Protection

Read from the textbook

- Ch17.6 Implementation of the Access Matrix
- Ch17.7 Revocation of Access Rights
- Ch17.10 Capability-Based Systems

You may want to see the Wikipedia entries on:

- http://en.wikipedia.org/wiki/Confused_deputy_problem
- http://en.wikipedia.org/wiki/Capabilities
- http://en.wikipedia.org/wiki/Selinux

Implementation of Access Matrix

- There is too much information required in an access matrix.
- We have cells for every combination of domains and objects.
- We could implement it as a sparse matrix but most OSs use one of two possible representations (and sometimes a mixture of both).
    - Only hold information on the rows – each row corresponds to the access rights of a domain over all objects it can use. If the domain has no rights over an object no information is stored. This approach is known as **capability lists**.
    - Only hold information on the columns – each column represents the access rights held over this object. No information is stored about domains that have no access. This approach is known as *access lists*.

Confused Deputy Problem

- This is connected with the changing domains problem from last lecture.
- A deputy is a process with special access rights to objects. e.g., the Unix `passwd` program has access to the password files.
- Another process has access to the deputy. e.g., you can change your own password.
- The process asks the deputy to do something for it. e.g., change someone else's password
- The deputy can do this but should it?
- If the deputy doesn't check we have a security hole.
  - Capabilities remove this hole.

Capabilities

- A **capability** is a permission to access an object in certain ways.
- Capabilities are stored with domains. They always refer to the object and the access rights.
- So a domain has a capability list.

  <f1, “read,write”>

  <f2, “execute”>

  <d2, “control”>
- When a process needs to access a protected resource the capability is passed with the request. The capability is checked by the reference monitor to make sure the access is permitted.
- We need to ensure that a process executing in a domain cannot freely change any of its capabilities.
- The capability list is itself a protected object.

Keeping Capabilities Safe

- How do we stop a process altering its capabilities?
  - In particular we don't want a process making new capabilities without strict controls.
  - So capabilities should not be forgeable.
- We can use hardware or encryption.
- On a single machine we can store all capabilities in protected kernel memory (so that the user level process cannot directly access them or add to them).
- Tagged architectures – some machines have extra bits for every word in memory. Thus integers can be distinguished from strings from floats from capabilities. Only the OS is allowed to create or change capabilities.
- With distributed systems we need to encrypt capabilities. With a public key anyone can check them but they cannot be created.

Use of Capabilities

- Regardless of how they are protected – capabilities must only be created by the OS.
- When a new object is created the OS should construct an owner capability for the creator process.
- Usually the owner can pass the capability on to other processes (or domains).
  - Like a secure password – secure because the capability cannot be forged or altered.
- We must also ensure that capabilities can't be snooped off the network.
- Capabilities are now widely used – many microkernel OSs (like Mach) use capabilities.
- Kerberos tickets are capabilities.

Problems with Capabilities

- It is difficult to determine which domains have access rights to a given object.
    - not only are the capabilities associated with domains but they can also be passed on – keeping track of them is a problem.
- Similarly because it is difficult to find references to all domains with particular capabilities it is difficult to revoke access permissions.
- Revoking capabilities
    - keep track of all domains with the capability (made difficult if they have the capability to pass on capabilities)
    - indirection – the capability is not really to the object but to an intermediate object which points to the object (we change the pointer to null). All domains with the old capability no longer have access.
    - reacquisition - have an expiry time and then request the capability again. (In this case we need another way of determining if the capability should be given.)

Access Control Lists

- Each object has a list of domains and their access rights. This list is an ACL.
- An access control list (ACL) could look like:

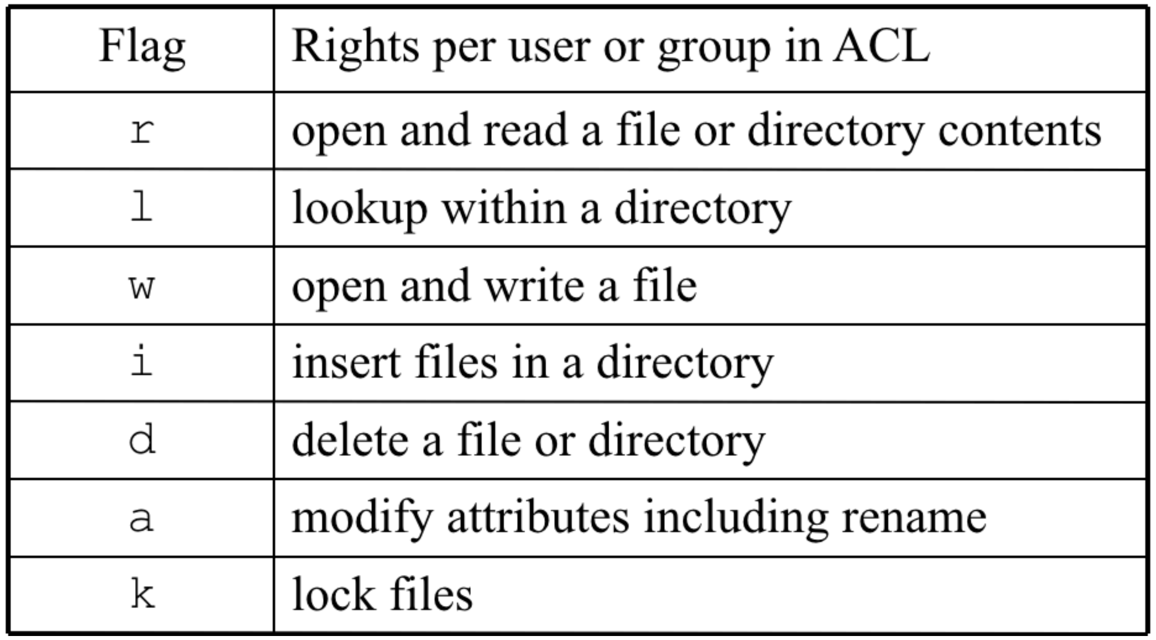
  <d1, “read,write”>

  <d2, “execute”>

  <d3, “control”>
- When a request comes to the object from a specific domain for a particular access the reference monitor checks the ACL to see if the access is allowed.
- ACLs don’t have any revocation problems. The ACL just gets changed.

AFS vs NTFS ACLs

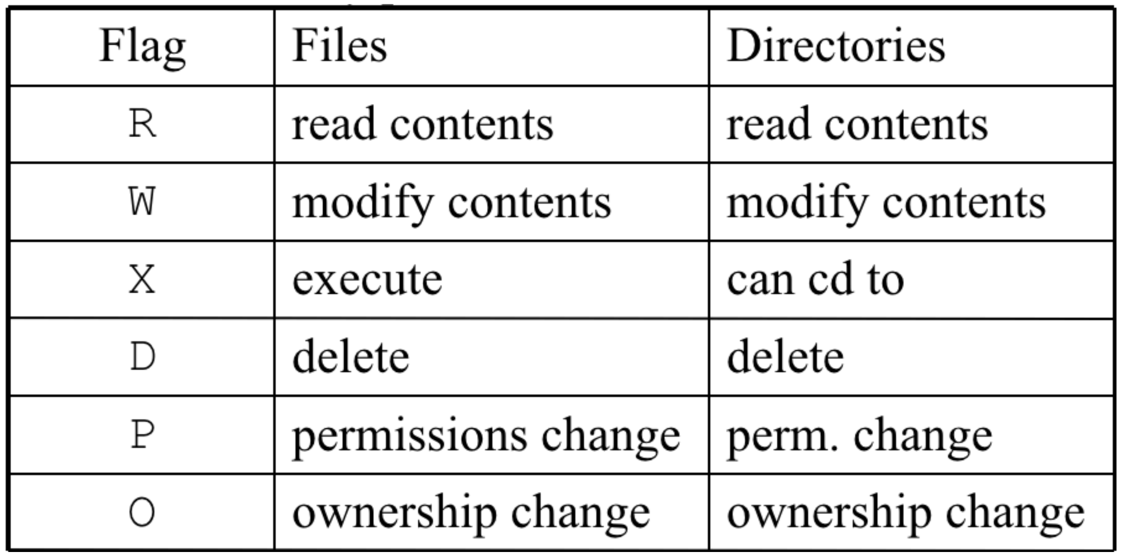
AFS uses ACLs for its access permissions. But these are only on directories. It has a number of possible permissions:



Flag	Rights per user or group in ACL
`r`	open and read a file or directory contents
`l`	lookup within a directory
`w`	open and write a file
`i`	insert files in a directory
`d`	delete a file or directory
`a`	modify attributes including rename
`k`	lock files

There are also shorthand forms:
write = `rlwidk` read = `rl` all = `rlwidak`

NTFS - it is possible to access a file even if you don’t have the directory permissions.



Flag	Files	Directories
`R`	read contents	read contents
`W`	modify contents	modify contents
`X`	execute	can cd to
`D`	delete	delete
`P`	permissions change	perm. change
`O`	ownership change	ownership change

Checking your ACLs

OSs that support ACLs provide tools to deal with them e.g. `getfacl/setfacl`

NT has `cacls`

```
>cacls profile
C:\Documents and Settings\...\system\profile
  BUILTIN\Users:R
  EC\rshe019:F
  NT AUTHORITY\SYSTEM:F
  BUILTIN\Administrators:F
```

F – means full access.

Problems with ACLs

- They slow down file search operations – all ACLs (which can be long) have to be checked for all directories.
  - AFS imposes a limit, commonly set to 20.
- They are considered by some as unnecessarily complex:
- "Ironically, for all the flexibility that ACLs offer, they have proven to be confusing and difficult to understand, and the extra functionality they provide is dwarfed by the feeling of dread which they instill in administrators and users alike." – Mark Burgess, **Principles of Network and System Administration**
- Many prefer the simplicity of Unix file protection mechanism.

Reducing Information

- We still have the problem with lots of permission information in our systems.
- Both ACL and capability systems can have default access to objects to reduce the amount of necessary information.
- ACLs (with hierarchical objects, such as files) can inherit permissions from objects above them.
- A capability to a directory could give you access to all files in the directory.
    - This is less flexible. What if we want to put a hidden file in the directory which we don’t want accessed?

UNIX Permissions

- UNIX associates permission bits with every file.
- read/write/execute bits for files associated with the files owner, the group and everyone
- You can change the permission code of a file or directory only if you own it or if you have superuser authority.
- The permissions are:

`r` Read permission. **list files in directory**

`w` Write permission. **add/remove files into directory**

access file content

`x` Execute permission for files, search permission for directories.

`X` Execute permission only if file is a directory or at least one execute bit is set.

`s` Set-user-ID or set-group-ID permission.

There is also a sticky bit – `t` permission but this depends on the version of UNIX.

- Remember that the permissions for owner, group and everyone are checked in that order. So what permissions do I have on this file?

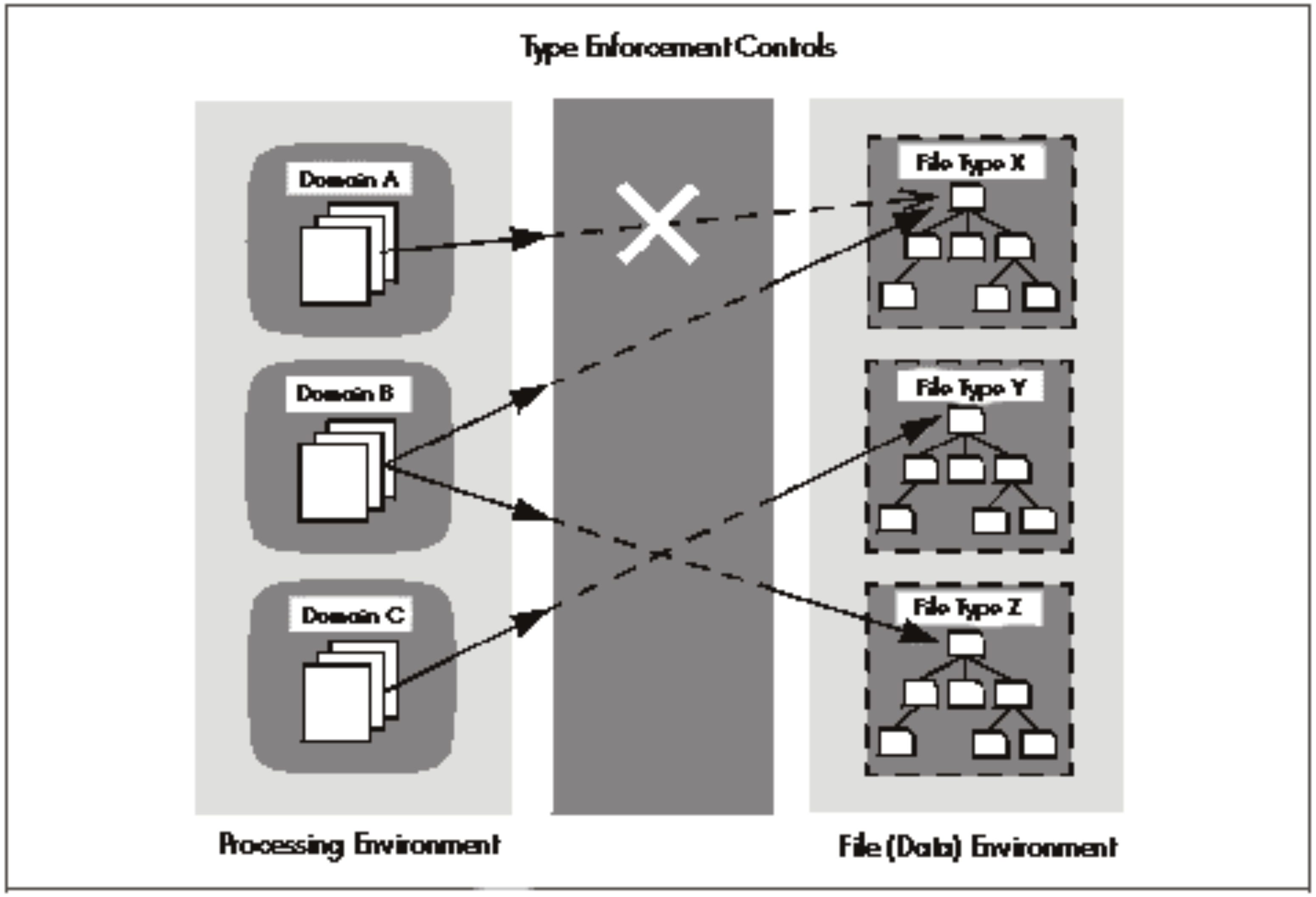
```
-r---w-rwx   1 robert-s staff      program
```

Adding Levels to Unix Like Systems

- The OS is divided into separate domains.
    - kernel
    - system and user applications
        - applications are separate from each other
- Each domain only has the minimum permissions (on files, sockets, directories) it needs to do its job.
- Each domain has limited access to system calls and file types (e.g. httpd_t)
- Use a strict file typing system.
- There is no global superuser.
    - Each domain has its own administrator and the administrator of one domain has no power over another.
- So how is global administration done?
    - The system is restarted with a different administration kernel (without any connections to networks)

Domains & Types

- The checking of access is built-in to the kernel and cannot be circumvented.



Meltdown

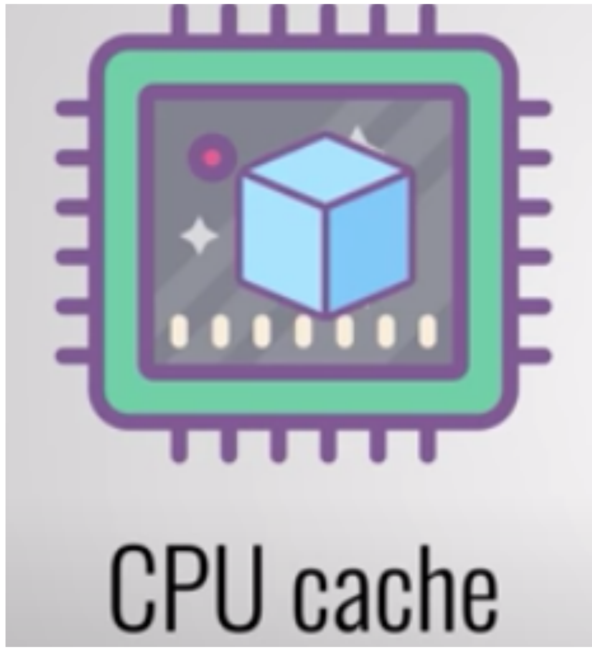
Did no one see this in 20 years?
https://meltdownattack.com

Simple explanation of Spectre:
https://www.youtube.com/watch?v=mgAN4w7LH2o
9:10 ->

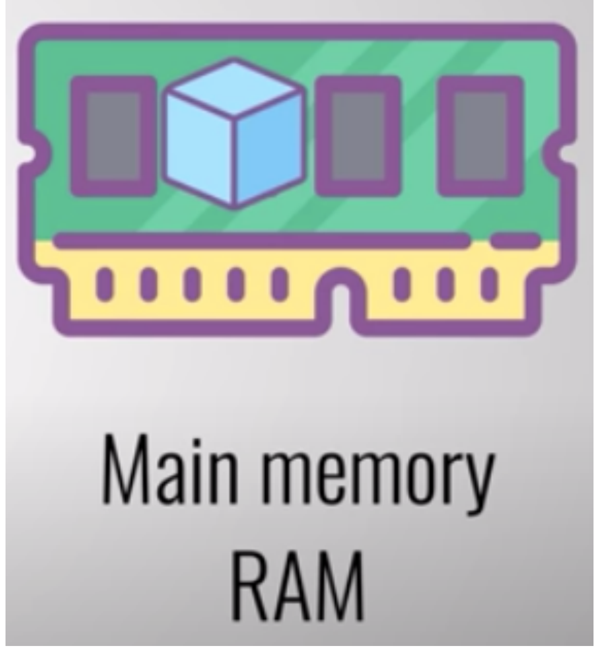
Spectre and Meltdown Attacks

Speculative Execution

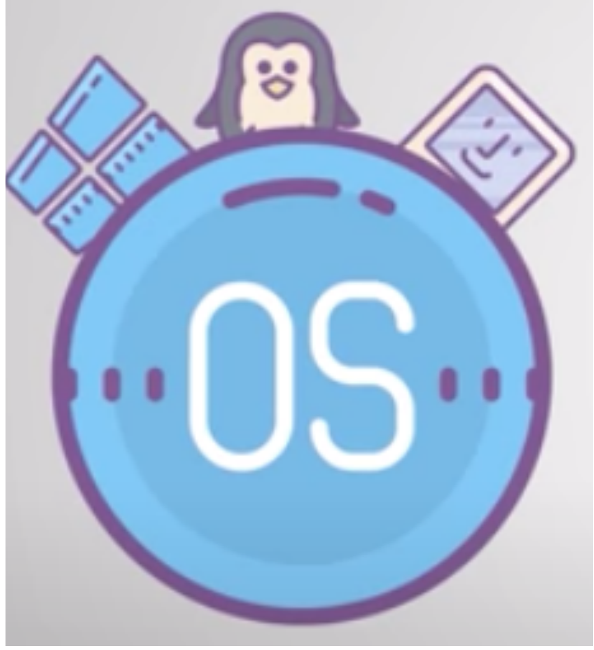
- The processors guess what the outcome of an instruction will be
- Execute all the subsequent steps in the background



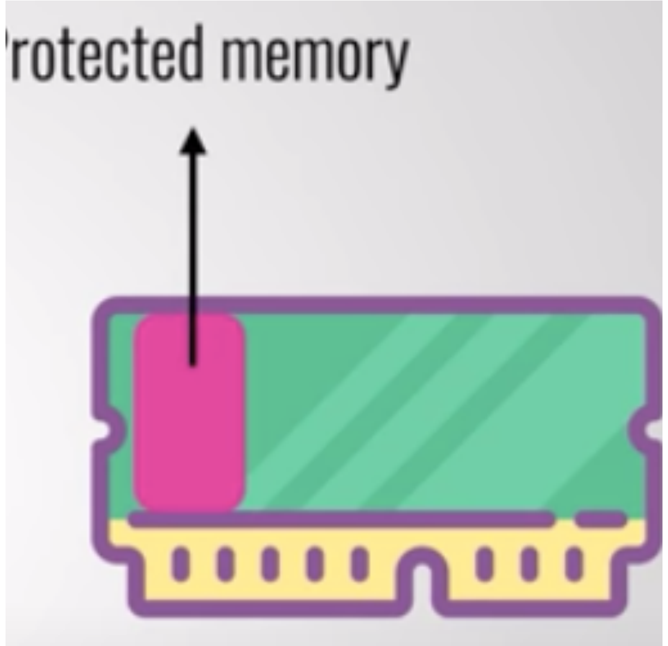
(a)



(b)



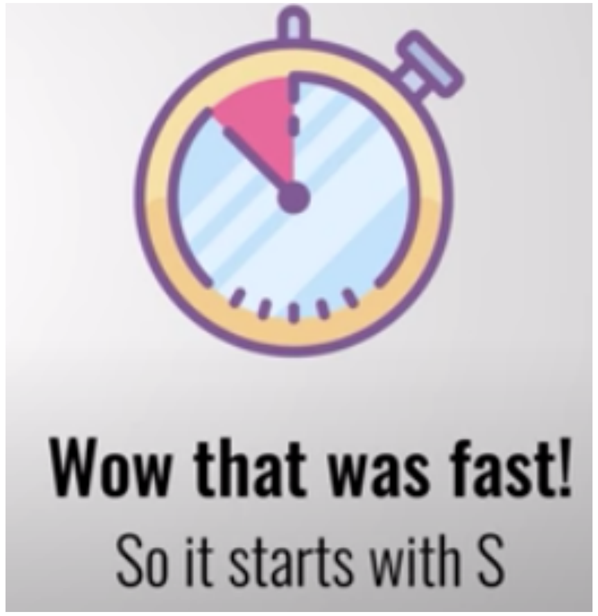
(c)



(d)

https://www.youtube.com/watch?v=mgAN4w7LH2o

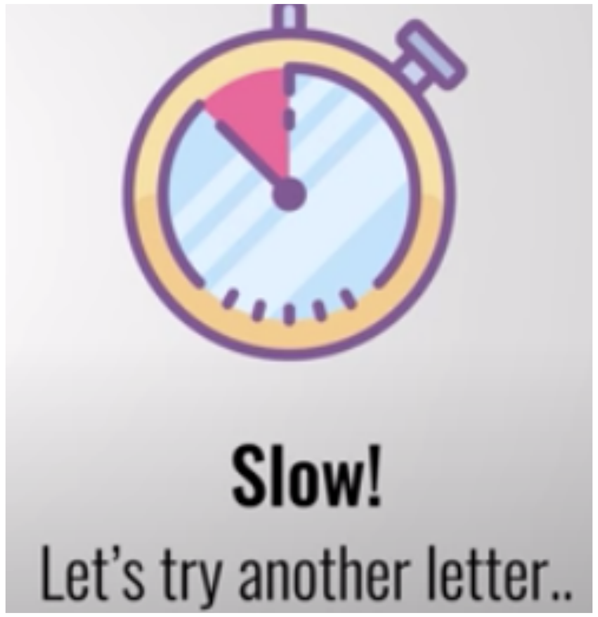
```
if ( readMemory(182379) == “S” ){
        readPixel(1)
}
```



Wow that was fast!

So it starts with S

(a)



Slow!

Let’s try another letter..

(b)

(c)

https://www.youtube.com/watch?v=mgAN4w7LH2o

Modern Processors and OSs

- Cache memory
- Checks on memory access - in particular kernel memory
- Partial execution and out of order execution
- Undo effects of execution if access not allowed

- Most OSs map the same kernel space into all processes
- Lots of memory mapped into kernel space

- And everything works as specified so we are safe, right?
    - Race condition
    - Side channel timing attack

The Exploit

- Let’s try to access some memory.
- Our access uses a memory value "A" (in kernel space) as an offset (a bit like the index to an array). e.g read base[A * 4096]
  - why 4096? (page size, every value of A gives a different page)
- "A" is restricted, we will get an address violation from user mode.
  - and hence the instruction doesn’t get carried out.
- But a race condition exists between accessing the value in "A" and reacting to the address violation. So the value in base[A * 4096] may have reached the CPU cache (and this is the case with most processors in the last 20 years).
- Now the timing attack.

The Timing Attack

- We don’t know the value of "A" but we can now read all values of base[0 * 4096] to base[255 * 4096]. (Actually it is quicker to deal with values over bit ranges.)
- We time the accesses. The cached value of base[A * 4096] will be retrieved faster than the others.
- We can then say what the value of A probably was.
- We have thus accessed a value from within kernel memory while we are running in user mode. AAARGH!!!

- Passwords, tokens, anything in kernel memory.

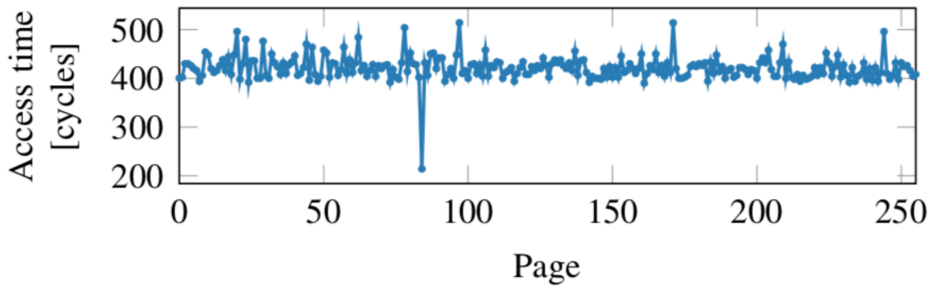


Figure 4: Even if a memory location is only accessed during out-of-order execution, it remains cached. Iterating over the 256 pages of probe_array shows one cache hit, exactly on the page that was accessed during the out-of-order execution.

From the Meltdown paper:
https://meltdownattack.com/meltdown.pdf

That is Bad Enough

- Now a user level process can read anything in kernel memory.
- It is even worse than that
  - Many OSs map the user memory of all processes (or large parts of it) into the kernel address space. This is for increased efficiency.
- So all memory from all processes is accessible to a user level process carrying out the meltdown attack.
- Is there a fix?

Kernel Page-Table Isolation

- KPTI
- We keep a separate page table which is used when in kernel mode.
- Now a process in user space cannot even try to access the kernel addresses as they are not mapped into its address space.
- This means every transition between user mode and kernel mode will be slowed down some as we move to different page tables.
  - At one level this is just a pointer change, but don’t forget the TLB and other memory caches.
  - Estimates are that KPTI leads to 5-15% reduction in throughput.
    - Some researchers claim 30%, others effectively none, it depends on several factors including the number of system calls made by active processes.

Before Next Time

Read from the textbook

- Ch16.1 The Security Problem
- Ch16.4 Cryptography as a Security Tool
- Ch16.5 User Authentication