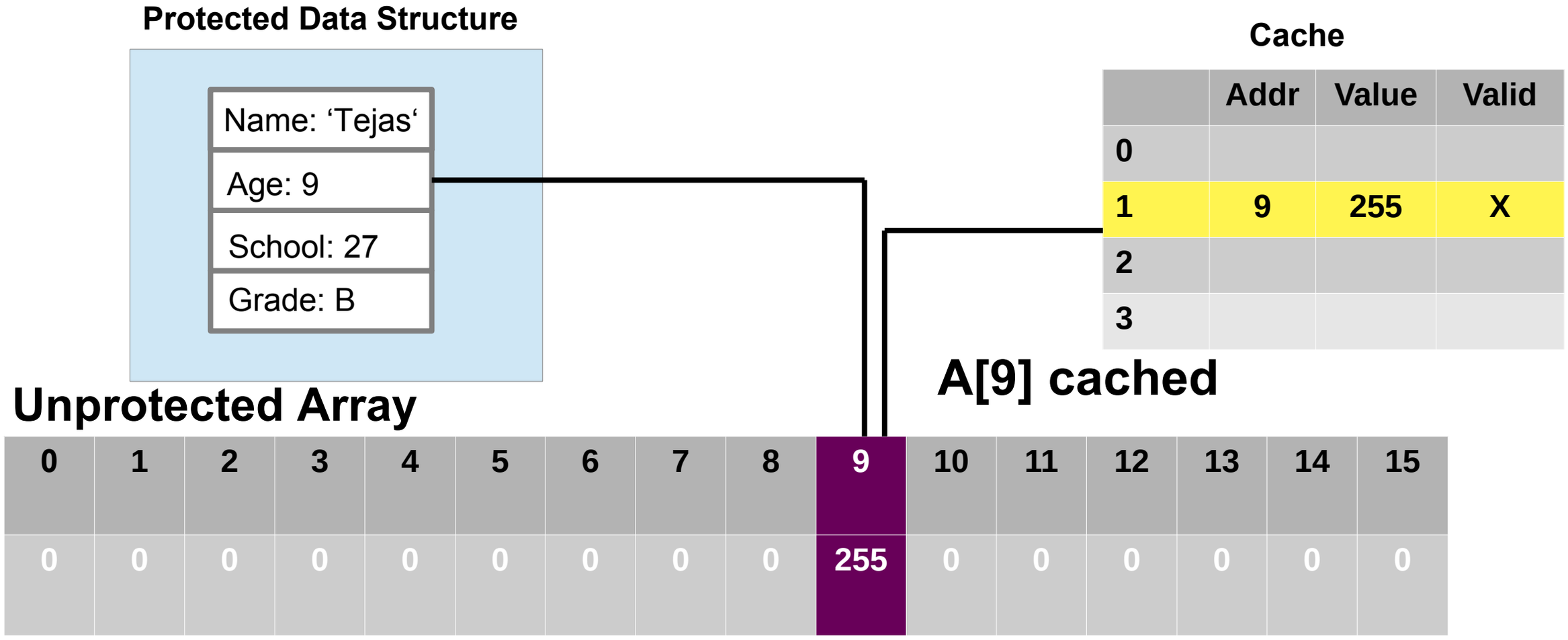
Cache

	Addr	Value	Valid
0			
1			
2			
3			

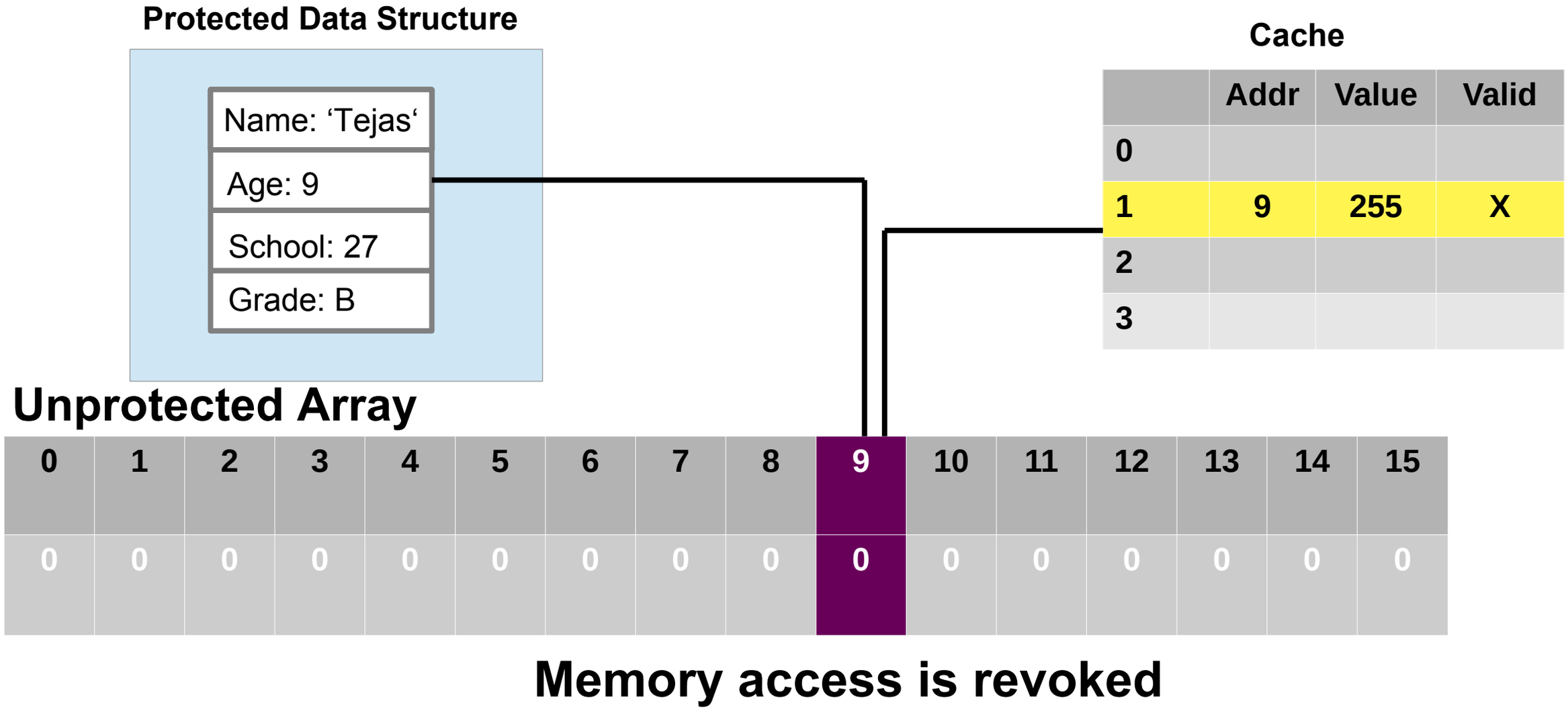
Unprotected Array

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	255	0	0	0	0	0	0

Flush+Reload



Flush+Reload



Flush+Reload

Protected Data Structure

Name: ‘Tejas‘
Age: 9
School: 27
Grade: B

Cache

	Addr	Value	Valid
0			
1	9	0	X
2			
3			

But stays in cache

Unprotected Array

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Flush+Reload

Timing access to each array element reveals that position 9 is cached.

Cache

	Addr	Value	Valid
0			
1	9	0	X
2			
3			

Unprotected Array

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
40ms	40ms	41ms	40ms	41ms	40ms	40ms	41ms	40ms	**10ms**	40ms	40ms	40ms	38ms	40ms	42ms

Meltdown

- **Meltdown:**
  - Access an out-of-bounds memory location
  - This will generate a segmentation fault
  - However, in the meantime, further computation can be used to trigger an **in-bounds** memory access.
  - This access will also be ultimately be discarded, but not before the access is cached
  - Use Flush+Reload to determine the out-of-bounds value

Meltdown

- **For this to work:**
  - Must handle the segmentation fault so it doesn‘t crash the program:
    - Catch it with a signal handler
    - Run it in a child process
  - Must access the information before the CPU zeros it out
    - If address is zero, try again until the segfault terminates the process
  - Must run on certain vulnerable Intel CPUs

Spectre

- Meltdown uses only out-of-order execution to read protected memory from userspace
- Other processors (ARM, AMD) are vulnerable to an attack which combines this with branch misprediction
- Spectre adds branch prediction to make this work on other CPUs:

```
if (idx < bounds) {
    val = target[idx];
    A[val] = 255
}
```

- Need to trick the CPU into thinking it's going to be executed for idx > bounds

Spectre

- **BTB – Branch Target Buffer**

  Uses current PC to predict next PC so that we can begin fetching instructions from a branch

- **BHB – Branch History Buffer**
  - Stores information about last 29 branches (determined experimentally by Project Zero[1])
  - Used to predict if we will take a branch or not

1. *Jann Horn, Reading Privileged Memory with a Side-channel, Project Zero Blog, Jan. 3, 2018*

Spectre

Code

```
for (i=0; i < 30; ++i) {

        if (i != 30) {
          p = &A[i];
        } else {
           p = &A[target];
        }

        if (p < end) {
           val = *p;
        }
}
```

0x8601060

Branch History Buffer

0x08601060	0x08601060	0x08601060
0x08601060	0x08601060	0x08601060
0x08601060	0x08601060	0x08601060
0x08601060	0x08601060	0x08601060
0x08601060	0x08601060	0x08601060
0x08601060	0x08601060	0x08601060
0x08601060	0x08601060	0x08601060
0x08601060	0x08601060	0x08601060
0x08601060	0x08601060	0x08601060
0x08601060	0x08601060	0x08601060
0x08601060	0x08601060	**0x08601060**

Spectre

- Does not segfault, but can only access in-process values
- Can be exploited in-kernel
  - Existing vulnerability in kernel source
  - eBPF JIT compiler (Variant 1)
  - Browser Javascript JIT compilers
- Can cross VM boundaries (Variant 2)

1. *Jann Horn, Reading Privileged Memory with a Side-channel, Project Zero Blog, Jan. 3, 2018*

Solutions and Workarounds

- KPTI (Kernel Page Table Isolation)/KAISER
  - Separates the user page table and kernel page table
  - Access to the KPT only allowed while in Kernel Mode
  - Comes with about a 5% performance penalty (more for some workloads)
  - Particularly noticeable for PostgreSQL and Redis systems (up to 30% penalty)
- Compiler patches
- Browser patches
- New processor design

Questions?

- Dr. Robert Marmorstein

  Longwood University

  marmorsteinrm@longwood.edu
  (434)395-2185

http://marmorstein.org/~robert/SwiftSpecTalk.pdf