Exploiting System Vulnerabilities

Roja Eswaran, PhD

What's a vulnerability?

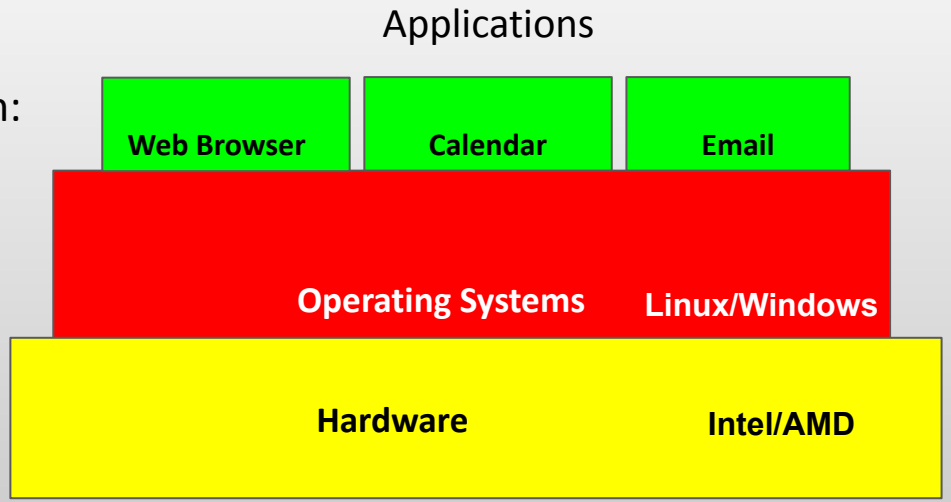
"A vulnerability is a weakness or flaw in a Hardware, Operating system, or application that can be exploited by attackers to gain unauthorized access, disrupt services, or steal sensitive information."

-ChatGPT

Computer System Components

Three main components of any computer system:

- Applications
- Operating System
- Hardware



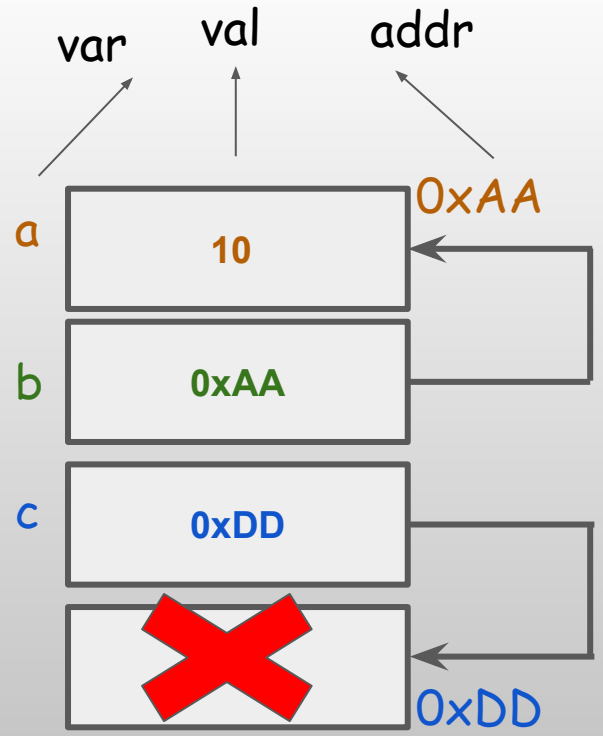
Vulnerability in one of the above components can potentially compromise the entire system

Application Vulnerability

Pointers

Pointers: A variable that stores a memory address

```
void function () {
        int a;
        a = 10;
        int *b;
        b = &a;
      int *c = malloc (sizeof(int));
      *c = 20;
       free (c);
}
```



malloc allocates a memory region of requested size

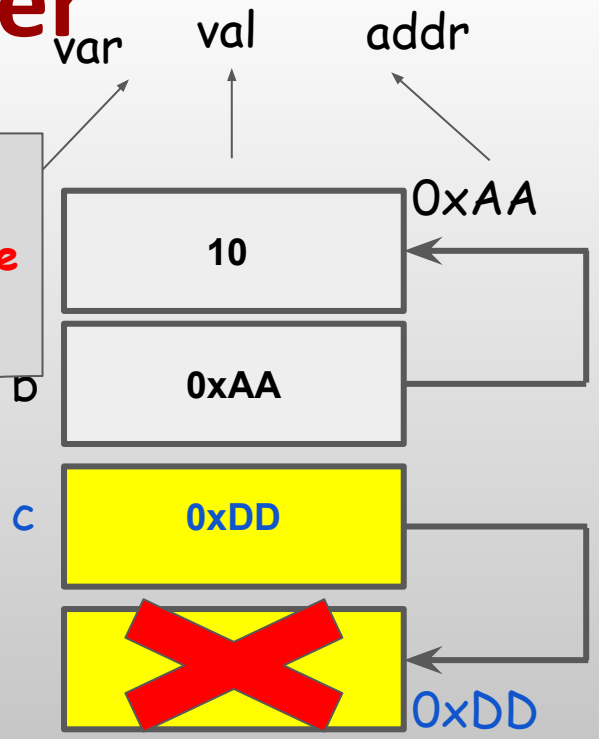
free deallocates a previously allocated region (**Problematic**)

Dangling Pointer

```
void function () {
        int a;
        a = 10;
        int *b;
        b = &a;
        int *c = malloc (sizeof(int));
        *c = 20;
        free (c);
        printf("%d", *c);
}
```

printf("%d", *c);

Ideally there should be segmentation fault!



c is a dangling pointer - it points to a deallocated/freed memory region

Major security threat!!!

Why Dangling Pointer is a threat?

```
1.  int *ptr = malloc(sizeof(int));
2.  *ptr = 10;
3.  printf("Val:%d addr:%p\n", *ptr, ptr);

4.  free (ptr); //ptr is a dangling pointer now
5.  printf("Val:%d addr:%p\n", *ptr, ptr);

6.  *ptr = 15;
7.  printf("Val:%d addr:%p\n", *ptr, ptr);
```

Ptr:
Dangling pointer is a major bug in C programming and very dangerous as it can still access the deallocated/freed memory region!

Output:

Val:10 addr:0xDD

Val:-634323 addr:0xDD

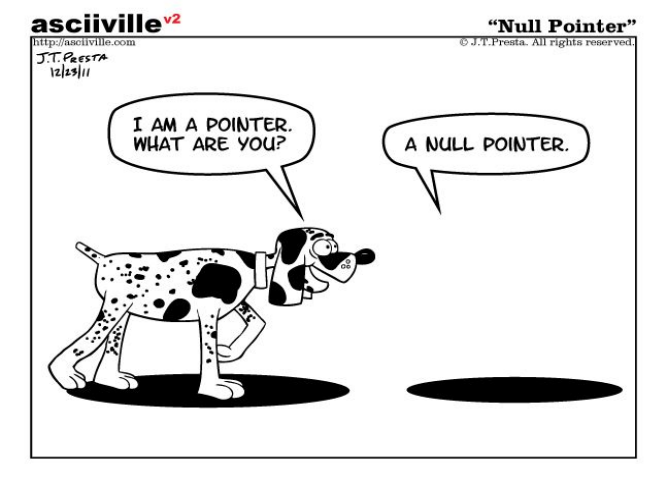
Val:15 addr:0xDD

NULL Pointer to Rescue

```
1.  int *ptr = malloc(sizeof(int));
2.  *ptr = 10;
3.  printf("Val:%d addr:%p\n", *ptr, ptr);

4.  free (ptr); //ptr is a dangling pointer now
5.  ptr = NULL;
6.  printf("Val:%d addr:%p\n", *ptr, ptr);

7.  *ptr = 15;
8.  printf("Val:%d addr:%p\n", *ptr, ptr);
```



Output

Val:10 addr:0xDD

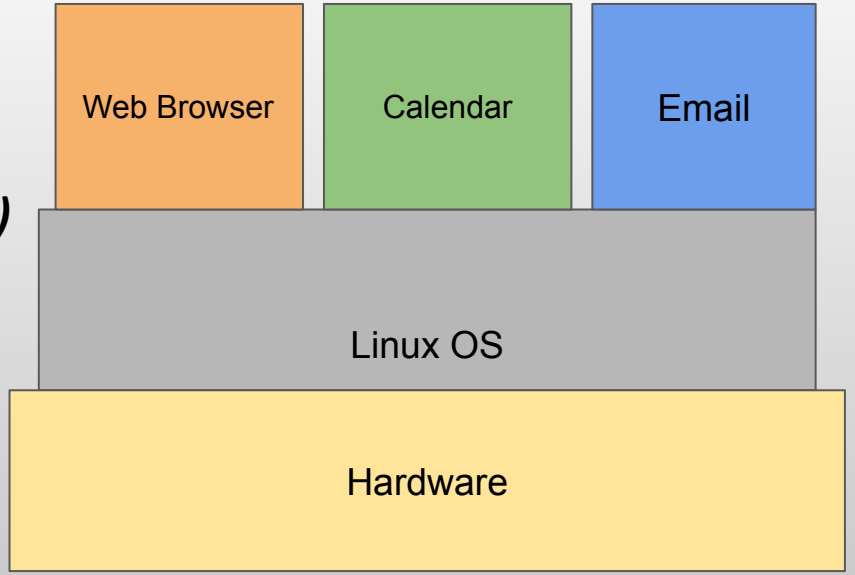
Val:nil addr:nil

Seg. fault

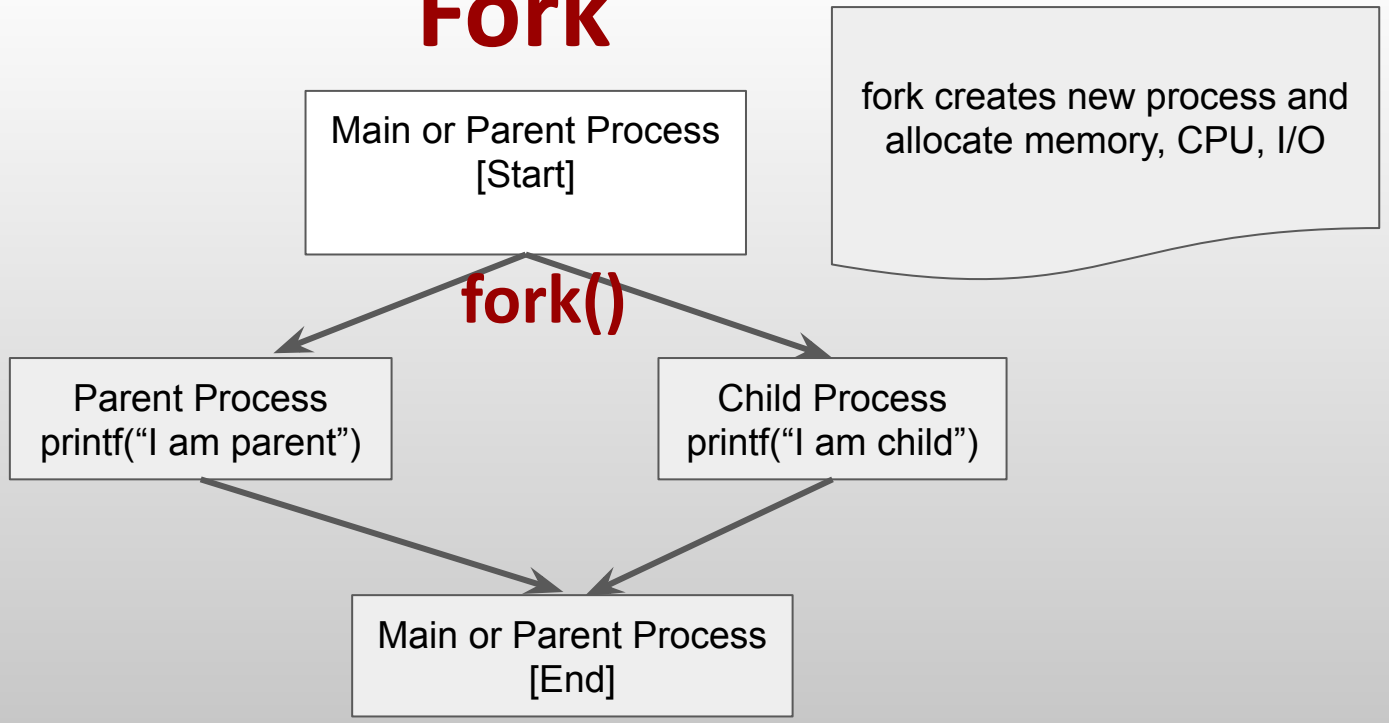
Operating System Vulnerability

Process

- Program under execution: There is a subtle difference between Process and Program
- A new process can only be created using ***Fork()***
- Whenever a process is created, OS allocates dedicated resources such as memory, CPU, IO
- The more the number of processes, the more system resources are utilized.

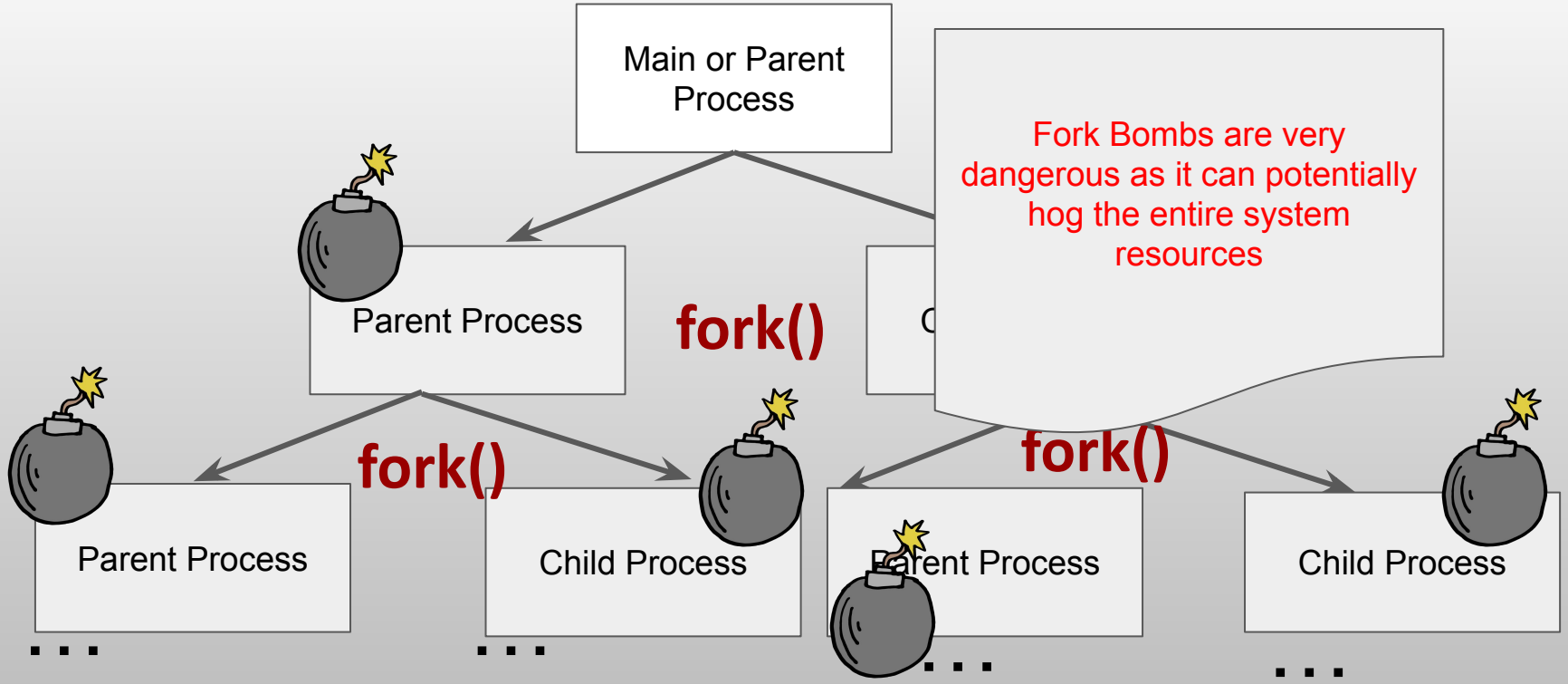


Fork



- Parent and Child can either do a task together or independently
- Parent should properly kill the child process before termination

Fork Bomb


Main or Parent Process
Fork Bombs are very dangerous as it can potentially hog the entire system resources
Parent Process
fork()
fork()
fork()
Parent Process
Child Process
Parent Process
Child Process

Preventing Fork Bomb

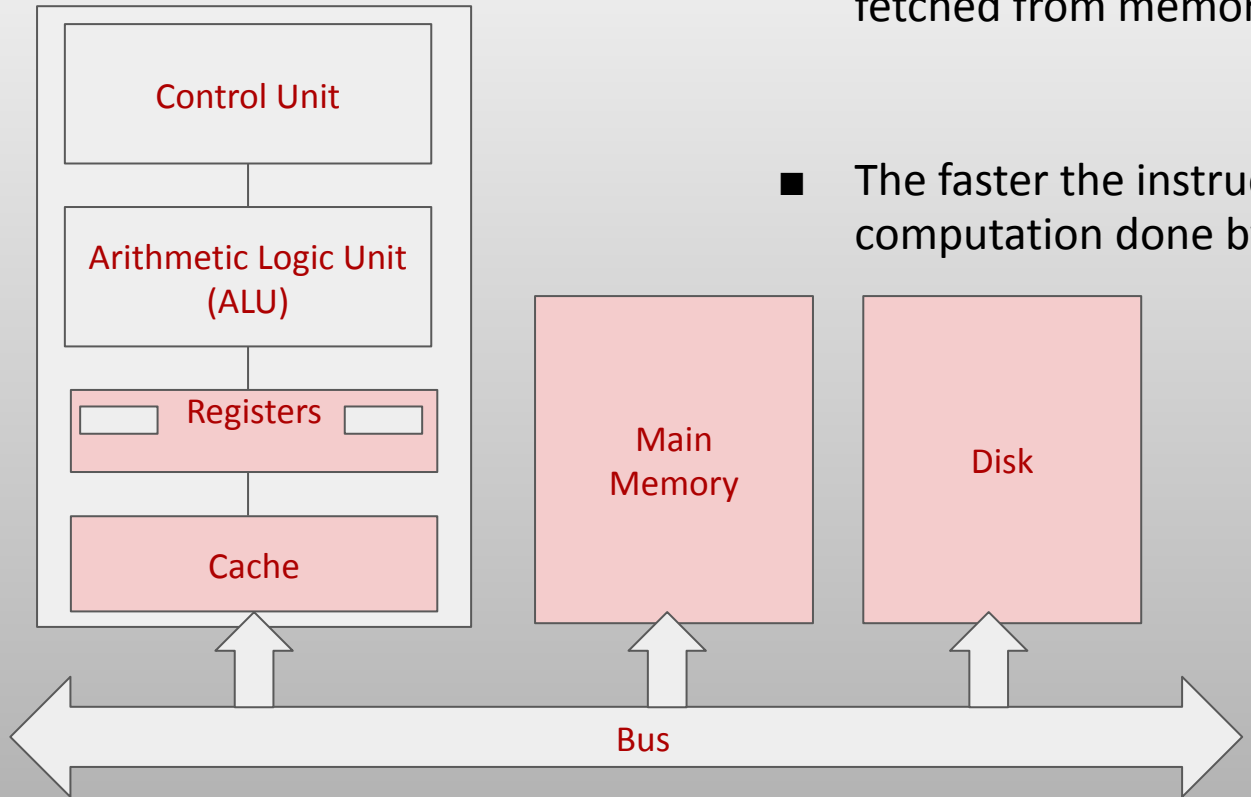
- Fork bombs are dangerous, as it hogs the system resources affecting the performance of critical apps

- The system administrator should set the ***ulimit*** to avoid the fork bomb

Hardware Vulnerability

Processor

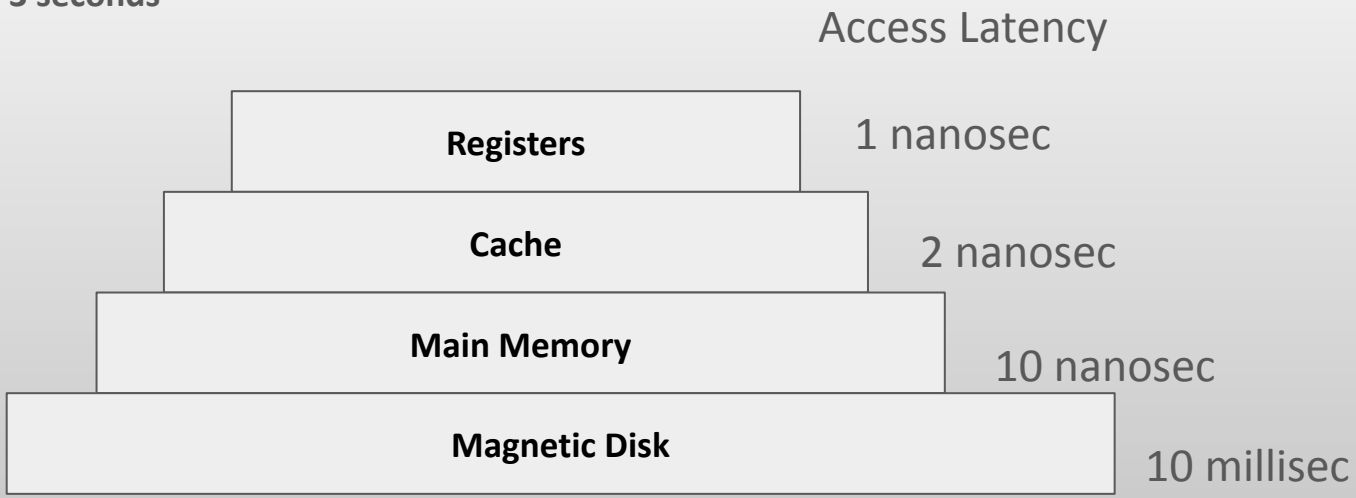
Central Processing Unit (CPU)



- CPU - “Brain” of a computer execute instructions fetched from memory
- The faster the instruction is fetched, the faster is the computation done by CPU

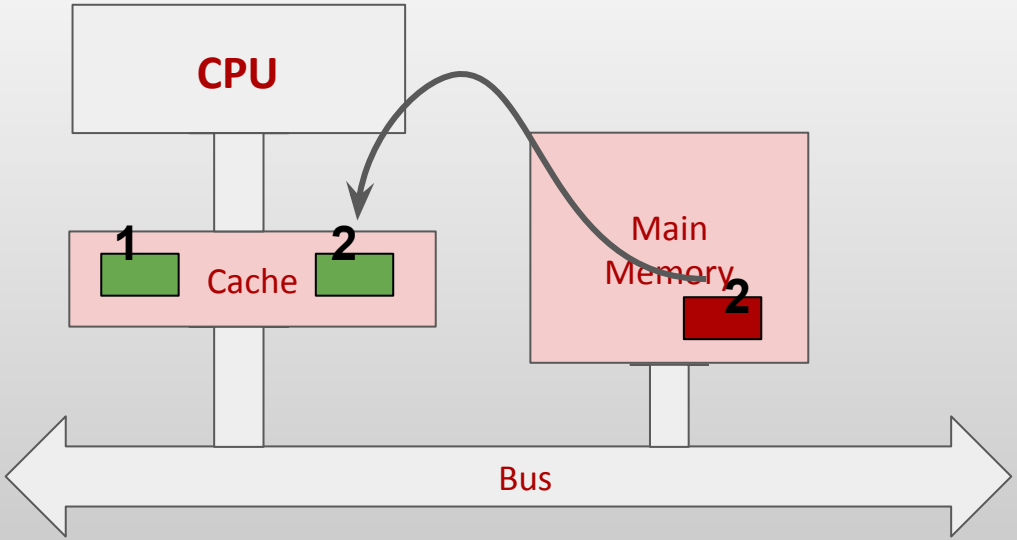
Memory

1 nanosec = 10^{-9} seconds
1 millisec = 10^{-3} seconds



- Next to registers, cache has faster access latency
- If the CPU requested data is not in cache, it's fetched from main memory
- From top to bottom, access latency increases and so do the size

Cache



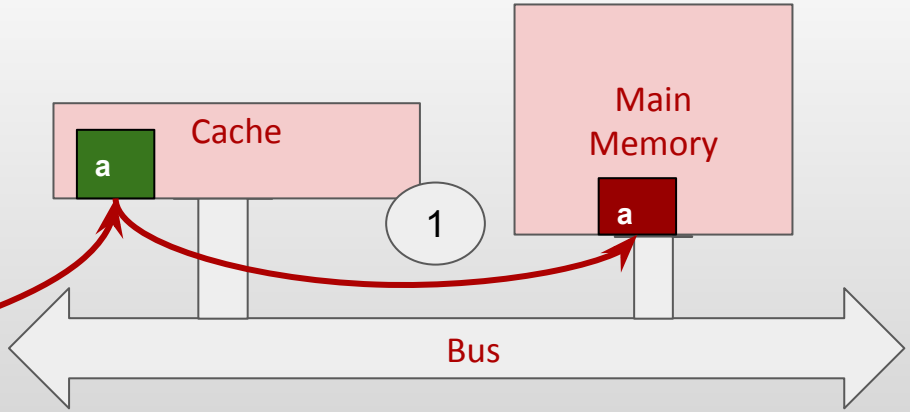
1. **Cache hit:** If the CPU requested data is present in cache (***Faster***)
2. **Cache miss:** If the data is not present in cache, they need to be fetched and loaded into cache from memory (***Slower***)

By using the timing difference between cache hit/miss, attacker can gauge the access pattern of victim

Flush and Reload Attack (1/4)

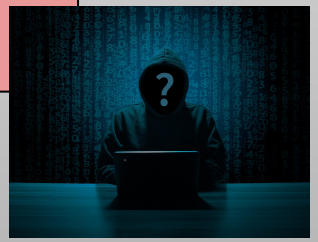
```
if secret do
        maccess(&a);

else
        maccess(&b);
endif
```



1. Attacker flush the shared variable ***a***

```
flush(&a);
```

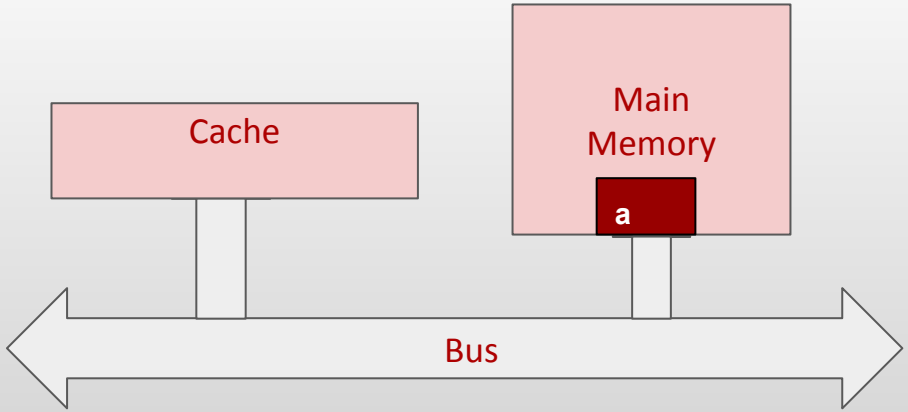


Flush and Reload Attack (2/4)

2

```
if secret do
        maccess(&a);

else
        maccess(&b);
endif
```

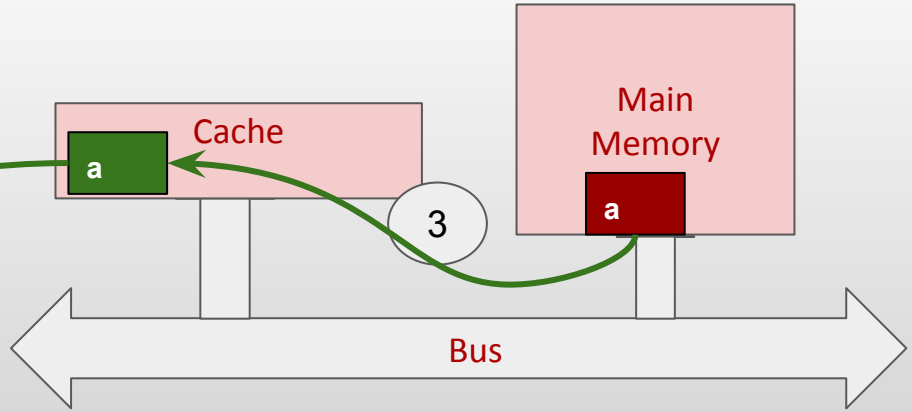


2. Waits for the victim to access the same shared memory region flushed to memory

Flush and Reload Attack (3/4)

```
if secret do
        maccess(&a);

else
        maccess(&b);
endif
```

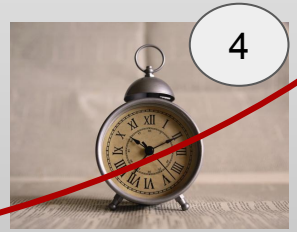


3. Once the victim access the shared data again, due to ***cache miss***, the data is fetched from main memory and loaded into cache.

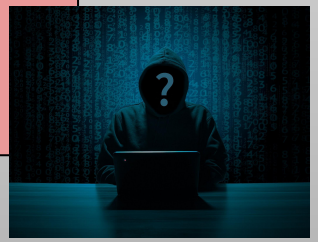
Flush and Reload Attack (4/4)

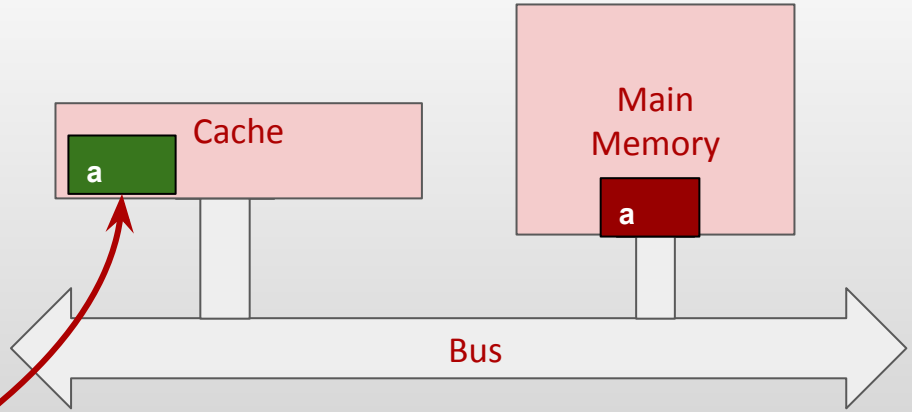
```
if secret do
        maccess(&a);

else
        maccess(&b);
endif
```




```
start_timer
        maccess(&a);
end_timer
```





4. Attacker starts the timer and check the access time of ***a*.**

If the victim has accessed the shared region ***a***, due to ***cache hit***, the access time is fast!

Summary

- We have identified how ***a vulnerability*** in any one of the computer subsystem stack could potentially compromise the entire system.

- We discussed different types of vulnerabilities such as ***dangling pointers, fork bomb , flush and reload attack*** and potential solutions to address them.

Thank you!

Questions?