

ECE/CS230

Computer Systems Security

Charalambos (Harrys) Konstantinou

<https://sites.google.com/view/ececs230kaust>

OS

Memory and address protection

Read more: <https://flylib.com/books/en/4.270.1.46/1/>

- Prevent one program from corrupting other programs or data, operating system and maybe itself
- Often, the OS can exploit **hardware support** for this protection, so it's cheap
- Memory protection is part of translation from virtual to physical addresses
 - Memory management unit (MMU) generates exception if something is wrong with virtual address or associated request
 - OS maintains mapping tables used by MMU and deals with raised exceptions

Memory and address protection

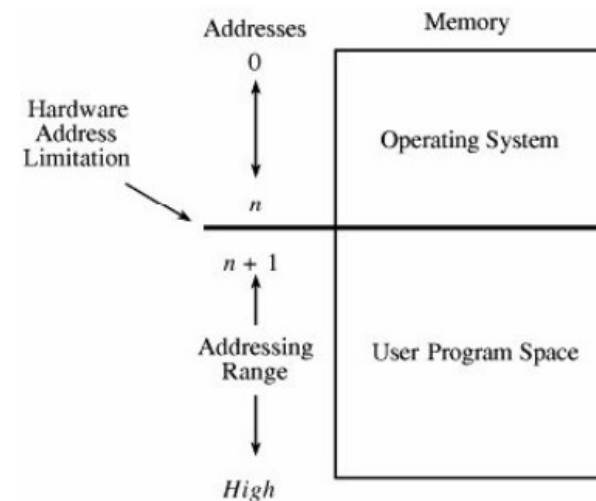
- The most obvious problem of multiprogramming is **preventing** one program from affecting the data and programs in the **memory space** of other users
- **Fence**: The simplest form of memory protection for single user OS
 - Prevent a faulty user program from destroying part of the resident portion of the OS:
 - Exception if memory access below address in fence register
 - Protects operating system from user programs
 - Single-user OS only

Memory and address protection

- **Fence**: The simplest form of memory protection for single user OS
 - Prevent a faulty user program from destroying part of the resident portion of the OS:
 - Exception if memory access below address in fence register
 - Protects operating system from user programs
 - Single-user OS only

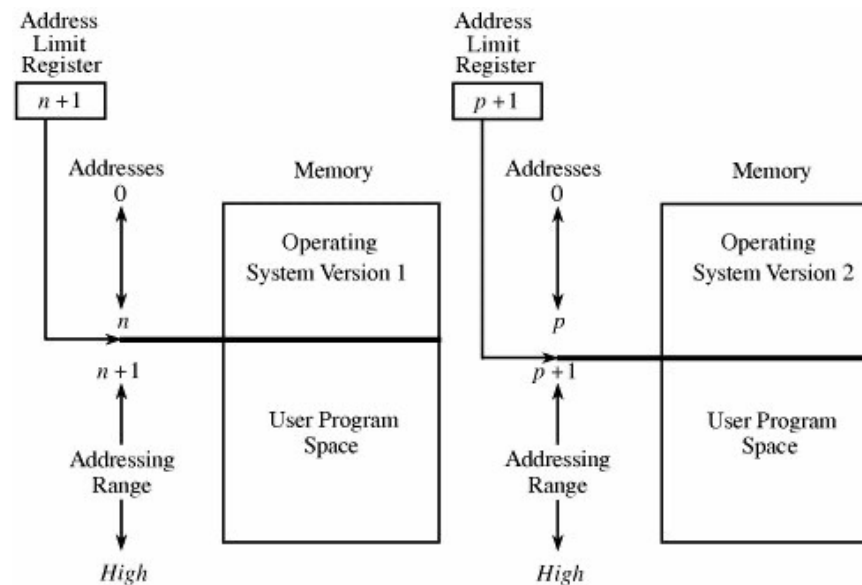
- **Fixed Fence**

- Predefined space used for the OS
- If less than that is required, it is wasted
- If more space is needed we cannot grow it



Fence Register

- Fence Register (Hardware Register): containing the address of the end of the Oss
 - This value could be changed
 - Each time a user program issues a modification of data it will check this value to allow it or prohibit it
 - One way protection only
 - A user from the OS
 - Not a user from another user



Relocation

- If the OS can be assumed to be of a fixed size, programmers can write their code assuming that the program begins at a constant address.
 - This make it easy determine the starting address of any object in the program
 - If the OS is allowed to change in size
 - i.e. if a new OS is greater or smaller, then programs must be written in a way that does not depend on displacement
 - **Relocation**: is the process of taking a program written as if it began at address 0 and changing all addresses to reflect the actual address at which the program is located in memory.
 - adding a constant
 - **Relocation Factor**: to each address of the program.
 - The relocation factor is the starting address of the memory assigned for the program.

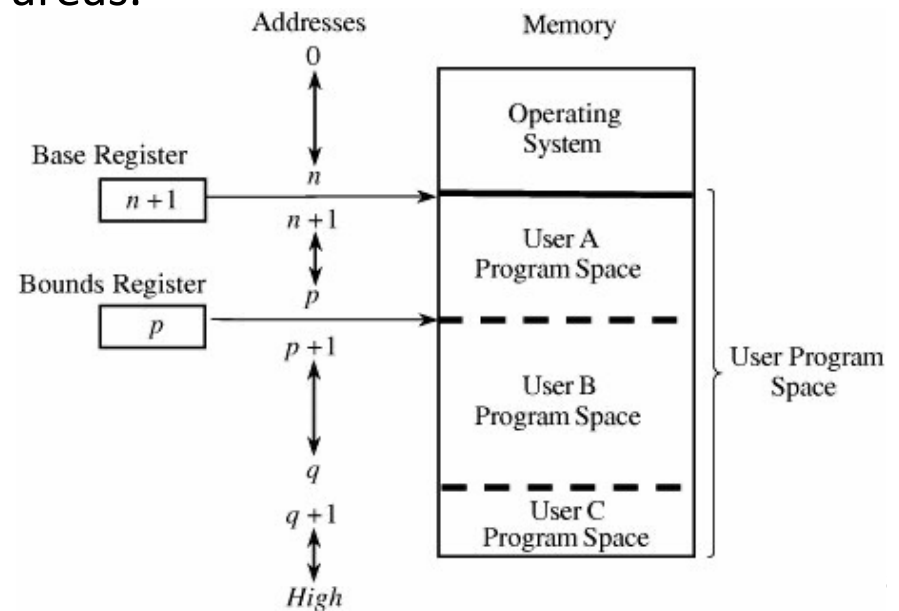
Base/Bounds Registers

- A major advantage of an OS with fence registers is the ability to relocate
 - Useful especially in multiuser environment
 - Users can know in advance where the program will be loaded for execution
- The relocation register solves the problem by providing a base or starting address
 - All addresses inside a program are offsets from the base address

Base/Bounds Registers

- Fence registers provides a lower bound {a starting address) but not an upper one
 - Upper bounds can be useful in knowing how much space is allotted, and in checking for overflows into forbidden areas.
 - It is called is a bound register

Each program address is forced to be above the base address and below the bound address



Base/Bounds Registers

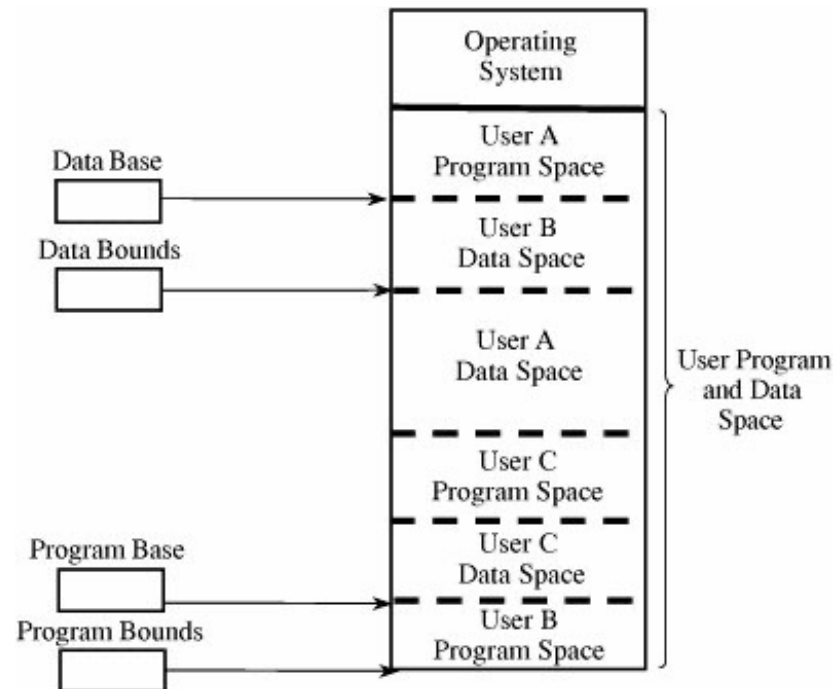
- The technique protects a program's addresses from modification by another user
- When executing changes from one user's program to another's
 - The operating systems must change the contents of the base and bound register to reflect the true address space of the user
 - **Context Switch**: Part of the general preparation done by the OS when transferring control from one user to another

Base/Bounds Registers

- With base/bound register the user is protected from outside users
 - Outside users are protected from errors in any other user's program
- **Erroneous addresses inside a user's address space can still affect the program**
 - A user may write over the address of the instruction by error in a way he destroys his program
 - It only affects his own program not others

Base Bound Registers

- We can **solve the problem of overwriting on the user own code by incorporating a Base and Bound Register in the user space**



Tagged Architecture


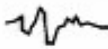
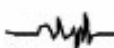



- Another problem with Base/Bound registers for protection or relocation is their contagious nature.
 - Each pair of registers confines access to a consecutive range of addresses
 - A compiler rearrange data sections so that all sections are adjacent
 - However in some cases we want to protect data values but not all of them.
 - Ex. Not all data just a personal record like salary but not address
 - Also a programmer to insure integrity by prohibiting writing rights
 - This schemes also protect against user's owners on errors

Tagged Architecture

- Every word of a machine memory has one or more extra bits to identify the access rights to that word.
 - These access bits can only be set by privileged (O.S) instructions
 - The bits are tested every time an instruction accesses that location

Tagged Architecture

- As shown in the figure one memory location can be protected while the others not
- Tag bits are usually small in size
 - IBM used a tag to control integrity and access
- Disadvantages:
 - wastes memory
 - compatibility issues: Fundamental change of all (old) OS code

Tag	Memory Word
R	0001
RW	0137
R	0099
X	
X	
X	
X	
X	
X	
R	4091
RW	0002

Code: R = Read-only RW = Read/Write
X = Execute-only

Protection techniques recap

- Fence register

- Exception if memory access below address in fence register
- Protects operating system from user programs
- Single-user OS only

- Base/bounds register pair

- Exception if memory access below/above address in base/bounds register
- Different values for each user program
- Maintained by operating system during context switch
- Limited flexibility

Protection techniques recap

- Tagged architecture

- Each memory word has one or more extra bits that identify access rights to word
- Very flexible
- Large overhead
- Difficult to port OS from/to other hardware architectures

- Segmentation

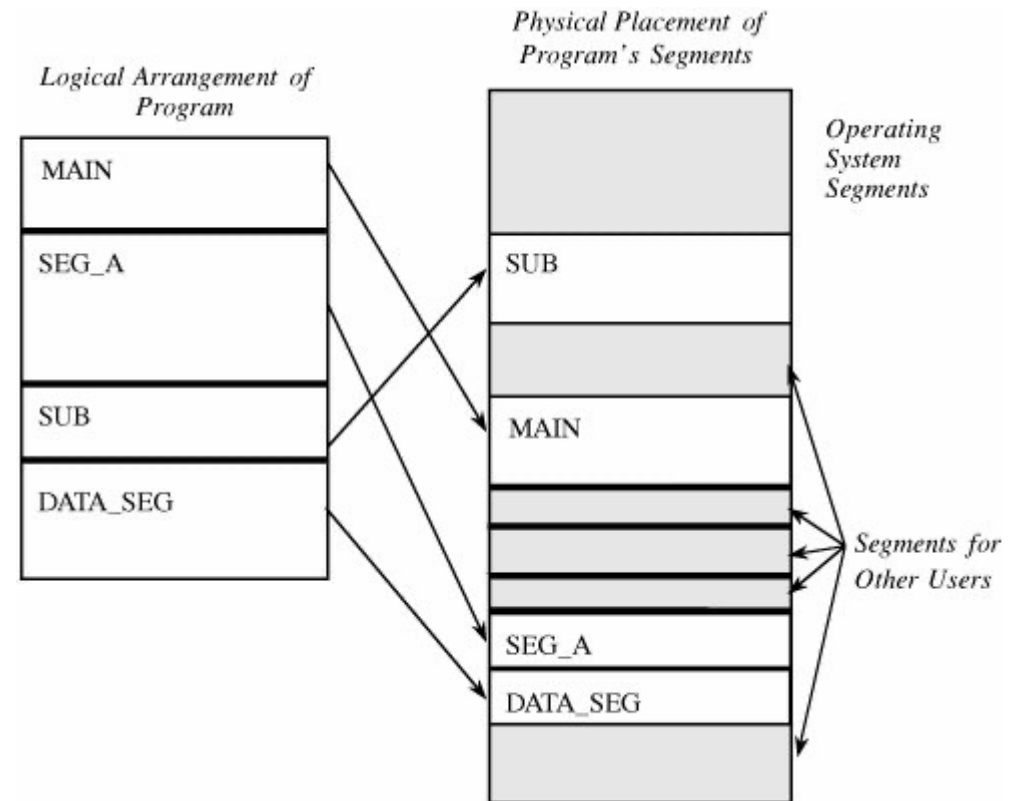
- Paging

Segmentation

- **Segmentation**: involves the simple notion of dividing a program into separate pieces.
 - Each piece has a logical unity, exhibiting a relationship among all of its code or data values
 - For example, a segment may be the:
 - code of a single procedure
 - data of an array
 - collection of all local data values used by a particular module
- **Segmentation** was developed as a feasible means to produce the effect of the equivalent of an unbounded number of base/bounds registers.
 - In other words, segmentation allows a program to be divided into many pieces having different access rights.

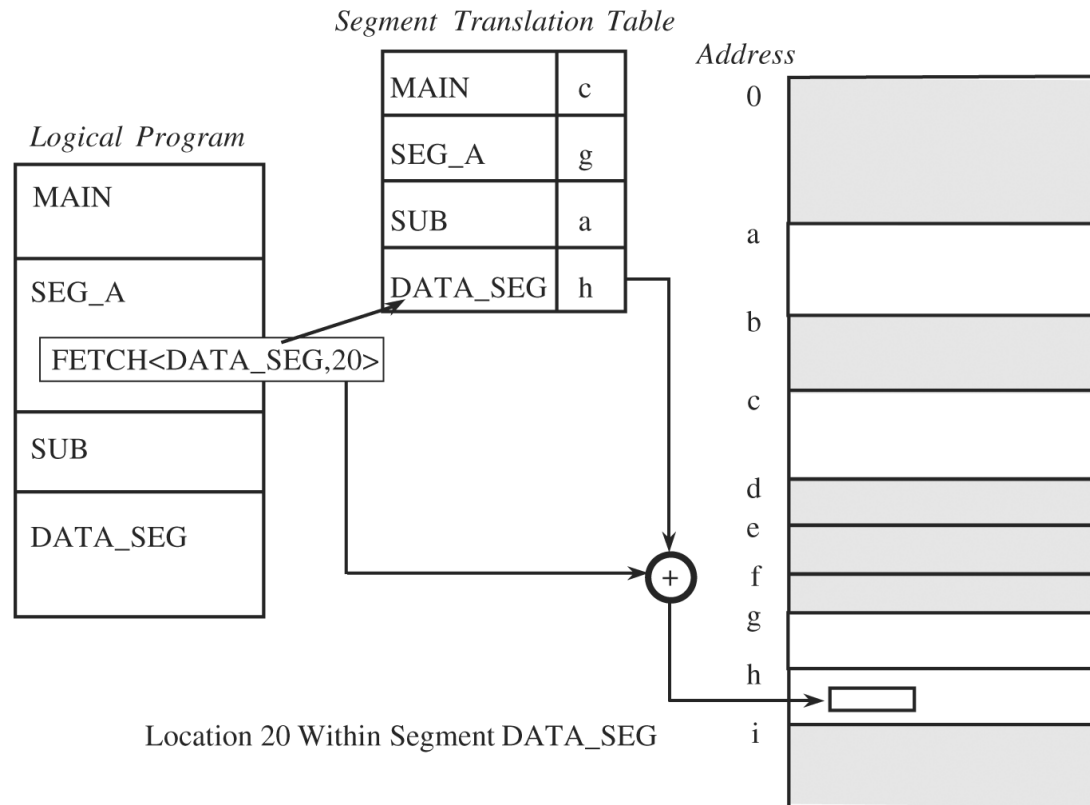
Segmentation

- Each segment has a unique name
 - A code or data item is addressed as the pair <name, offset>
 - Name: the name of the segment containing the data item
 - Offset: the location within the segment i.e. the distance from the start of the segment



- Logical and Physical representation of segments
- A program can be pictured as long Collection of segments that can be separately relocated

Segment table



Protection attributes and segment length are missing in table

Segmentation

- The O.S. must maintain a table of segment names and their true address in memory
- When a program generates an address of the form <name, offset>:
 - The OS looks up name in the segment directory and determines its real beginning memory address.
 - To that address the operating system adds offset, giving the true memory address of the code or data item
 - Two processes that need to share access to a single segment would have the same segment name and address in their segment tables.

Segmentation

- A user's program does not know what true memory addresses it uses.
 - It has no way and no need to determine the actual address associated with a particular <name, offset>.
 - The <name, offset> pair is adequate to access any data or instruction to which a program should have access

Segmentation Advantages

Advantages:

- The OS can place any segment at any location or move any segment to any location, even after the program begins to execute
 - the OS needs only update the address in that one table when a segment is moved.
- A segment can be removed from main memory (and stored on an auxiliary device) if it is not being used currently
- Every address reference passes through the operating system, so there is an opportunity to check each one for protection.

Segmentation Security Benefits

Segmentation offers these security benefits:

- Each address reference is checked for protection.
- Many different classes of data items can be assigned different levels of protection.
- Two or more users can share access to a segment, with potentially different access rights.
- A user cannot generate an address or access to an unpermitted segment

But...

Segmentation Vulnerability

- One protection difficulty inherent in segmentation concerns **segment size**
- Each segment has a particular size.
- However, a program can generate a reference to a valid segment name, but with an offset beyond the end of the segment.
 - For example, reference <A,9999> looks perfectly valid, but in reality segment A may be only 200 bytes long. If left unplugged, this security hole could allow a program to access any memory address beyond the end of a segment just by using large values of offset in an address.

Segmentation Vulnerability

- This problem cannot be stopped during compilation or even when a program is loaded, because effective use of segments requires that they be allowed to grow in size during execution.
- For example, a segment might contain a dynamic data structure such as a stack. Therefore, **secure implementation of segmentation requires checking a generated address to verify that it is not beyond the current end of the segment referenced.**
 - Although this checking results in extra expense (in terms of time and resources), segmentation systems must perform this check; the segmentation process must maintain the current segment length in the translation table and compare every address generated.

Review of segmentation

- Each program has multiple address spaces (**segments**)
- Different segments for code, data, and stack
 - Or maybe even more fine-grained, e.g., different segments for data with different access restrictions
- Virtual addresses consist of two parts:
 - **<segment name, offset within segment>**
- OS keeps mapping from segment name to its base physical address in **Segment Table**
 - A segment table for each process
- OS can (transparently) relocate or resize segments and share them between processes
- Segment table also keeps protection attributes

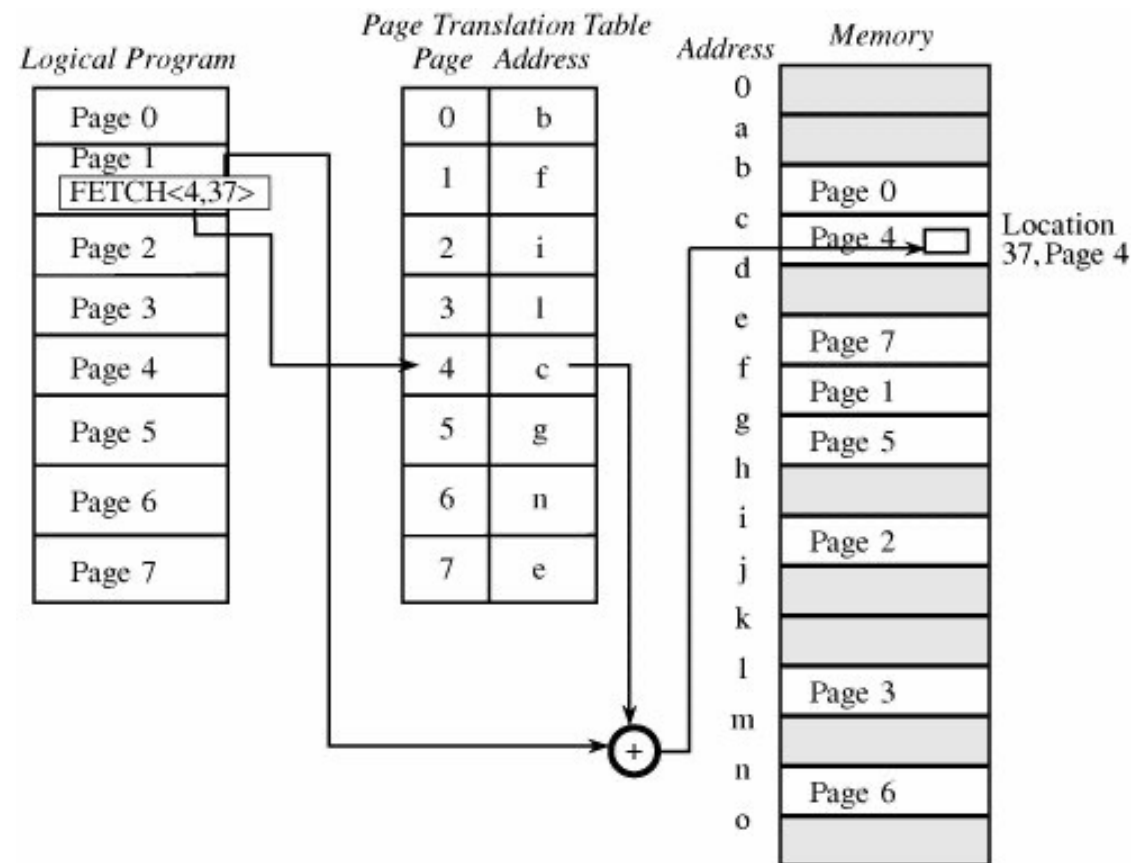
Review of segmentation

- Advantages:
 - Each address reference is checked for protection by hardware
 - Many different classes of data items can be assigned different levels of protection
 - Users can share access to a segment, with potentially different access rights
 - Users cannot access an unpermitted segment
- Disadvantages:
 - External fragmentation
 - Dynamic length of segments requires costly out-of-bounds check for generated physical addresses
 - Segment names are difficult to implement efficiently

Paging

- One alternative to segmentation is **paging**.
- Program (i.e., virtual address space) is divided into equal-sized chunks (**pages**)
- Physical memory is divided into equal-sized chunks (**frames**)
- Frame size equals page size
- Virtual addresses consist of two parts:
 - **<page #, offset within page>**
 - # bits for offset = $\log_2(\text{page size})$
- OS keeps mapping from page # to its base physical address in **Page Table**
- Page table also keeps memory protection attributes

Paging



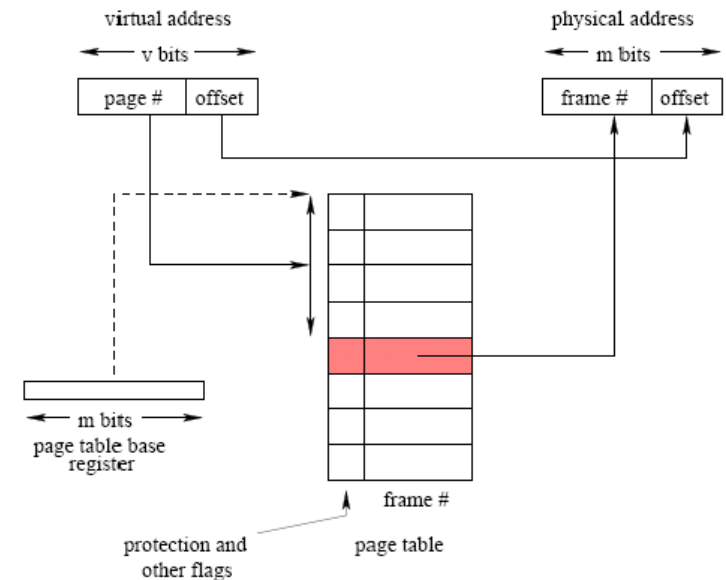
Paging

Unlike segmentation, all pages in the paging approach are of the same fixed size, so fragmentation is not a problem.

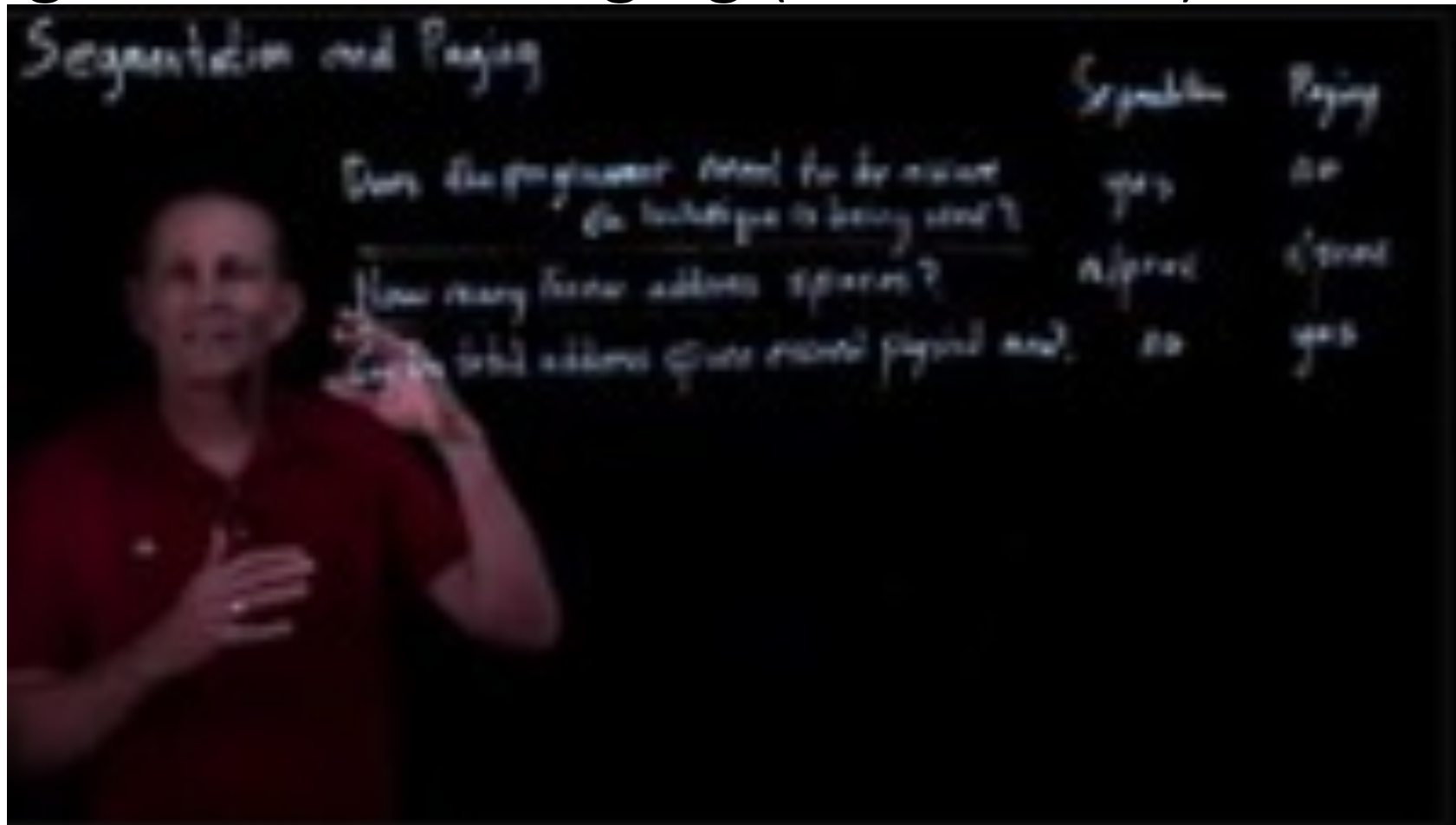
- Each page can fit in any available page in memory, and thus there is no problem of addressing beyond the end of a page.
- Example:
 - Consider a page size of 1024 bytes ($1024 = 2^{10}$)
 - 10 bits are allocated for the offset portion of each address.
 - A program cannot generate an offset value larger than 1023 in 10 bits.
 - Moving to the next location after $\langle x, 1023 \rangle$ causes a carry into the page portion, thereby moving translation to the next page.
 - During the translation, the paging process checks to verify that a $\langle \text{page}, \text{offset} \rangle$ reference does not exceed the maximum number of pages the process has defined.

Review of paging

- Advantages:
 - Each address reference is checked for protection by hardware
 - Users can share access to a page, with potentially different access rights
 - Users cannot access an unpermitted page
 - Unpopular pages can be moved to disk to free memory
- Disadvantages:
 - Internal fragmentation
 - Assigning different levels of protection to different classes of data items not feasible



Segmentation & Paging (short video)

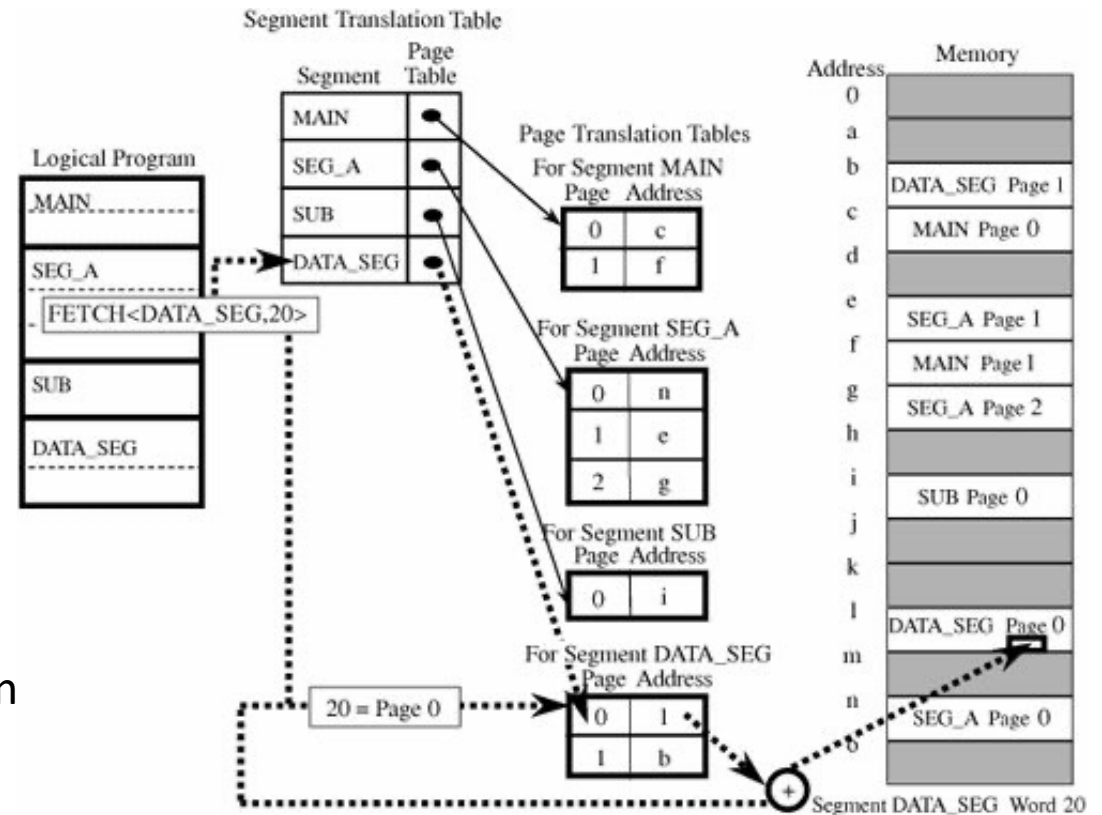


Combining Paging with Segmentation [x86] (video)



Combining Paging with Segmentation

- We have seen how paging offers implementation efficiency, while segmentation offers logical protection characteristics
 - Each approach has drawbacks as well as desirable features, the two approaches have been combined.
 - The programmer could divide a program into logical segments.
 - Each segment was then broken into fixed-size pages.
 - The segment name portion of an address was an 18-bit number with a 16-bit offset. The addresses were then broken into 1024-byte pages



x86 architecture

- x86 architecture has both segmentation and paging
 - Linux and Windows use both
 - Only simple form of segmentation, helps portability
 - Segmentation cannot be turned off on x86
- Memory protection bits indicate no access, read/write access or read-only access
- Most processors also include **NX (No eXecute) bit**, forbidding execution of instructions stored in page
 - E.g., make stack/heap non-executable
 - Does this avoid all buffer overflow attacks?

Outline

- Protection in general-purpose operating systems
- **Access control**
- User authentication
- Security policies and models
- Trusted operating system design

Control Access to General Objects

- Protecting memory is a specific case of the more general problem of protecting objects:
 - memory
 - a file or data set on an auxiliary storage device
 - an executing program in memory
 - a directory of files
 - a hardware device
 - a data structure, such as a stack
 - a table of the operating system
 - instructions, especially privileged instructions
 - passwords and the user authentication mechanism
 - the protection mechanism itself

Access control

- Memory is only one of many objects for which OS has to run access control
- In general, access control has three goals:
 - **Check every access**: Else OS might fail to notice that access has been revoked
 - **Enforce least privilege**: Grant program access only to **smallest** number of objects required to perform a task
 - **Verify acceptable use**: Limit types of activity that can be performed on an object
 - E.g., for integrity reasons

Access control structures

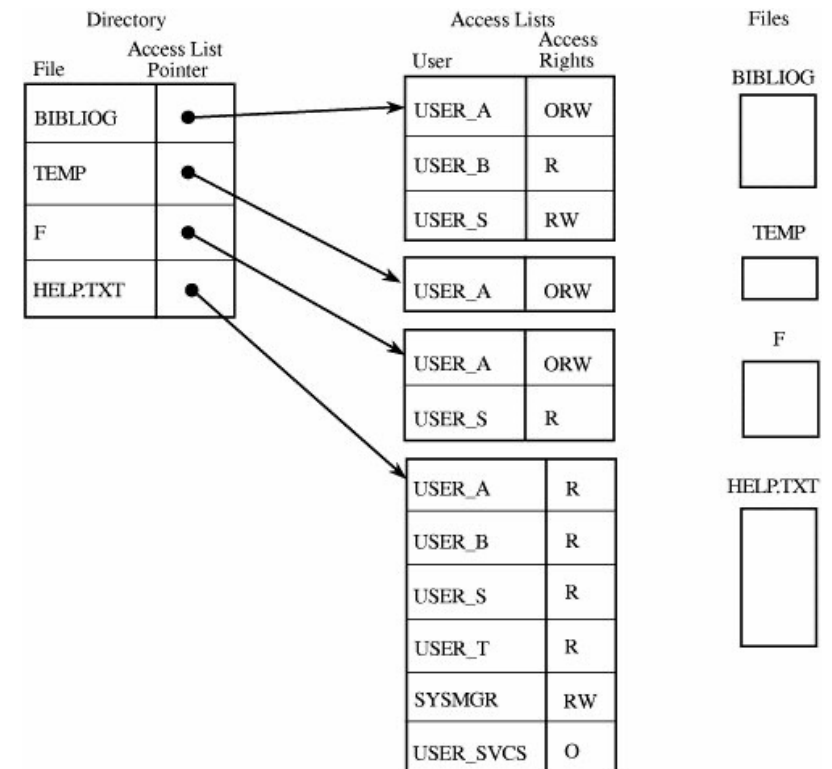
- Access control matrix
- Access control lists
- Capabilities
- Role-based access control

Access Control List

There is **one such list for each object**, and the **list shows all subjects who should have access to the object and what their access is**.

Example Access control list

- Consider subjects A and S, both of whom have access to object F.
- The OS will maintain just one access list for F, showing the access rights for A and S,
- The access control list can include general default entries for any users
 - In this way, specific users can have explicit rights, and all other users can have a default set of rights
 - With this organization, a public file or program can be shared by all possible users of the system without the need for an entry for the object in the individual directory of each user.



Access control matrix

- A table in which each row represents a subject, each column represents an object, and each entry is the set of access rights for that subject to that object.
- The access control matrix is sparse (meaning that most cells are empty):
 - Most subjects do not have access rights to most objects.
 - The access matrix can be represented as a list of triples, having the form <subject, object, rights>.
 - Searching a large number of these triples is inefficient enough that this implementation is seldom used.

Access control matrix

- Set of protected objects: O
 - E.g., files or database records
- Set of subjects: S
 - E.g., humans (users), processes acting on behalf of humans or group of humans/processes
- Set of rights: R
 - E.g., read, write, execute, own
- Access control matrix consists of entries $a[s,o]$, where $s \in S$, $o \in O$ and $a[s,o] \subseteq R$

Example access control matrix

	File 1	File 2	File 3
Alice	orw	rx	o
Bob	r	orx	
Carol		rx	

Implementing access control matrix

- Access control matrix is rarely implemented as a matrix
- Instead, an access control matrix is typically implemented as
 - a set of **access control lists**
 - column-wise representation
 - a set of **capabilities**
 - row-wise representation
 - or a combination

(Refresh) Access control lists (ACLs)

- Each object has a list of subjects and their access rights
 - File 1: Alice:orw, Bob:r, File 2: Alice:rx, Bob:orx, Carol:rx
 - ACLs are implemented in Windows file system (NTFS), user entry can denote entire user group (e.g., “Students”)
 - Classic UNIX file system has simple ACLs. Each file lists its owner, a group and a third entry representing all other users. For each class, there is a separate set of rights.
Groups are system-wide defined in /etc/group, use chmod/chown/chgrp for setting access rights to your files

Capabilities

- A user may be required to have a ticket or pass that enables access, much like a ticket or identification card that cannot be duplicated.
- A **capability** is an un-forgable token that gives the possessor certain rights to an object.
- A subject can create new objects and can specify the operations allowed on those objects.
 - For example, users can create objects, such as files, data segments, or sub-processes, and can also specify the acceptable kinds of operations, such as read, write, and execute. But a user can also create completely new objects, such as new data structures, and can define types of accesses previously unknown to the system.

Capabilities

- The OS holds all tickets on behalf of the users.
- The OS returns to the user a pointer to an OS data structure, which also links to the user.
- A capability can be created only by a specific request from a user to the operating system.
 - Each capability also identifies the allowable accesses

Capabilities

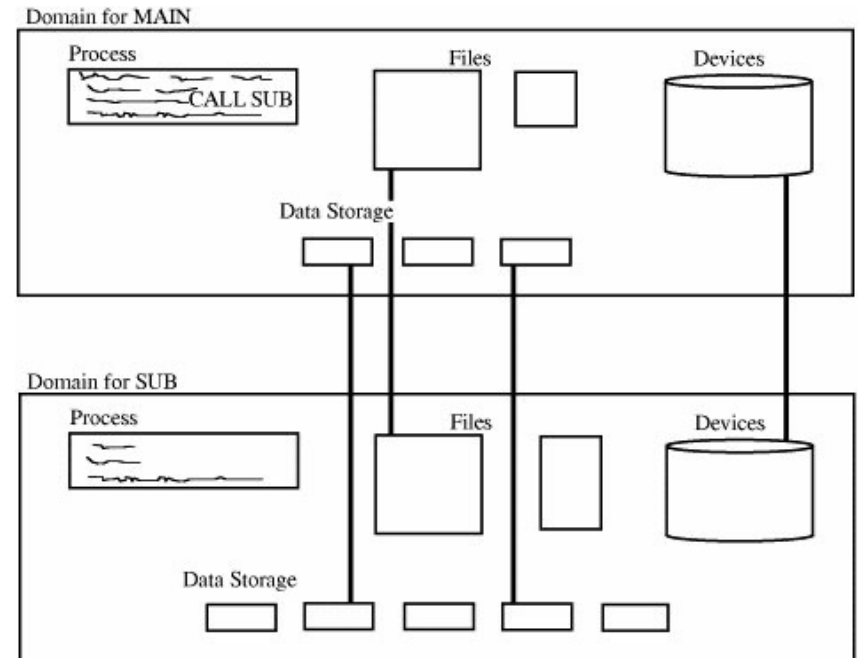
- Capabilities can be encrypted under a key available only to the access control mechanism.
- If the encrypted capability contains the identity of its rightful owner, user A cannot copy the capability and give it to user B.
- One possible access right to an object is transfer or propagate.
 - subject having this right can pass copies of capabilities to other subjects
 - In turn, each of these capabilities also has a list of permitted types of accesses, one of which might also be transfer

Capabilities

- As a process executes, it operates in a domain or local name space.
- The domain is the collection of objects to which the process has access.
- A domain for a user at a given time might include some programs, files, data segments, and I/O devices such as a printer and a terminal.

Capabilities

- Capabilities are a straightforward way to keep track of the access rights of subjects to objects during execution.
- The capabilities are backed up by a more comprehensive table, such as an access control matrix or an access control list.
- Each time a process seeks to use a new object, the operating system examines the master list of objects and subjects to determine whether the object is accessible. If so, the OS creates a capability for that object.



Capabilities

- Capabilities must be stored in memory inaccessible to norm all users.
 - User's segment table or to enclose them in protected memory as from a pair of base/bounds registers
 - Another approach is tagged architecture machine to identify capabilities as structures requiring protection.
- During execution, only the capabilities of objects that have been accessed by the current process are kept readily available.
- Capabilities can be revoked. When an issuing subject revokes a capability, no further access under the revoked capability should be permitted

Capabilities Recap

- A capability is an **unforgeable token** that gives its owner some access rights to an object
 - Alice: File 1:orw, File 2:rx, File 3:o
- Unforgeability enforced by having OS store and maintain tokens or by cryptographic mechanisms
 - E.g., digital signatures allow tokens to be handed out to processes/users. OS will detect tampering when process/user tries to get access with modified token.
- Tokens might be transferable (e.g., if anonymous)
- Some research/experimental OSs (e.g., Hydra, Fuchsia) have fine-grained support for tokens
 - Caller gives callee procedure only minimal set of tokens
- Answer questions from previous slide for capabilities

Combined usage of ACLs and cap.

- In some scenarios, it makes sense to use both ACLs and capabilities
- In a UNIX file system, each file has an ACL, which is consulted when executing an `open()` call
- If approved, caller is given a capability listing type of access allowed in ACL (read or write)
 - Capability is stored in memory space of OS
- Upon `read()/write()` call, OS looks at capability to determine whether type of access is allowed

Role-based access control (RBAC)

- In a company, objects that a user can access often do not depend on the identity of the user, but on the user's job function (role) within the company
 - Salesperson can access customers' credit card numbers, marketing person only customer names
- In RBAC, administrator assigns users to roles and grants access rights to roles
 - Sounds similar to groups, but groups are less flexible
- When a user takes over new role, need to update only her role assignment, not all her access rights
- Available in many commercial databases

RBAC extensions

- RBAC also supports more complex access control scenarios
- **Hierarchical roles**
 - “A manager is also an employee”
 - Reduces number of role/access rights assignments
- Users can have **multiple roles** and assume/give up roles as required by their current task
 - “Alice is a manager for project A and a tester for project B”
 - User’s current session contains currently initiated role
- **Separation of Duty**
 - “A payment order needs to be signed by both a manager and an accounting person, where the two cannot be the same person”

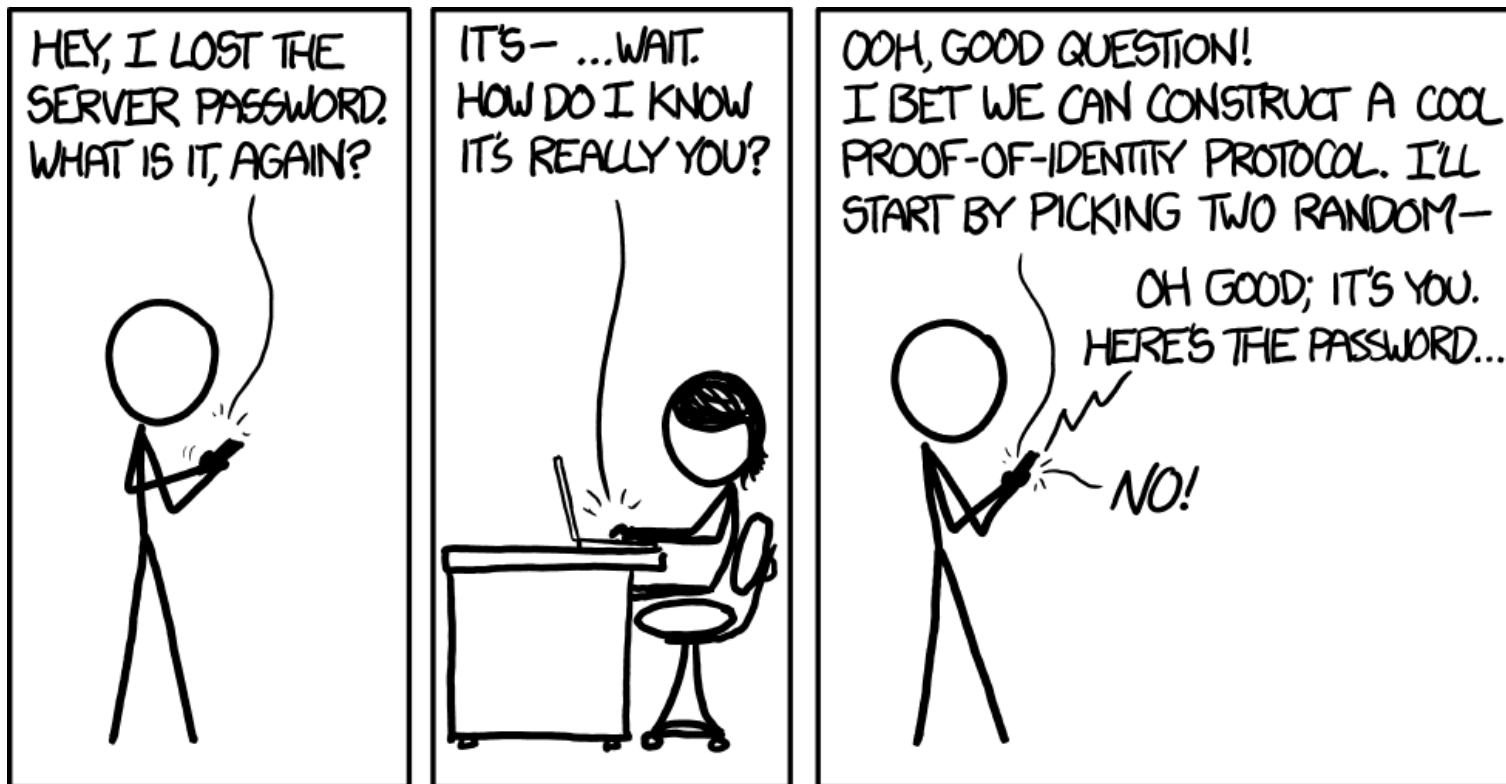
Outline

- Protection in general-purpose operating systems
- Access control
- **User authentication**
- Security policies and models
- Trusted operating system design

User authentication

- Computer systems often have to **identify** and **authenticate** users before **authorizing** them
- Identification: Who are you?
- Authentication: Prove it!
- Identification and authentication is easy among people that know each other
 - For your friends, you do it based on their face or voice
- More difficult for computers to authenticate people sitting in front of them
- Even more difficult for computers to authenticate people accessing them remotely

User authentication <https://xkcd.com/1121/>



Authentication factors

- **Four** classes of authentication factors
- Something the user **knows**
 - Password, PIN, answer to “secret question”
- Something the user **has**
 - ATM card, badge, browser cookie, physical key, uniform, smartphone
- Something the user **is**
 - Biometrics (fingerprint, voice pattern, face, . . .)
 - Have been used by humans forever, but only recently by computers
- Something about the user’s **context**
 - Location, time, devices in proximity

Combination of auth. factors

- **Different classes** of authentication factors can be combined for more solid authentication
 - Two- or multi-factor authentication
- Using multiple factors from the **same** class might not provide better authentication
- “Something you have” can become “something you know”
 - Token can be easily duplicated, e.g., magnetic strip on ATM card
 - Token (“fob”) displays number that changes over time and that needs to be entered for authentication
 - SMS message

Passwords

- Probably oldest authentication mechanism used in computer systems
- User enters user ID and password, maybe multiple attempts in case of error
- Many usability problems, such as
 - Entering passwords is inconvenient, in particular on
 - small screens
 - Password composition/change rules
 - Forgotten passwords might not be recoverable
 - If password is shared among many people, password
 - updates become difficult

Security problems with passwords

- If password is disclosed to unauthorized individual, the individual can immediately access protected resource
 - Unless we use multi-factor authentication
- Shoulder surfing
- Keystroke logging
- Interface illusions / Phishing
- Password re-use across sites
- Password guessing

Next slides provide information covered

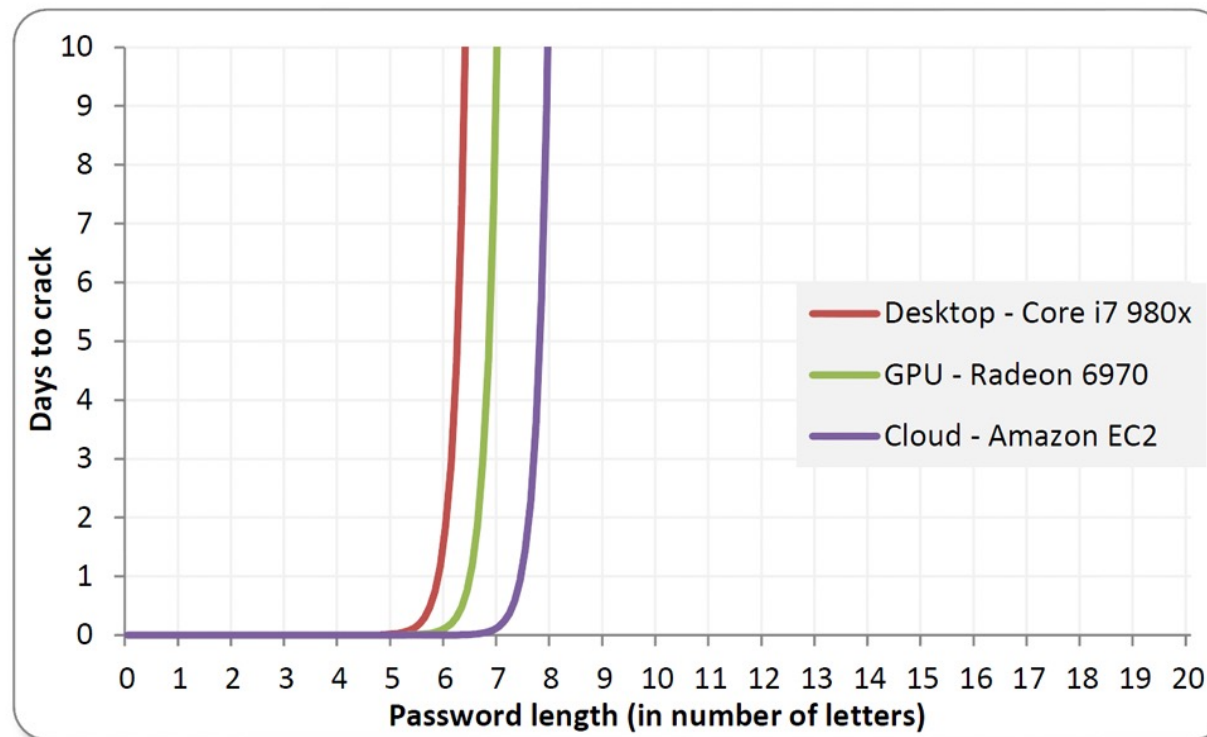
- Password + biometric authentication
- Skip to slide 108

Password guessing attacks

- **Brute-force**: Try all possible passwords using exhaustive search
- Can test 350 billion Windows NTLM passwords per second on a cluster of 25 AMD Radeon graphics cards
- Can try 95^8 combinations in 5.5 hours
- Enough to brute force every possible eight-character password containing upper- and lower-case letters, digits, and symbols

Brute-forcing passwords is exponential

<https://blog.erratasec.com/2012/08/common-misconceptions-of-password.html>



Password guessing attacks

- Exhaustive search assumes that people choose passwords randomly, which is often not the case
- Attacker can do much better by exploiting this
- For example, assume that a password consists of a root and a pre- or postfix appendage
 - “password1”, “abc123”, “123abc”
- Root is from dictionaries (passwords from previous password leaks, names, English words, . . .)
- Appendage is combination of digits, date, single symbol, . . .
- >90% of 6.5 million LinkedIn password hashes leaked in June 2012 were cracked within six days

Password guessing attacks

- So should we just give up on passwords?
- Attack requires that attacker has encrypted password file or encrypted document
 - Offline attack
- Instead, attacker might want to guess your banking password by trying to log in to your bank's website
 - Online attack
- Online guessing attacks are detectable
 - Bank shuts down online access to your bank account after n failed login attempts (typically $n \leq 5$)
 - But! How can an attacker circumvent this lockout?

Password hygiene

- Use a password manager to create and store passwords
 - At least for low- and medium-security passwords
 - All (most) eggs are now in one basket, so keep your computer's software up to date
 - Prevents password re-use across sites
- Use a pass phrase
 - Phrase of randomly chosen words, avoid common phrases (e.g., advertisement slogans)

Password hygiene

- Have site-specific passwords
- Don't reveal passwords to others
 - In email or over phone
 - If your bank really wants your password over the phone, switch banks
 - Studies have shown that people disclose passwords for a cup of coffee, chocolate, or nothing at all
 - Caveat of these studies?
- Don't enter password that gives access to sensitive information on a public computer (e.g., Internet cafe)
 - Don't do online banking on them
 - While travelling, forward your email to a free Webmail provider and use throwaway (maybe weak) password

Advice for developers (NIST 2017)

- No password composition rules
 - Otherwise everybody uses the same simple tricks to follow rule
- At least 8 characters minimum length
- At least 64 characters maximum length
- Allow any characters, including space, Unicode, and emoji
- Black list frequently used or compromised passwords (from password leaks)
- Avoid password hints or “secret questions”

Advice for developers (NIST 2017)

- Don't ask users to periodically change passwords
 - Leads to password cycling and similar
 - “myFavoritePwd” -> “dummy” -> “myFavoritePwd”
 - goodPwd.”1” -> goodPwd.”2” -> goodPwd.”3”
- Allow passwords to be copy-pasted into password fields
- Use two-factor authentication (but avoid SMS-based second factor)

Attacks on password files

- Website/computer needs to store information about a password in order to validate entered password
- Storing passwords in plaintext is dangerous, even when file is read protected from regular users
 - Password file might end up on backup tapes
 - Intruder into OS might get access to password file
 - System administrator has access to file and might use passwords to impersonate users at other sites
 - Many people re-use passwords across multiple sites

Storing password fingerprints

- Store only a **digital fingerprint** of the password (using a cryptographic hash, see later) in the password file
- When logging in, system computes fingerprint of entered password and compares it with user's stored fingerprint
- Still allows offline guessing attacks when password file leaks

Defending against guessing attacks

- UNIX makes guessing attacks harder by including **user-specific salt** in the password fingerprint
 - Salt is initially derived from time of day and process ID of /bin/passwd
 - Salt is then stored in the password file in plaintext
- Two users who happen to have the same password will likely have different fingerprints
- Makes guessing attacks harder, can't just build a single table of fingerprints and passwords and use it for any password file

Defending against guessing attacks

- Don't use a standard cryptographic hash (like SHA-1 or SHA-512) to compute the stored fingerprint
- They are relatively cheap to compute (microseconds)
- Instead use an iterated hash function that is expensive to compute (e.g., bcrypt) and maybe also uses lots of memory (e.g., scrypt)
 - Hundreds of milliseconds
- This slows down a guessing attack significantly, but is barely noticed when a users enters his/her password

Defending against guessing attacks

- An additional defense is to use a MAC (see later), instead of a cryptographic hash
- A MAC mixes in a secret key to compute the password fingerprint
- If the fingerprints leak, guessing attacks aren't useful anymore
- Can protect the secret key by embedding it in tamper resistant hardware
- If the key does leak, the scheme remains as secure as a scheme based on a cryptographic hash

Password Recovery

- A password cannot normally be recovered from a hash value (fingerprint)
- If password recovery is desired, it is necessary to store an **encrypted version** of the password in the password file
- We need to keep encryption key away from attacker

Password Recovery

- As opposed to fingerprints, this approach allows the system to (easily) re-compute a password if necessary
 - E.g., have system email password **in the clear** to predefined email address when user forgets password
 - This has become the norm for many websites
 - In fact, some people use this reminder mechanism whenever they want to log in to a website
- There are many problems with this approach!

The Adobe Password Hack (November 2013)

- In November 2013, 130 million **encrypted** passwords for Adobe accounts were revealed.
- The encryption mechanism was the following:
 - First a NUL byte was appended to the password.
 - Next, additional NUL bytes were appended as required to make the length a multiple of 8 bytes.
 - Then the padded passwords were encrypted 8 characters at a time using a fixed key. (This is called **ECB mode** and it is the **weakest possible** encryption mode.)
- The password hints were not encrypted.
- It turns out that many passwords can be decrypted, without breaking the encryption and not knowing the key.

Interception attacks

- Attacker intercepts password while it is in transmission from client to server
- One-time passwords make intercepted password useless for later logins
 - Fobs (see earlier)
 - Challenge-response protocols

Challenge-response protocols

- Server sends a random challenge to a client
- Client uses challenge and password to compute a one-time password
- Client sends one-time password to server
- Server checks whether client's response is valid
- Given intercepted challenge and response, attacker might be able to brute-force password

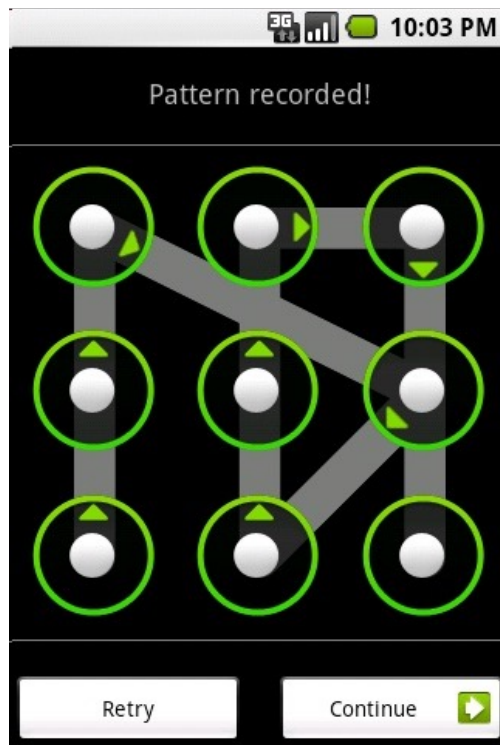
Interception attacks

- There are cryptographic protocols (e.g., SRP) that make intercepted information useless to an attacker
- On the web, passwords are transmitted mostly in plaintext
 - Sometimes, digital fingerprint of them
 - Encryption (TLS, see later) protects against interception attacks on the network
- Alternative solutions are difficult to deploy
 - Patent issues, changes to HTTP protocol, hardware
- And don't help against interception on the client side
 - Malware

Graphical passwords

- Graphical passwords are an alternative to text-based passwords
- Multiple techniques, e.g.,
 - User chooses a picture; to log in, user has to re-identify this picture in a set of pictures
 - User chooses set of places in a picture; to log in, user has to click on each place
- Issues similar to text-based passwords arise
 - E.g., choice of places is not necessarily random
- Shoulder surfing becomes a problem
- Ongoing research

Android unlock patterns



Server authentication

- With the help of a password, system authenticates user (client)
- But **user should also authenticate system (server)** else password might end up with attacker!
- Classic attack:
 - Program displays fake login screen
 - When user “logs in”, program prints error message,
 - sends captured user ID/password to attacker, and ends
 - current session (which results in real login screen)
 - That’s why Windows trains you to press
 - <CTRL-ALT-DELETE> for login, key combination cannot be overridden by attacker
- Today’s attack:
 - **Phishing**

Biometrics

- Biometrics have been hailed as a way to get rid of the problems with password and token-based authentication
- Unfortunately, they have their own problems
- Idea: Authenticate user based on **physical characteristics**
 - Fingerprints, iris scan, voice, handwriting, typing pattern, . . .
- If observed trait is **sufficiently close** to previously stored trait, accept user
 - Observed fingerprint will never be completely identical to a previously stored fingerprint of the same user

Local vs. remote authentication

- Biometrics work well for local authentication, but are less suited for remote authentication or for identification
- In local authentication, a guard can ensure that:
 - I put my own finger on a fingerprint scanner, not one made out of gelatin
 - I stand in front of a camera and don't just hold up a picture of somebody else
- In remote authentication, this is much more difficult

Authentication vs. identification

- Authentication: Does a captured trait correspond to a particular stored trait?
- Identification: Does a captured trait correspond to any of the stored traits?
 - Identification is an (expensive) **search problem**, which is made worse by the fact that in biometrics, matches are based on closeness, not on equality (as for passwords)
- **False positives** can make biometrics-based identification useless
 - False positive: Alice is accepted as Bob
 - False negative: Alice is incorrectly rejected as Alice

Biometrics-based identification

- Example (from Bruce Schneier's "Beyond Fear"):
- Face-recognition software with (unrealistic) accuracy of 99.9% is used in a football stadium to detect terrorists
 - 1-in-1,000 chance that a terrorist is not detected
 - 1-in-1,000 chance that innocent person is flagged as terrorist
- If one in 10 million stadium attendees is a **known** terrorist, there will be 10,000 false alarms for every real terrorist
- Remember "The Boy Who Cried Wolf"?
- Another example of the base rate fallacy

Other problems with biometrics

- **Privacy**

- Why should my employer (or a website) have information about my fingerprints, iris,...?
 - Aside: Why should a website know my date of birth, my mother's maiden name, . . . for "secret questions"?
- What if this information leaks? Getting a new password is easy, but much more difficult for biometrics

- **Accuracy**: False negatives are annoying

- What if there is no other way to authenticate?
- What if I grow a beard, hurt my finger,...?

Other problems with biometrics

- **Secrecy**: Some of your biometrics are not particularly secret
 - Face, fingerprints,...
- **Legal protection**: The law may allow the police to put your finger on your phone's fingerprint reader (or simply hold your phone's camera in front of you). But the law may protect you from you having to reveal your password (depending on the country).

Outline

- Protection in general-purpose operating systems
- Access control
- User authentication
- **Security policies and models**
- Trusted operating system design

Trusted operating systems

- Trusting an entity means that if this entity misbehaves, the security of the system fails
- We trust an OS if we have **confidence** that it provides security services, i.e.,
 - Memory and file protection
 - Access control and user authentication

Trusted operating systems

- Typically a trusted operating system builds on four factors:
 - **Policy**: A set of rules outlining what is secured and why
 - **Model**: A model that implements the policy and that can be used for reasoning about the policy
 - **Design**: A specification of how the OS implements the model
 - **Trust**: Assurance that the OS is implemented according to design
- Let's go a bit deeper in some of those factors (some info already discussed)

Trusted software

Software that has been rigorously developed and analyzed, giving us reason to trust that the code does what it is expected to do and nothing more

- **Functional correctness**
 - Software works correctly
- **Enforcement of integrity**
 - Wrong inputs don't impact correctness of data
- **Limited privilege**
 - Access rights are minimized and not passed to others
- **Appropriate confidence level**
 - Software has been rated as required by environment
- **Trust can change over time, e.g., based on experience**

Security policies

- Many OS security policies have their roots in military security policies
 - That's where lots of research funding came from
- Each object/subject has a sensitivity/clearance level
 - “Top Secret” > “Secret” > “Confidential” > “Unclassified”
where “>” means “more sensitive”
- Each object/subject might also be assigned to one or more compartments
 - E.g., “Soviet Union”, “East Germany”
 - **Need-to-know rule**
- Subject s can access object o iff $\text{level}(s) \geq \text{level}(o)$ and $\text{compartments}(s) \supseteq \text{compartments}(o)$
 - **s dominates o** , short “ $s \geq_{\text{dom}} o$ ”

Example

- Secret agent James Bond has clearance “Top Secret” and is assigned to compartment “East Germany”
- Can he read a document with sensitivity level “Secret” and compartments “East Germany” and “Soviet Union”?
- Which documents can he read?

Commercial security policies

- Rooted in military security policies
- Different classification levels for information
 - E.g., external vs. internal
- Different departments/projects can call for need-to-know restrictions
- Assignment of people to clearance levels typically not as formally defined as in military
 - Maybe on a temporary/ad hoc basis

Other security policies

- So far we've looked only at confidentiality policies
- Integrity of information can be as or even more important than its confidentiality
 - Based on **well-formed transactions** that transition system from a consistent state to another one
 - Also supports Separation of Duty (see RBAC slides)
- Another issue is dealing with **conflicts of interests**
 - Chinese Wall Security Policy
 - Once you've decided for a side of the wall, there is no easy way to get to the other side

Security Policies Review

- Next slides cover info already covered, skip to slide #126

Chinese Wall security policy

- Once you have been able to access information about a particular kind of company, you will no longer be able to access information about other companies of the same kind
 - Useful for consulting, legal or accounting firms
 - Need history of accessed objects
 - Access rights change over time
- **ss-property**: Subject s can access object o iff each object previously accessed by s either belongs to the same company as o or belongs to a different kind of company than o does
- ***-property**: For a write access to o by s , we also need to ensure that all objects readable by s either belong to the same company as o or have been sanitized

Example

- Fast Food Companies = {McDonalds, Wendy's}
- Book Stores = {Chapters, Amazon}
- Alice has accessed information about McDonalds
- Bob has accessed information about Wendy's
- ss-property prevents Alice from accessing information about Wendy's, but not about Chapters or Amazon
 - Similar for Bob
- Suppose Alice could write information about McDonalds to Chapters and Bob could read this information from Chapters
 - Indirect information flow violates Chinese Wall Policy
 - *-property forbids this kind of write

Security models

- Many security models have been defined and interesting properties about them have been proved
- Unfortunately, for many models, their relevance to practically used security policies is not clear
- Two prominent models
 - Bell-La Padula Confidentiality Model (BLP)
 - Biba Integrity Model
- Targeted at Multilevel Security (MLS) policies, where subjects/objects have clearance/classification levels

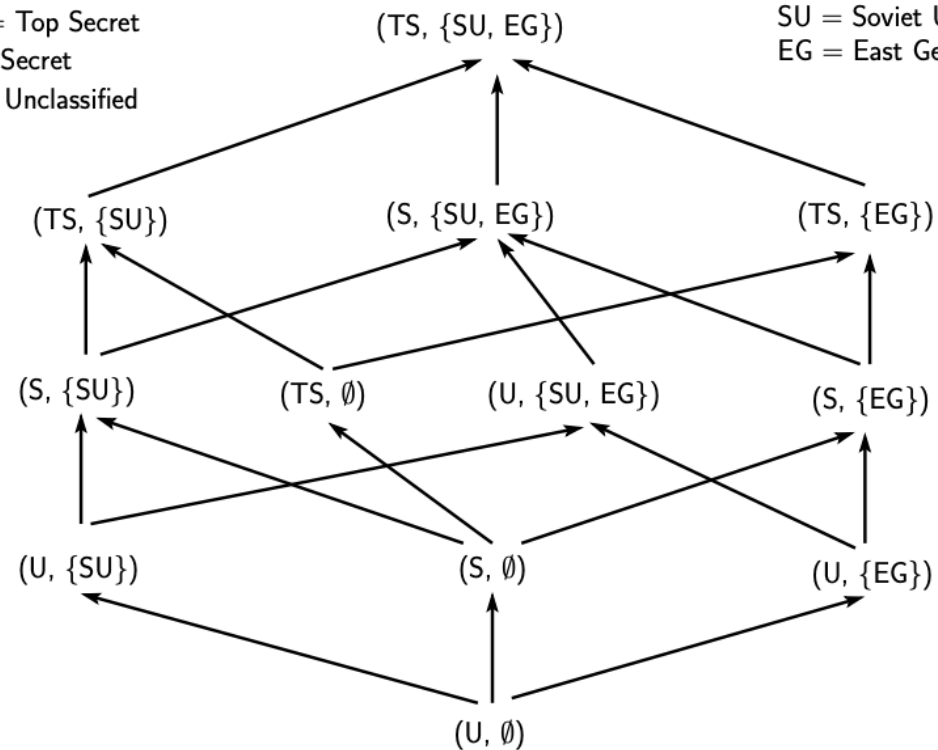
Lattices

- Dominance relationship \geq_{dom} defined in military security model is transitive and antisymmetric
- Therefore, it defines a **partial** order (neither $a \geq_{\text{dom}} b$ nor $b \geq_{\text{dom}} a$ might hold for two levels a and b)
- In a **lattice**, for every a and b , there is a **unique lowest upper bound** u for which $u \geq_{\text{dom}} a$ and $u \geq_{\text{dom}} b$ and a **unique greatest lower bound** l for which $a \geq_{\text{dom}} l$ and $b \geq_{\text{dom}} l$
- There are also two elements U and L that dominate/are dominated by all levels
 - $U = (\text{"Top Secret"}, \{\text{"Soviet Union"}, \text{"East Germany"}\})$
 - $L = (\text{"Unclassified"}, \emptyset)$

Example lattice

Sensitivity levels:
TS = Top Secret
S = Secret
U = Unclassified

Compartments:
SU = Soviet Union
EG = East Germany



Bell-La Padula confidentiality model

- Regulates **information flow** in MLS policies, e.g., lattice-based ones
- Users should get information only according to their clearance
- Should subject s with clearance $C(s)$ have access to object o with sensitivity $C(o)$?
- Underlying principle: **Information can only flow up**
- ss-property (“**no read up**”): s should have read access to o only if $C(s) \geq_{\text{dom}} C(o)$
- *-property (“**no write down**”): s should have write access to o only if $C(o) \geq_{\text{dom}} C(s)$

Example

- No read up is straightforward
- No write down avoids the following leak:
 - James Bond reads secret document and summarizes it in a confidential document
 - Miss Money Penny with clearance “confidential” now gets access to secret information
- In practice, subjects are programs (acting on behalf of users)
 - Else James Bond couldn’t even talk to Miss Money Penny
 - If program accesses secret information, OS ensures that it can’t write to confidential file later
 - Even if program does not leak information
 - Might need explicit declassification operation for usability purposes

Biba integrity model

- Prevent inappropriate **modification** of data
- Dual of Bell-La Padula model
- Subjects and objects are ordered by an integrity classification scheme, $I(s)$ and $I(o)$
- Should subject s have access to object o ?
- Write access: s can modify o only if $I(s) \geq_{\text{dom}} I(o)$
 - Unreliable person cannot modify file containing high integrity information
- Read access: s can read o only if $I(o) \geq_{\text{dom}} I(s)$
 - Unreliable information cannot “contaminate” subject

Low Watermark Property

- Biba's access rules are very restrictive, a subject cannot ever read lower integrity object
- Can use dynamic integrity levels instead
 - **Subject Low Watermark Property:**
If subject s reads object o , then $I(s) = \text{glb}(I(s), I(o))$,
where $\text{glb}()$ = greatest lower bound
 - **Object Low Watermark Property:**
If subject s modifies object o , then $I(o) = \text{glb}(I(s), I(o))$
- Integrity of subject/object can only go down, information flows **down**

Review of Bell-La Padula & Biba

- Very simple, which makes it possible to prove properties about them
 - E.g., can prove that if a system starts in a secure state, the system will remain in a secure state
- Probably too simple for great practical benefit
 - Need declassification
 - Need both confidentiality and integrity, not just one
 - What about object creation?
- Information leaks might still be possible through covert channels in an implementation of the model

Information flow control

- An information flow policy describes authorized paths along which information can flow
- For example, Bell-La Padula describes a lattice-based information flow policy
- In compiler-based information flow control, a compiler checks whether the information flow in a program could violate an information flow policy
- How does information flow from a variable x to a variable y ?
- Explicit flow: E.g., $y := x$; or $y := x / z$;
- Implicit flow: If $x = 1$ then $y := 0$; else $y := 1$

Information flow control (cont.)

- Input parameters of a program have a (lattice-based) security classification associated with them
- Compiler then goes through the program and updates the security classification of each variable depending on the individual statements that update the variable (using dynamic BLP/Biba)
- Ultimately, a security classification for each variable that is output by the program is computed
- User (more likely, another program) is allowed to see this output only if allowed by the user's (program's) security classification

Outline

- Protection in general-purpose operating systems
- Access control
- User authentication
- Security policies and models
- **Trusted operating system design**