Protection

Disclaimer: some slides are adopted from book authors' slides with permission

Today

- Protection
- Security

Examples of OS Protection

- Memory protection
  - Between user processes
  - Between user and kernel
- File protection
  - Prevent unauthorized accesses to files

p 626

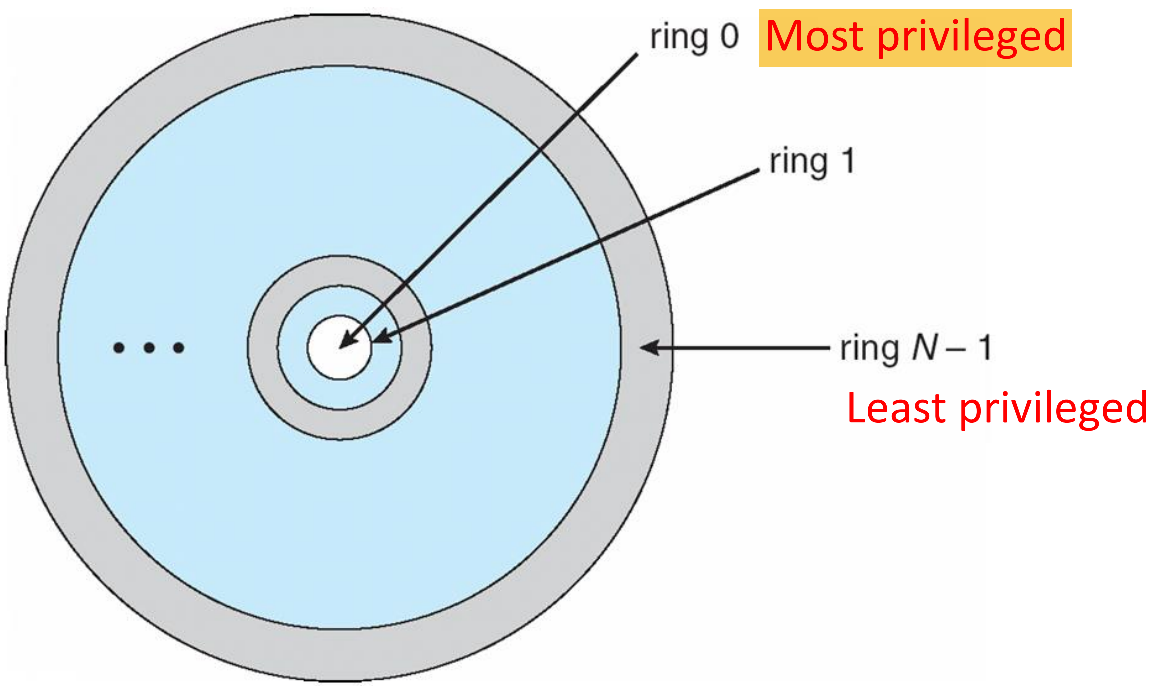
Principles of Protection

- Principle of least privilege
  - Programs and users should be given just enough privileges to perform their tasks
  - Limit the damage if the entity has a bug or abused

Protection Domains

- Let D_i and D_j be any two domain rings
- If $j < I \Rightarrow D_i \subseteq D_j$
- Kernel mode vs. user mode

p631



Access Control Matrix

- **Domains** in rows
  - Domain: a user or a group of users
- **Resources** in columns
  - File, device, …

E.g., User D1 can read F1 or F3

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Method 1: Access Control List

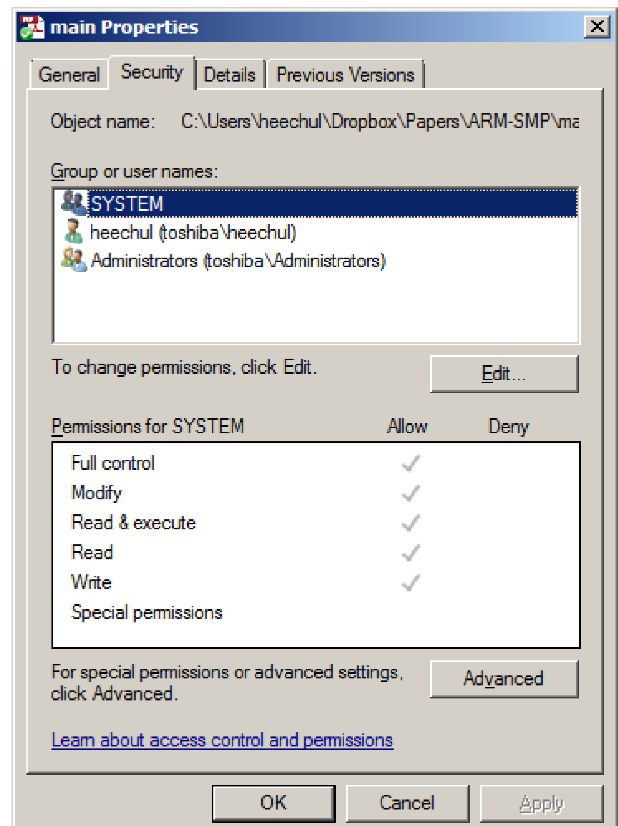
- Each **object** stores users and their permissions

-rw-rw-r-- **heechul heechul** 38077 Apr 23 15:16 **main.tex**

owner

group

world



Method 2: Capability List

- Each **domain** tracks which objects can access
  - Page table: each process (domain) tracks all pages (objects) it can access

Summary

- Protection
  - Prevent unintended/unauthorized accesses
- Protection domains
  - Class hierarchy: *root* can to everything a normal *user* can do + alpha
- Access control matrix
  - Domains (Users) ← → Resources (Objects)
  - Resource oriented: Access control list
  - Domain oriented: Capability list

Security

Today

- Security basics
- Stack overflow
- Some recent security bugs

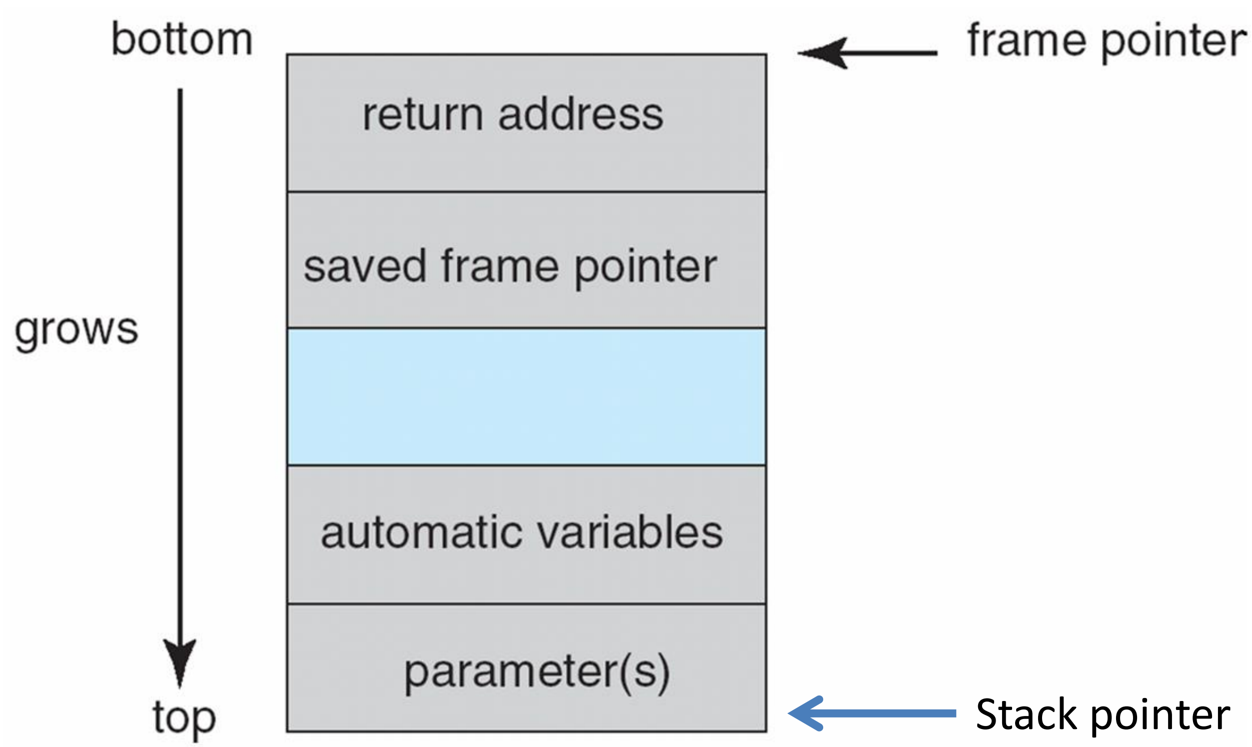
Security

- System **secure** if resources used and accessed as intended under all circumstances
  - Unachievable
- **Intruders** (**crackers**) attempt to breach security
- **Threat** is potential security violation
- **Attack** is attempt to breach security

Threats: Software

- **Stack and Buffer Overflow**
  - Exploits a bug in a program (overflow either the stack or memory buffers)
  - Failure to check bounds on inputs, arguments
  - Write past arguments on the stack into the return address on stack
  - When routine returns from call, returns to hacked address
    - Pointed to code loaded onto stack that executes malicious code
  - Unauthorized user or privilege escalation

Stack Frame Layout



Code with Buffer Overflow

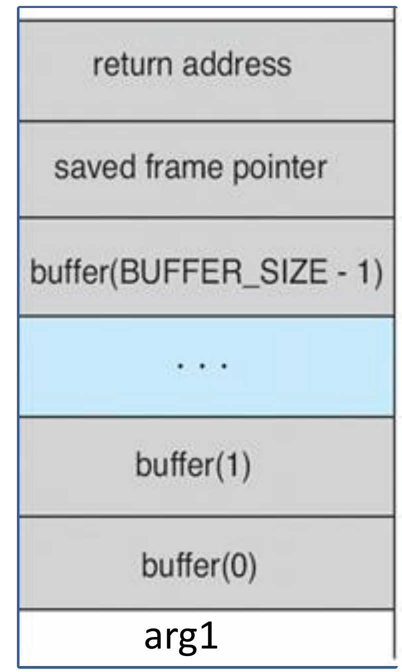
BUFFER_SIZE only has 256; if more than, will cause segmentation fault

```
#define BUFFER_SIZE 256
int process_args(char *arg1)
{
    char buffer[BUFFER SIZE];
    strcpy(buffer,arg1);
    ...
}

int main(int argc, char *argv[])
{
    process_args(argv[1]);
    ...
}
```

- What is wrong in this code?

Code with Buffer Overflow

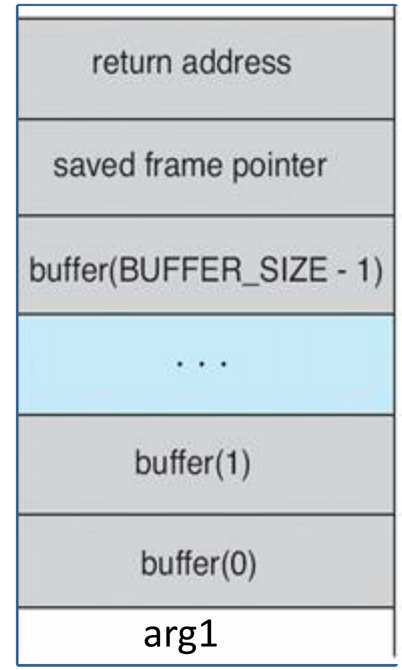
```
#define BUFFER_SIZE 256
int process_args(char *arg1)
{
    char buffer[BUFFER SIZE];
    strcpy(buffer,arg1);
    ...
}

int main(int argc, char *argv[])
{
    process_args(argv[1]);
    ...
}
```



- Stack layout after calling *process_arg()*

Code with Buffer Overflow

```
#define BUFFER_SIZE 256
int process_args(char *arg1)
{
    char buffer[BUFFER SIZE];
    strcpy(buffer,arg1);
    ...
}

int main(int argc, char *argv[])
{
    process_args(argv[1]);
    ...
}
```



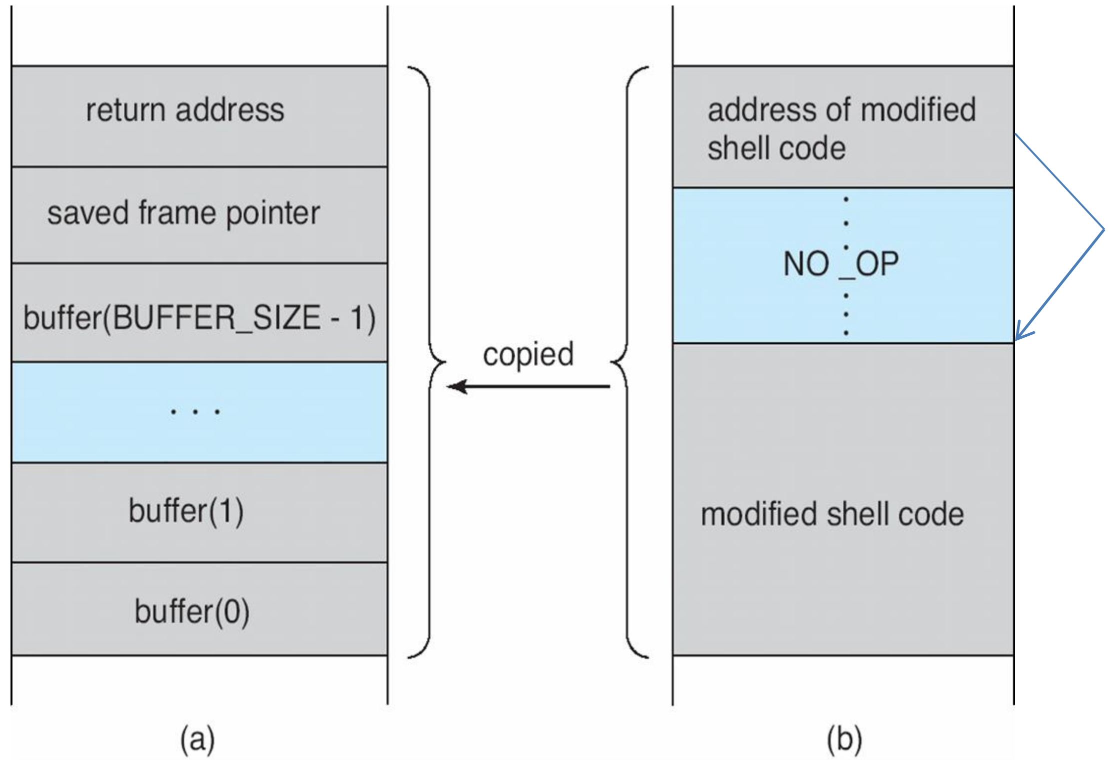
- Do you remember *strcpy()* in C?

Let’s Get the Shell

- Steps
  - Compile the code you want to illegitimately execute
  - ‘Carefully’ modify the binary
  - Pass the modified binary as string to the *process_arg()*

```
#include <stdio.h>
int main(int argc, char *argv[])
{
        execvp(``/bin/sh'',``/bin/sh'', NULL);
        return 0;
}
```

The Attack: Buffer Overflow



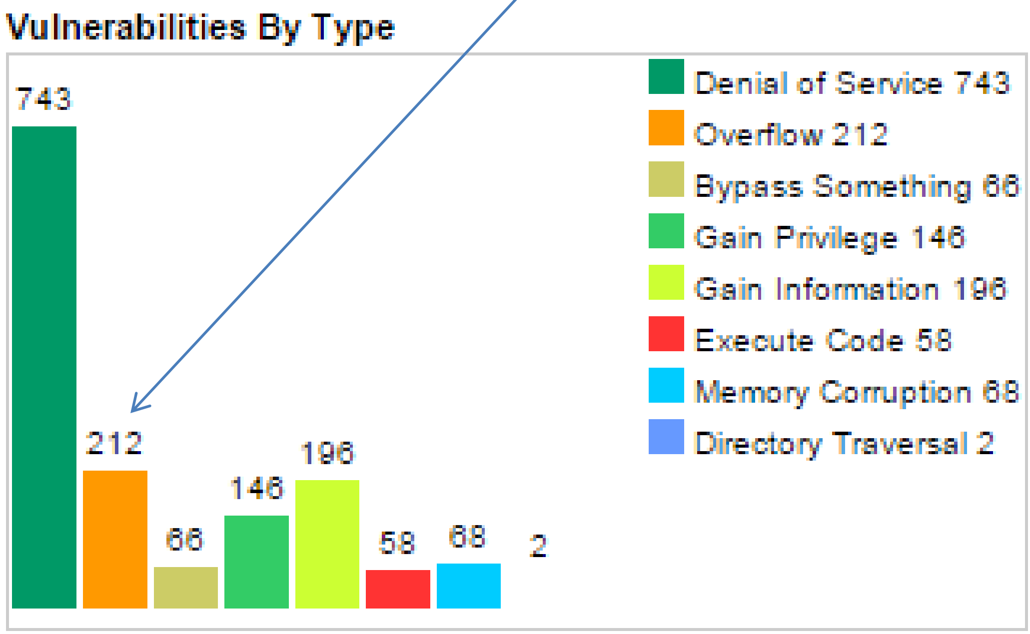
Before

After executing
*strcpy(buffer, **arg1**)*

the crafted string containing the illegitimate code

Linux Kernel Buffer Overflow Bugs

212 reported buffer overflow bugs in Linux



Source: http://www.cvedetails.com/vulnerability-list/vendor_id-33/product_id-47/cvssscoremin-9/cvssscoremax-/Linux-Linux-Kernel.html

Linux Kernel Buffer Overflow Bugs

6	CVE-2010-2521	119	DoS Exec Code Overflow	2010-09-07	2012-03-19	10.0	None	Remote	Low	Not required	Complete	Complete	Complete

Multiple buffer overflows in fs/nfsd/nfs4xdr.c in the XDR implementation in the NFS server in the Linux kernel before 2.6.34-rc6 allow remote attackers to cause a denial of service (panic) or possibly execute arbitrary code via a crafted NFSv4 compound WRITE request, related to the read_buf and nfsd4_decode_compound functions.

9	CVE-2009-0065	119	Overflow	2009-01-07	2012-03-19	10.0	Admin	Remote	Low	Not required	Complete	Complete	Complete

Buffer overflow in net/sctp/sm_statefuns.c in the Stream Control Transmission Protocol (sctp) implementation in the Linux kernel before 2.6.28-git8 allows remote attackers to have an unknown impact via an FWD-TSN (aka FORWARD-TSN) chunk with a large stream ID.

10	CVE-2008-5134	119	Overflow	2008-11-18	2012-03-19	10.0	None	Remote	Low	Not required	Complete	Complete	Complete

Buffer overflow in the lbs_process_bss function in drivers/net/wireless/libertas/scan.c in the libertas subsystem in the Linux kernel before 2.6.27.5 allows remote attackers to have an unknown impact via an "invalid beacon/probe response."

11	CVE-2008-3915	119	Overflow	2008-09-10	2012-03-19	9.3	None	Remote	Medium	Not required	Complete	Complete	Complete

Buffer overflow in nfsd in the Linux kernel before 2.6.26.4, when NFSv4 is enabled, allows remote attackers to have an unknown impact via vectors related to decoding an NFSv4 acl.

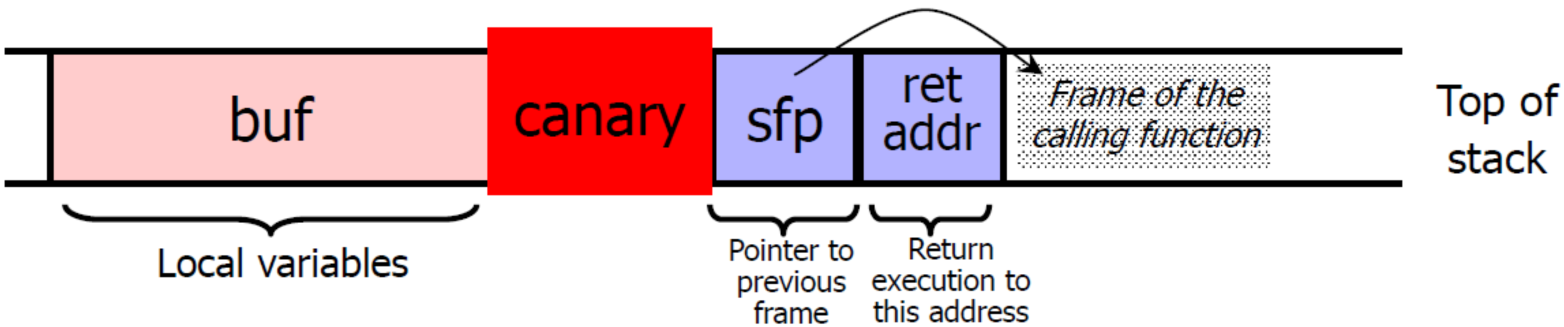
12	CVE-2008-3496	119	Overflow	2008-08-06	2012-03-19	10.0	None	Remote	Low	Not required	Complete	Complete	Complete

Buffer overflow in format descriptor parsing in the uvc_parse_format function in drivers/media/video/uvc/uvc_driver.c in uvcvideo in the video4linux (V4L) implementation in the Linux kernel before 2.6.26.1 has unknown impact and attack vectors.

13	CVE-2008-1673	119	DoS Exec Code Overflow	2008-06-09	2012-11-26	10.0	None	Remote	Low	Not required	Complete	Complete	Complete

Run-Time Checking: StackGuard

- Embed "canaries" (stack cookies) in stack frames and verify their integrity prior to function return
  - Any overflow of local variables will damage the canary



- Choose random canary string on program start
  - Attacker can't guess what the value of canary will be
- Terminator canary: "\0", newline, linefeed, EOF
  - String functions like strcpy won't copy beyond "\0"

Slide from Dr. Vitaly Shmatikov (Cornell)

PaX

- Linux kernel patch
- Goal: prevent execution of arbitrary code in an existing process's memory space
- Enable executable/non-executable memory pages
- Any section not marked as executable in ELF binary is non-executable by default
  - Stack, heap, anonymous memory regions
- Access control in mmap(), mprotect() prevents unsafe changes to protection state at runtime
- Randomize address space layout

slide 5

Slide from Dr. Vitaly Shmatikov (Cornell)

Problem: Lack of Diversity

- Buffer overflow and return-to-libc exploits need to know the (virtual) address to hijack control
  - Address of attack code in the buffer
  - Address of a standard kernel library routine
- Same address is used on many machines
  - Slammer infected 75,000 MS-SQL servers using same code on every machine
- Idea: introduce artificial diversity
  - Make stack addresses, addresses of library routines, etc. unpredictable and different from machine to machine

Slide from Dr. Vitaly Shmatikov (Cornell)

ASLR

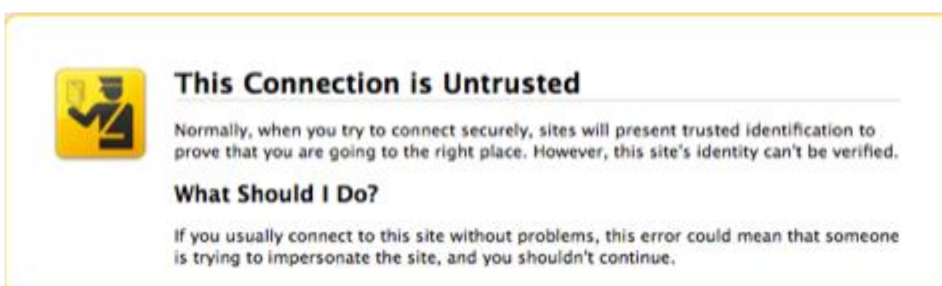
- Address Space Layout Randomization
- Randomly choose base address of stack, heap, code segment
- Randomly pad stack frames and malloc() calls
- Randomize location of Global Offset Table
- Randomization can be done at compile- or link-time, or by rewriting existing binaries
  - Threat: attack repeatedly probes randomized binary

Goto Fail Bug

iOS 7.0.6
Data Security
Available for: iPhone 4 and later, iPod touch (5th generation), iPad 2 and later

Impact: An attacker with a privileged network position may capture or modify data in sessions protected by SSL/TLS

Description: Secure Transport *failed to validate the authenticity of the connection*. This issue was addressed by restoring missing validation steps.



Goto Fail Bug

```
    err = 0
    . . .
    hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
    hashOut.length = SSL_SHA1_DIGEST_LEN;
    if ((err = SSLFreeBuffer(&hashCtx)) != 0)
     goto fail;
    if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
     goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
     goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
     goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
     goto fail;
     goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
     goto fail;

    err = sslRawVerify(...); // This code must be executed
    . . .
fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    Return err;
```

MISTAKE! THIS LINE SHOULD NOT BE HERE

KU THE UNIVERSITY OF KANSAS

Heartbleed Bug

- Synopsis
  - Due to a bug in OpenSSL (popular s/w for encrypted communication), web server's internal memory can be dumped remotely

Heartbleed Bug

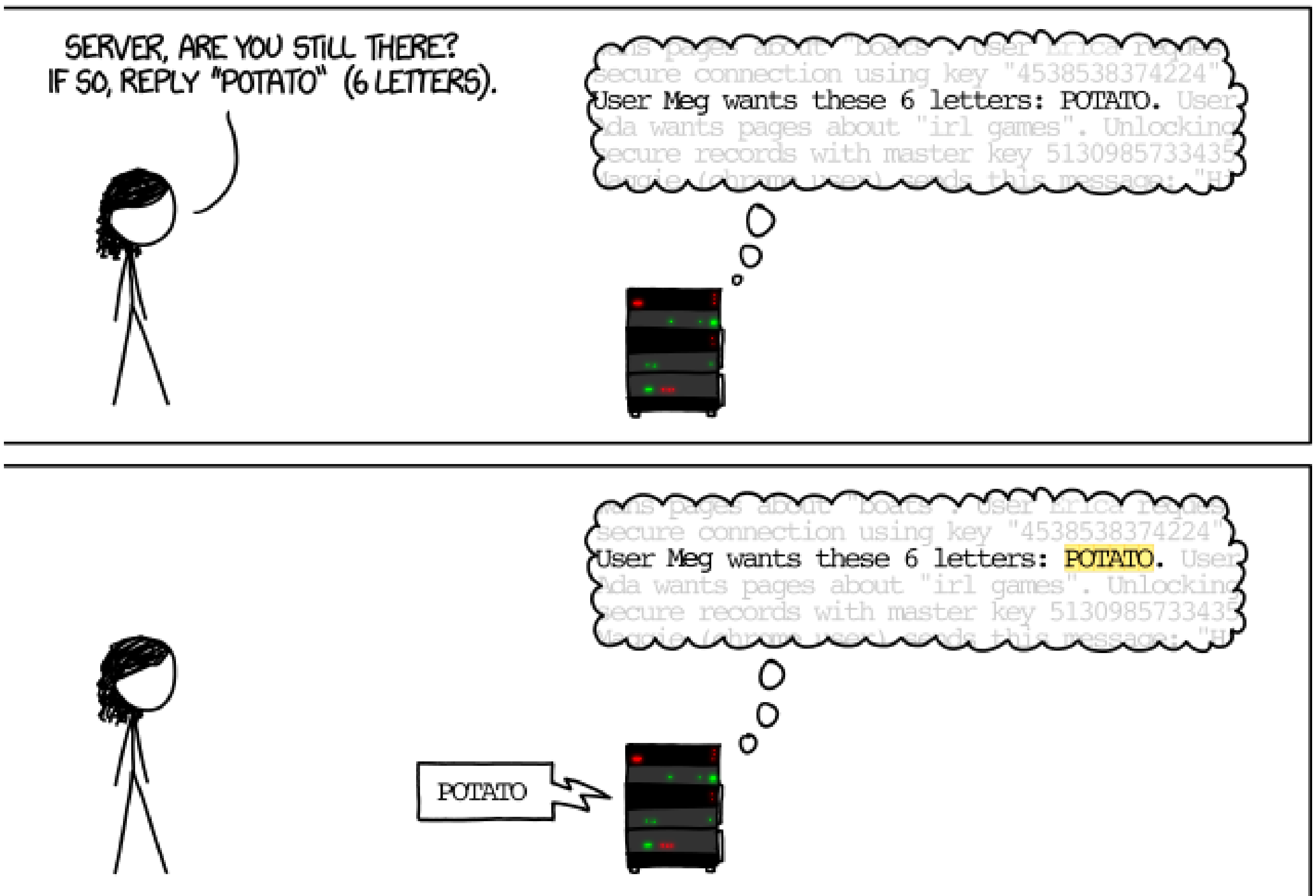


Image source: xkcd.com

Heartbleed Bug

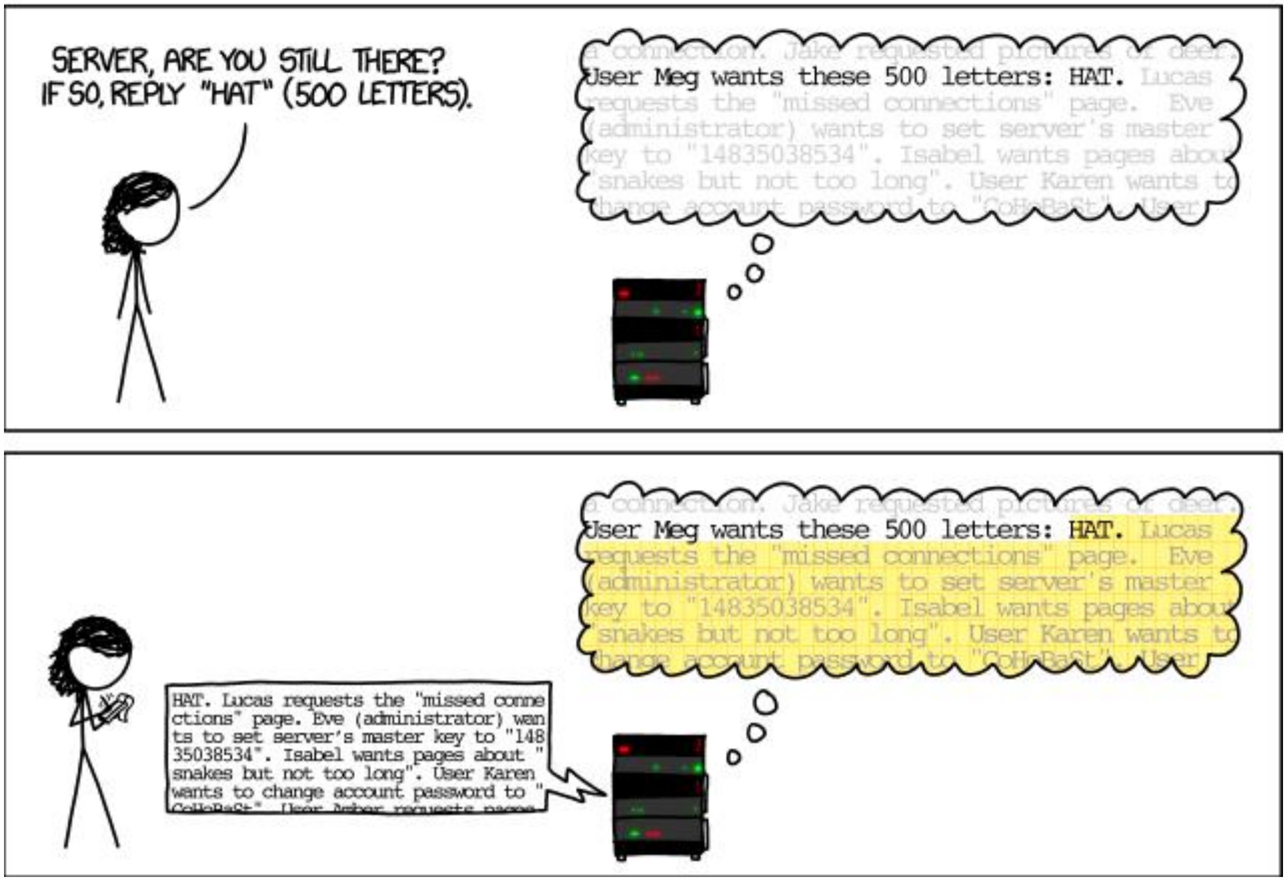


Image source: xkcd.com

Heartbleed Bug

```
struct {
   HeartbeatMessageType type;
   uint16 payload_length;
   opaque payload[HeartbeatMessage.payload_length];
   opaque padding[padding_length];
} HeartbeatMessage
```

Heartbeat
req. message

```
int tls1_process_heartbeat(SSL *s)
{
   ...
   /* Read type and payload length first */
   hbtype = *p++;
   n2s(p, payload); // payload = recv_packet.payload_length
   pl = p;
   ...
   if (hbtype == TLS1_HB_REQUEST) {
      ...
      buffer = OPENSSL_malloc(1 + 2 + payload + padding);
      bp = buffer;
      memcpy(bp, pl, payload);
      r = ssl3_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, 3 + payload + padding);
            ...
```

Heartbeat
Response function

Shellshock Bug

- Synopsis
  - You can *remotely execute arbitrary programs* on a server running a web server by simply sending a specially crafted http request.
  - Example

```
curl -H "User-Agent: () { :; }; /bin/eject" http://example.com/
```

- The problem
  - Fail to check the validity of a function definition before executing it

For detailed explanation: security.stackexchange.com

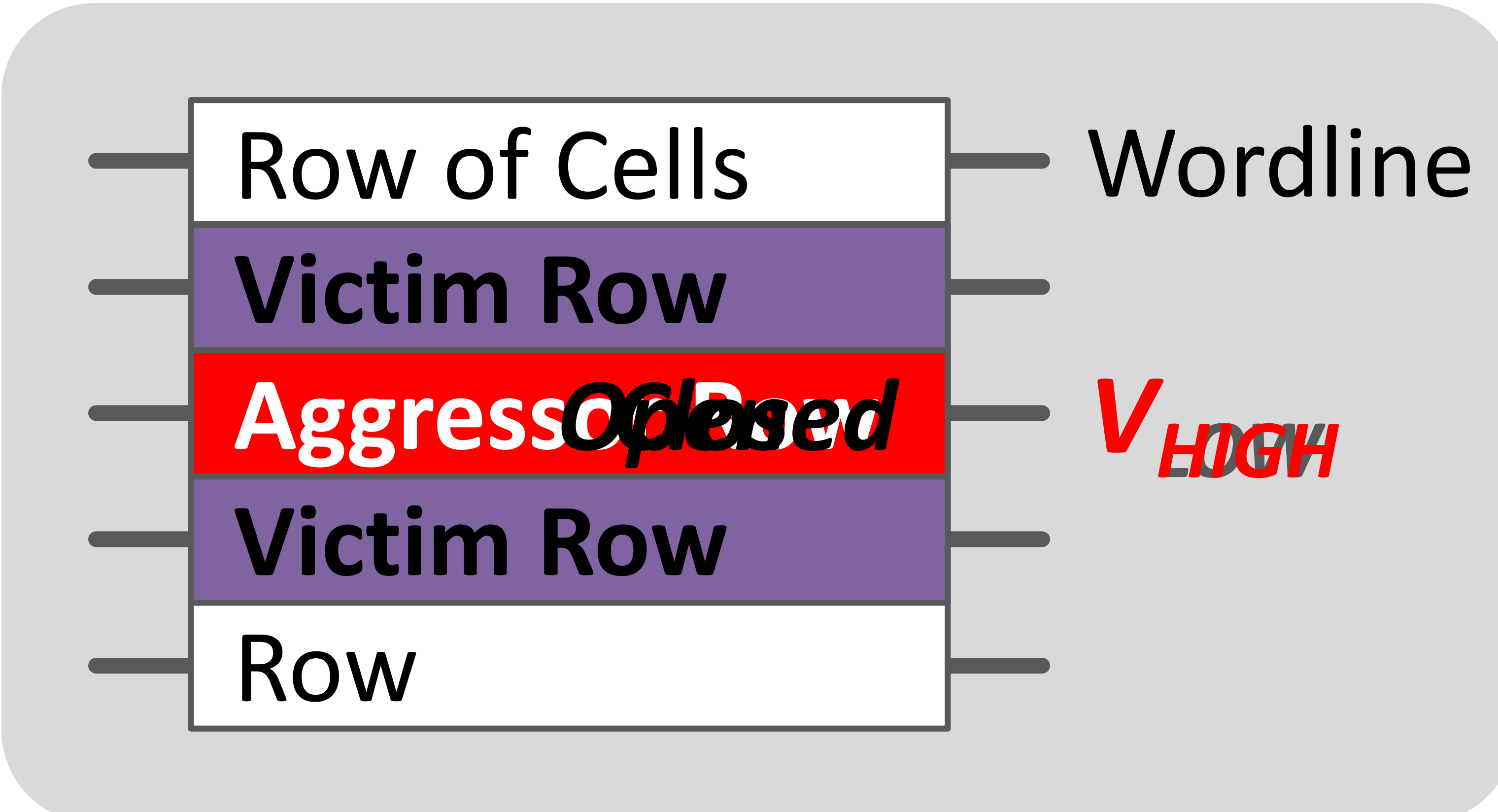
KU THE UNIVERSITY OF KANSAS

Threats: Hardware

- Disturbance errors in DRAM (*)
  - a.k.a. **Row Hammer** Bug
  - Repeated opening/closing a DRAM row can cause **bit flips** in adjacent rows.
  - In more than 80% DRAM modules between 2010 -2013
  - Google demonstrated successful hacking method utilizing the bug (**)
    - manipulate page tables at the user-level

(*) Yoongu Kim et al, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA'14
(**) Google Project Zero. Exploiting the DRAM rowhammer bug to gain kernel privileges, 2015

DRAM Chip



Repeatedly opening and closing a row induces ***disturbance errors*** *in adjacent rows*

Drammer

- Successful exploit to gain root privilege of Android smartphones
  - Exploit row hammer bugs on mobile DRAM
  - Use Android's special memory allocation feature
  - Alter page table entries (privileged) by hammering nearby memory blocks (non-privileged)
  - [Demo]

https://www.vusec.net/projects/drammer/

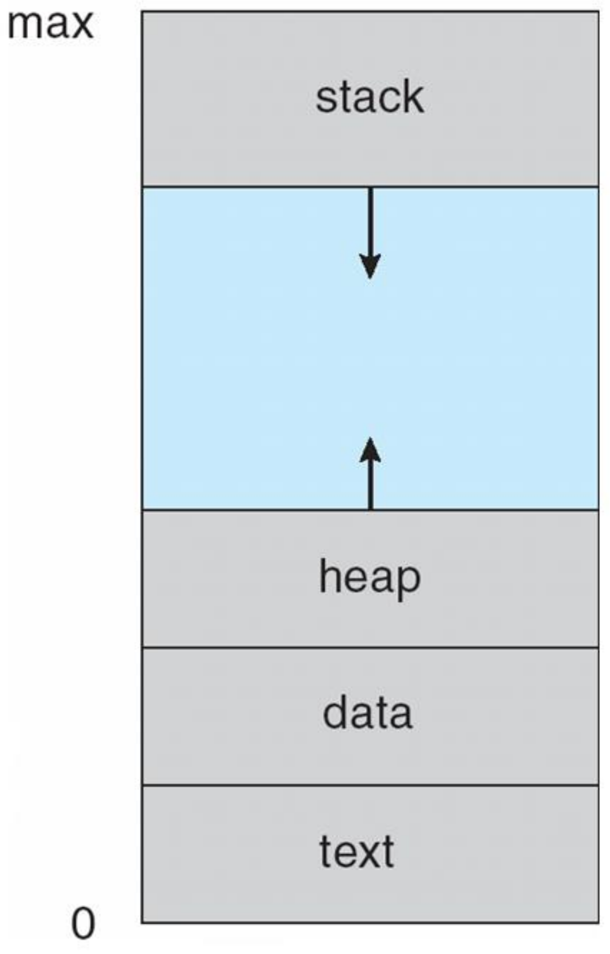
Meltdown

- What is it?
  - An attack that exploits Intel CPU’s flaw that allows any user-level process to read the content of the kernel-only accessible memory---usually the entire dram
- What’s the impact?
  - An attacker can dump the entire memory, including password and other confidential information
- Which CPUs are affected?
  - **Almost all** Intel CPUs that do **Out-of-Order Execution** to improve performance

KU THE UNIVERSITY OF KANSAS

Virtual Memory

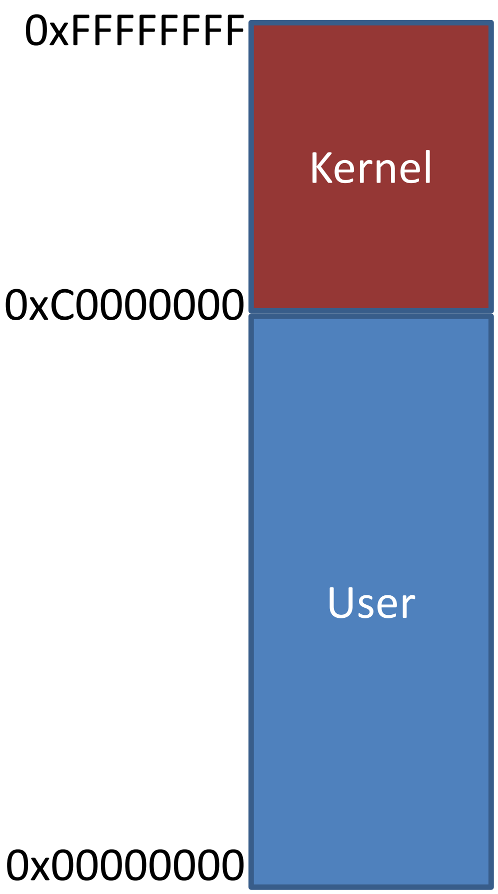
- Abstraction
  - A large (e.g., 4GB) linear address space for each process
- Reality
  - A limited (e.g., 1GB) amount of actual physical memory shared with many other processes
- How?



Properties of Virtual Memory

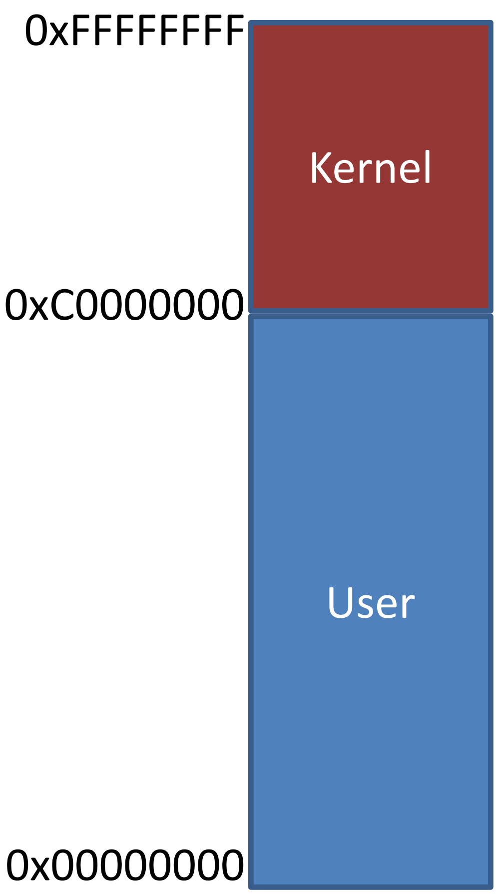
- Memory isolation among different processes
  - E.g., Process A cannot see process B's memory (vice versa.)
- What about memory isolation **between kernel and user**?
  - Q1. how does kernel map its own private memory?
  - Q2. how to prevent user processes from accessing the kernel mapped memory?

Kernel/User Virtual Memory



- Kernel memory
  - Kernel code, data
  - Identical to all address spaces
  - Fixed **1-1 mapping of physical memory**
- User memory
  - Process code, data, heap, stack,...
  - Unique to each address space
  - On-demand mapping (page fault)

Kernel/User Virtual Memory



- Every user-process has mappings to kernel memory
- But the kernel memory is only accessible at the kernel mode
  - when you execute system calls or interrupt handlers.
- Benefits of this design: **Performance**
  - Kernel can move data between user memory and kernel memory easily w/o changing the address space.

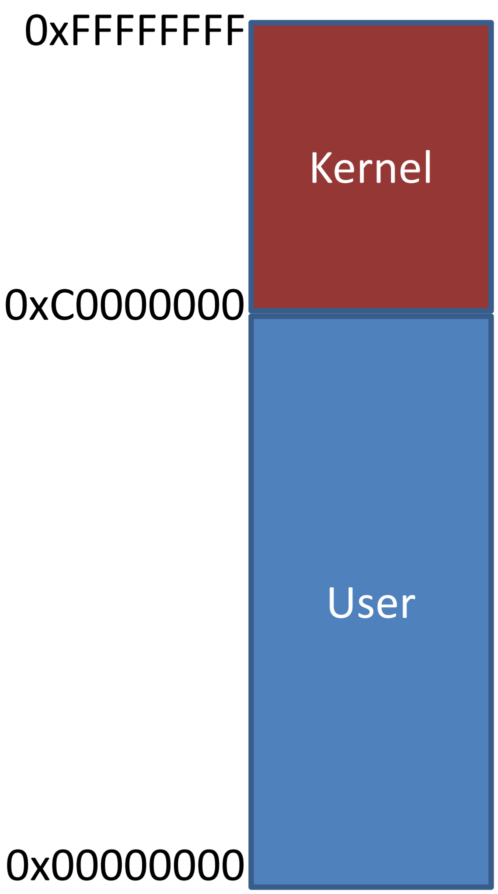
ARM Page Table

Small page	Small page base address, PA[31:12]	nG	S	AP[2]	TEX [2:0]	AP [1:0]	C	B	1	XN

Table B3-4 VMSAv7 MMU access permissions

AP[2]	AP[1:0]	Privileged permissions	User permissions	Description
0	00	No access	No access	All accesses generate Permission faults
0	01	Read/write	No access	Privileged access only
0	10	Read/write	Read-only	Writes in User mode generate Permission faults
0	11	Read/write	Read/write	Full access
1	00	-	-	Reserved
1	01	Read-only	No access	Privileged read-only
1	10	Read-only	Read-only	Privileged and User read-only, deprecated in VMSAv7[a]
1	11	Read-only	Read-only	Privileged and User read-only[b]

Kernel/User Virtual Memory

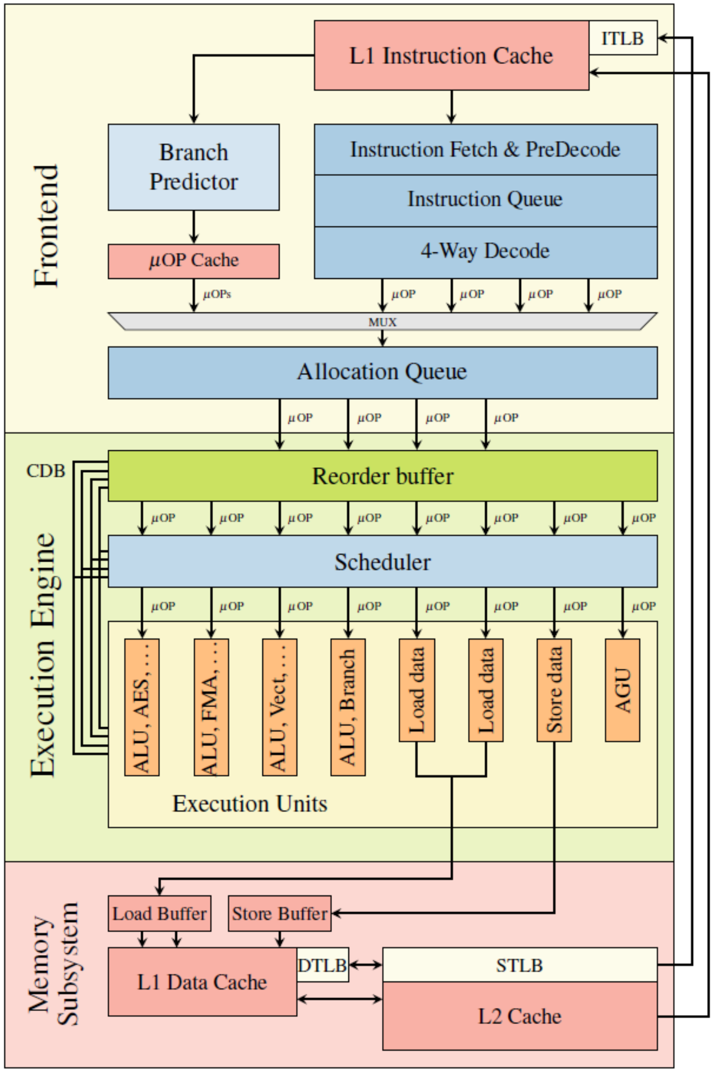


- **Meltdown** tricks the CPU so that the user can access its kernel memory
- How?
  - By exploiting weaknesses in Intel's out-of-order execution engine

Out-of-Order Execution

- Background
  - A cache-miss can take ~100 cycles
  - Idling CPU while waiting data from memory is bad
- Out-of-order execution
  - A technique to minimize data waiting time by executing future instructions
  - Introduced in 1967 (Tomasulo algorithm)
- Most (all) high-performance CPUs use OoOE
  - Intel, AMD, ARM, ….

Out-of-Order Execution



- Instructions are fetched into a queue
- Any instructions whose data (operands) are ready are executed **out-of-order** (subject to data dependency)
- Results are "retired" in-order.

Speculative Execution

```
If (condition)
{
    Do something A1
    Do something A2
    Do something A3
} else {
    Do something B1
    Do something B2
    Do something B3
}
```

- Guess which branch to take.
- Speculatively execute instructions in the likely branch.
- If guessed wrong, squash the results
- **But the side-effect (cache state change) remain**

Meltdown

```
; rcx = kernel address
; rbx = probe array
retry:
mov al, byte [rcx]
shl rax, 0xc
jz retry
mov rbx, qword [rbx + rax]
```

- This is it.

Meltdown

```
; rcx = kernel address
; rbx = probe array
retry:
mov al, byte [rcx]
shl rax, 0xc
jz retry
mov rbx, qword [rbx + rax]
```

- Step 1: load an attacker chosen kernel address into a register. This would raise an exception but it may take some time.

Meltdown

```
; rcx = kernel address
; rbx = probe array
retry:
mov al, byte [rcx]
shl rax, 0xc
jz retry
mov rbx, qword [rbx + rax]
```

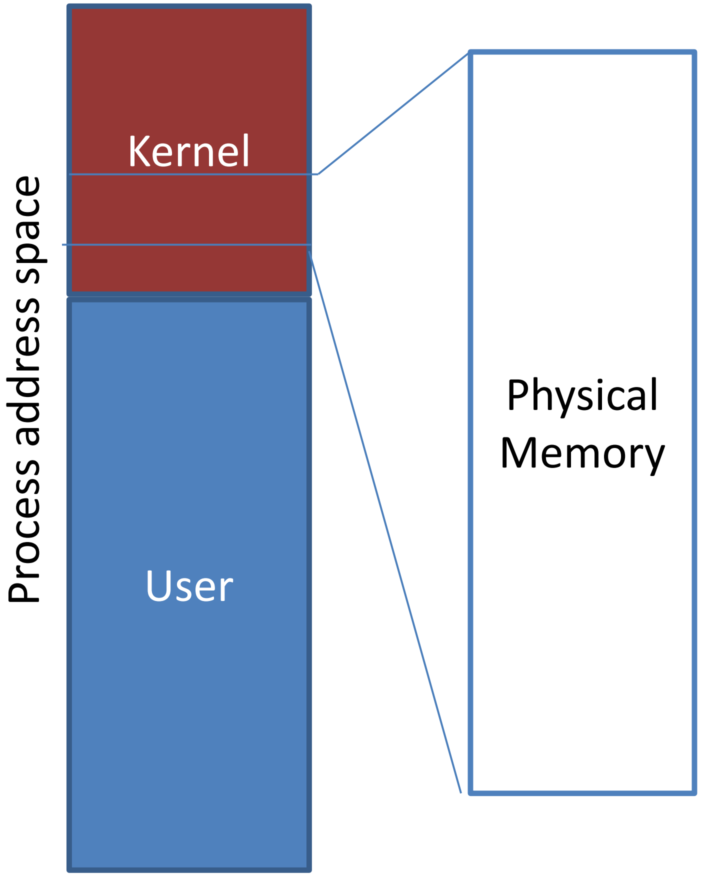
- Step 2: access memory based on the secret content of the *rax* register. This access will be in the cache.

Meltdown

```
; rcx = kernel address
; rbx = probe array
retry:
mov al, byte [rcx]
shl rax, 0xc
jz retry
mov rbx, qword [rbx + rax]
```

- Step 3: measure the access timing of the probe array (one per page) to determine which is in the cache. Which in turn tell what was the content of the kernel address [*rcx*]

Meltdown




```
79cbb30: 616f 61 4e 6b 32 38 46 31  34 67 65 68 61 7a 34 |aoaNk28F14gehaz4|
79cbb40: 5a74 4d 79 78 68 76 41 57  69 69 63 77 59 62 61 |ZtMyxhvAWiicwYba|
79cbb50: 356a 4c 76 4d 70 4b 56 56  32 4b 6a 37 4b 5a 4e |5jLvMpKVV2Kj7KZN|
79cbb60: 6655 6c 6e 72 38 64 74 35  54 62 43 63 7a 6f 44 |fUlnr8dt5TbCczoD|
79cbb70: 494e 46 71 58 6d 4a 69 34  58 50 39 62 43 53 47 |INFqXmJi4XP9bCSG|
79cbb80: 6c4c 48 32 5a 78 66 56 44  73 4b 57 39 34 68 6d |lLH2ZxfVDsKW94hm|
79cbb90: 3364 2f 41 4d 41 45 44 41  41 41 41 41 51 45 42 |3d/AMAEDAAAAAQEB|
79cbba0: 4141 41 41 41 41 3d 3d XX  XX XX XX XX XX XX XX |AAAAAA==........|
79cbbb0: XXXX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
79cbbc0: XXXX XX 65 2d 68 65 61 64  XX XX XX XX XX XX XX |...e-head.......|
79cbbd0: XXXX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
79cbbe0: XXXX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
79cbbf0: XXXX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
79cbc00: XXXX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
79cbc10: XXXX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
79cbc20: XXXX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
79cbc30: XXXX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
79cbc40: XXXX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
79cbc50: XXXX XX XX 0d 0a XX 6f 72  69 67 69 6e 61 6c 2d |.......original-|
79cbc60: 7265 73 70 6f 6e 73 65 2d  68 65 61 64 65 72 73 |response-headers|
79cbc70: XX44 61 74 65 3a 20 53 61  74 2c 20 30 39 20 44 |.Date: Sat, 09 D|
79cbc80: 6563 20 32 30 31 37 20 32  32 3a 32 39 3a 32 35 |ec 2017 22:29:25|
79cbc90: 2047 4d 54 0d 0a 43 6f 6e  74 65 6e 74 2d 4c 65 | GMT..Content-Le|
79cbca0: 6e67 74 68 3a 20 31 0d 0a  43 6f 6e 74 65 6e 74 |ngth: 1..Content|
79cbcb0: 2d54 79 70 65 3a 20 74 65  78 74 2f 68 74 6d 6c |-Type: text/html|
79cbcc0: 3b20 63 68 61 72 73 65 74  3d 75 74 66 2d 38 0d |; charset=utf-8.|
79cbcd0: 0a53 65 72 76 65 72 3a 20  54 77 69 73 74 65 64 |.Server: Twisted|
79cbce0: 5765 62 2f 31 36 2e 33 2e  30 0d 0a XX 75 6e 63 |Web/16.3.0...unc|
79cbcf0: 6f6d 70 72 65 73 73 65 64  2d 6c 65 6e XX XX XX |ompressed-len...|
```

- Entire physical memory is 1-to-1 mapped into part of kernel memory. So, you can dump it.

Summary

- System security
  - Increasingly important
    - Bank accounts (money theft)
    - Cars (remote control of steering, breaking of Jeep cars)
    - Airplanes (remote control of drones)
  - Understand threats in both **software** and **hardware**
  - Exploiting the threats often require deep understanding in OS

KU THE UNIVERSITY OF KANSAS