We skipped a bunch of stuff cause its review of cs350.

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  - Check every access: Else OS might fail to notice that access has been revoked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 (ADTs)

Check every access We want the OS to check everytime you try to access or it might notice that access had been revoked from you. Cause you're a dick.

Enforcce least privilige dont give people rights they dont need

Verify acceptable use limit the kinds of things people can do on objects

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	

Most access control systems are visualized as a matrix (not implemented as one). Objects are things you want access to, Subjects are things trying to get access, and Rights are the things the subjects want to do to the object.

Implementing access control matrix

- Access control matrix is rarely implemented as a matrix
  - Why?
- 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

Iterating over the huge access control matrix is super inefficient because it tends to be a very sparse table. Instead we tend to work by columns or rows.

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

- Which of the following can we do quickly for ACLs?
  - Determine set of allowed users per object
  - Determine set of objects that a user can access
  - Revoke a user's access right to an object or all objects

ACLs are what the UNIX system uses (chmod that shit). These are column wise interpretation of an access matrix.

- o = owner
- g = group
- w = world

Looking at these ACLs:

- we can easily find who is allowed to access an object: its very quick because this data is stored on the object
- we cannot easily find all objects a user can access: we'd have to go through all objects and check if the user can access it
- we can easily revoke access to a single object but stupid for all because you have to go through all of them

Capabilities

- 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 (see later) 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 OSs (e.g., Hydra) have fine-grained support for tokens
  - Caller gives callee procedure only minimal set of tokens
- Answer questions from previous slide for capabilities

Capabilities are when we assign rights on a user basis (a row wise look at access matrix). Security for this is very hard because we have to have a list of rights stored somewhere. You have to maintain that when a user access it they don't fuck it up and have permission to look at it. This list can be stored on the OS or on the user, we prefer storing it on the OS. If we give a digital signature to a user (letting them store their own rights) you have to ask them for their capabilities whenever they want access. If a user is not contactable then we cannot know anything about them. It also makes it very hard to revoke their access because they can just refuse to give over their token.

Combined usage of ACLs and cap.

- In some scenarios, it makes sense to use both ACLs and capabilities
  - Why?
- 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
- Problem with this approach?

Most systems use a combination of the two primarily for performance reasons. In linux rights are originally stored as acls but when you open a file they get converted into capabilities (for example fopen returns a file descriptor that tells you its rights).

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

We like to clump users into groups called roles. Everyone in a role has the same access. This makes it very easy to update someones rights by just moving them to a new group. Most comercial databases use this model.

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"

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

AUTHENTICATE EVERYTHING, yep thats pretty much it.

Identification is asking who someone is. **Authentication** is checking that you actually are who you say you are (make sure you have the rights you claim).

Authentication factors

- ThreeFour 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

We can use what a user knows (some data they would know), what they have (an object they have like cookie or atm card), something about what the user is (like biometrics), and something about the user's context (like are they at home now).

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

Good security usually looks to have multiple factors of authentication usually from different classes. With physical items we need to watch for them just becoming data.

Passwords

- Probably oldest authentication mechanism used in computer systems
- User enters user ID and password, maybe multiple attempts in case of error
- Usability problems
  - Forgotten passwords might not be recoverable (though this has been changing recently, see later)
  - Entering passwords is inconvenient
  - If password is disclosed to unauthorized individual, the individual can immediately access protected resource
    - Unless we use multi-factor authentication
  - If password is shared among many people, password updates become difficult

Attacks on Passwords

- Shoulder surfing
- Keystroke logging
- Interface illusions / Phishing
- Password re-use across sites
- Password guessing

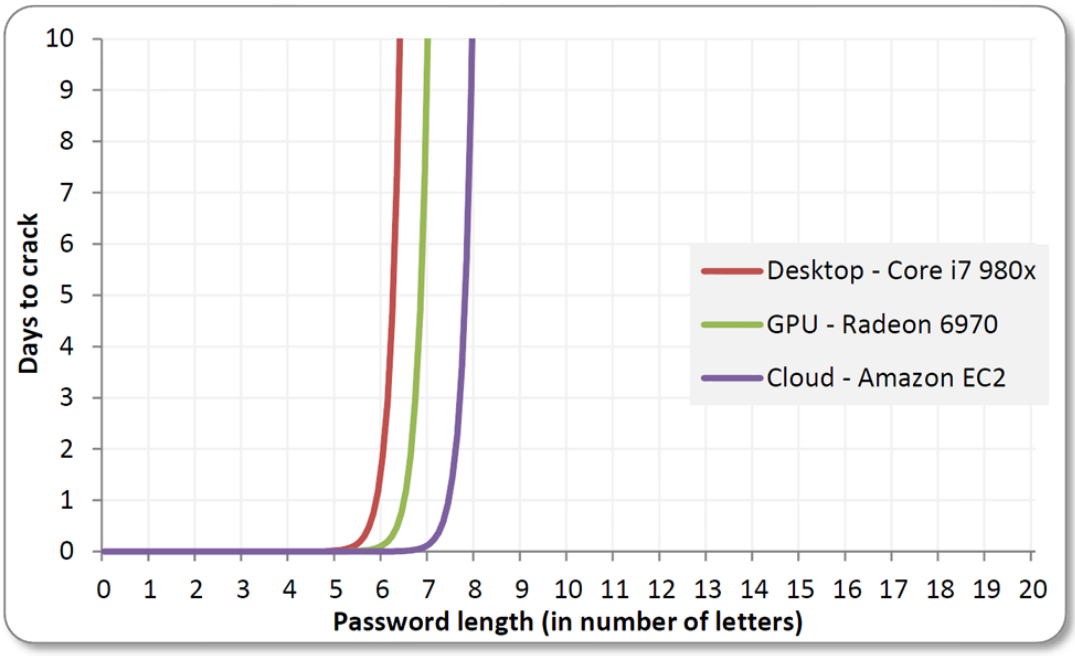
USE A PASSWORD MANAGER. That is all.

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 (see optional reading)
- 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

http://erratasec.blogspot.ca/2012/08/common-misconceptions-of-password.html



We want very long passwords to make brute force attacks much harder.

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

People use patterns and other methods to remember their passwords which can be leveraged to guess them. Software that does this often looks at common password leaks to spot patterns. You want to be able to tell when there is a breach. The most important thing is the throttle login attempts (don't let the same user try to login a ton of times). The iCloud login attack (that got celebrity pics) happened because an api call was not throttled.

Choosing good passwords

- Use letters, numbers and special characters
- Choose long passwords
  - At least eight characters
- Avoid guessable roots
- If supported, use pass phrase
  - Mix upper and lower case, introduce misspellings and special characters
  - Avoid common phrases (e.g., advertisement slogans)

Password hygiene

- Writing down passwords is more secure than storing many passwords on a networked computer or re-using same password across multiple sites
  - Unreasonable to expect users to remember long passwords, especially when changed often
  - Requires physical security for password sheet, don't use sticky notes
- Change passwords regularly
  - Especially if shorter than eight characters
  - Should users be forced to change their password?
  - Leads to password cycling and similar
    - "myFavoritePwd" -> "dummy" -> "myFavoritePwd"
    - goodPwd."1" -> goodPwd."2" -> goodPwd."3"

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 café)
  - Don't do online banking on them
  - While travelling, forward your email to a free Webmail provider and use throwaway (maybe weak) password

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

We do not what to store our passwords in plain text. We should hash them on the way in. Basically we want to make is so that you cannot figure out the password based on the username. Currently everyone uses sha2 hashing but we are soon going to switch to sha3. There are someplaces that store precomputed lists of hashes to help you attack faster (often called rainbow tables).

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

We should be salting our hashes. Generate a random string and pad the hash with it. It should be unique for each user, quite long (48-128 bits) and very random. This helps stop people from precomputing them. Frequently hackers will know the salt, but it still stops rainbow table attacks.

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

You could also delay a while when calculating the hash so that brute force attacks become way more expensive. Basically you just iterate over hashing the hash a bunch of times.

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

MAC is message authentication code (often based on hashes making then HMACs). The key difference is that MACs have keys that are required to create a MAC and to verify data. Hashes are just functions.

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!

You can pepper your password by putting in a special bit of hardware called a hardware security module. This is very hard to get into. This won't ever give out a password, it will MAC something for you and just return that. This will prevent attacks even if they have your code base and know how you hash stuff. Even if they were to get the key to your HSM its still as secure as using an iterative hash. Some laptops have TMPs that can work as very specific HSMs (they have the key burned into them at time of manufacturing).

The Adobe Password Hack (November 2013)

- In November 2013, 130 million encrypted passwords for Adobe accounts were revealed.
- The encryption mechanism was the following:
  1. First a NUL byte was appended to the password.
  2. Next, additional NUL bytes were appended as required to make the length a multiple of 8 bytes.
  3. 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.

The Adobe Password Hack (cont.)

Adobe password data		Password hint	
110edf2294fb8bf4	->	numbers 123456	❶ 123456
110edf2294fb8bf4	->	==123456	
110edf2294fb8bf4	->	c'est "123456"	
8fda7e1f0b56593f e2a311ba09ab4707	->	numbers	❷ 12345678
8fda7e1f0b56593f e2a311ba09ab4707	->	1-8	
8fda7e1f0b56593f e2a311ba09ab4707	->	8digit	
2fca9b003de39778 e2a311ba09ab4707	->	the password is password	❸ password
2fca9b003de39778 e2a311ba09ab4707	->	password	
2fca9b003de39778 e2a311ba09ab4707	->	rhymes with assword	
e5d8efed9088db0b	->	q w e r t y	❹ qwerty
e5d8efed9088db0b	->	ytrewq tagurpidi	
e5d8efed9088db0b	->	6 long qwert	
ecba98cca55eabc2	->	sixxone	❺ 111111
ecba98cca55eabc2	->	1*6	
ecba98cca55eabc2	->	sixones	

XKCD dubbed this "the greatest crossword puzzle in the history of the world": `http://xkcd.com/1286/`.

If a password is the same it will encrypt the same. And it encrypts in chunks so you can figure out what the encryption key is looking at it. Its little more secure than a slightly challenging puzzle.

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

You password still has to be sent through the network, this can be a vulnerability in itself. You have to encrypt the link between you and the server or anyone who is watching can just see it as it goes through. Another attack is if the server gets compromised. If they get read access thats ok because everything is hashed, but if they get write access they can catch the plaintext password and do what they want with it. You can defend agains this by having the server send a challenge that is different every time, the client uses that to create a one time password, then the server replys with a response. Anyone listening will not know what is going on there.

There are some cool ideas where you store passwords on your phone (fido and sqrl) where the website shows a QR code which your phone uses as a challenge to log in.

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

Challenge responses are the server giving the client something that lets them create a one time use password so that if someone intercepts it they cannot just straight use your password. If they do get challenge and response they can brute force figure out your 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

Graphical passwords are a version of passwords. This is easier for humans to remember and machines are super shit at understanding them. A big problem with this is that things aren't random so people can figure things out by guessing. People are very predictable so its also much easier to guess their password if its image based. Its also much easier for someone to remember your password if they shoulder surf.

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", programs 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

While its important that the server authenticates the users trying access it, it is also very important for you to validate the server that you are accessing. You want to make sure that the thing you are giving your password to is a trusted thing. TOFU (trust on first use) when you first log on we save the fingerprint of the server you are accessing so that in the future you can validate that this is the same server as last time.

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"?
- After pilot study, German FBI recently concluded that this kind of surveillance is useless
  - Average detection accuracy was 30%

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,...?
- Secrecy: Some of your biometrics are not particularly secret
  - Face, fingerprints,...

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

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 level(s) $\geq$ level(o) and compartments(s) $\supseteq$ compartments(o)
  - s dominates o, short "s $\geq$ o"

Important note: dont use the textbook notation for this, use the notation in this slide

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?

So we have James Bond who's clearance is Top Secret but he is only assigned to the east Germany section. James Bond cannot access documents that include departments that he is not a part of (so not documents about east germany and soviet union) and but he can access documents of any security level since he has the highest clearance level.

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
  - E.g., Clark-Wilson Security Policy
  - 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

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

Chinese wall security policy is that once you are asigned to one side of the wall (in a department for instance) you cannot have access to another side of the wall. An example of this is advertising firms that might work for conflicting client and you don't want any passing of data between them.

SS stands for simple security. Basically is is a policy to keep people from accessing data for multiple companies in the same sphere.

Star property (it was meant to be a placeholder, they forgot to actually come up with a name) governs write access to objects. Basically if you want to write to something you need to make sure that you have readable access only to that company (unless the data has been **sanitized**). Problems can occur if you have read from multiple companies as you cannot write at all.

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
- We'll focus on two prominent models
  - Bell-La Padula Confidentiality Model
  - Biba Integrity Model
- Targeted at Multilevel Security (MLS) policies, where subjects/objects have clearance/classification levels

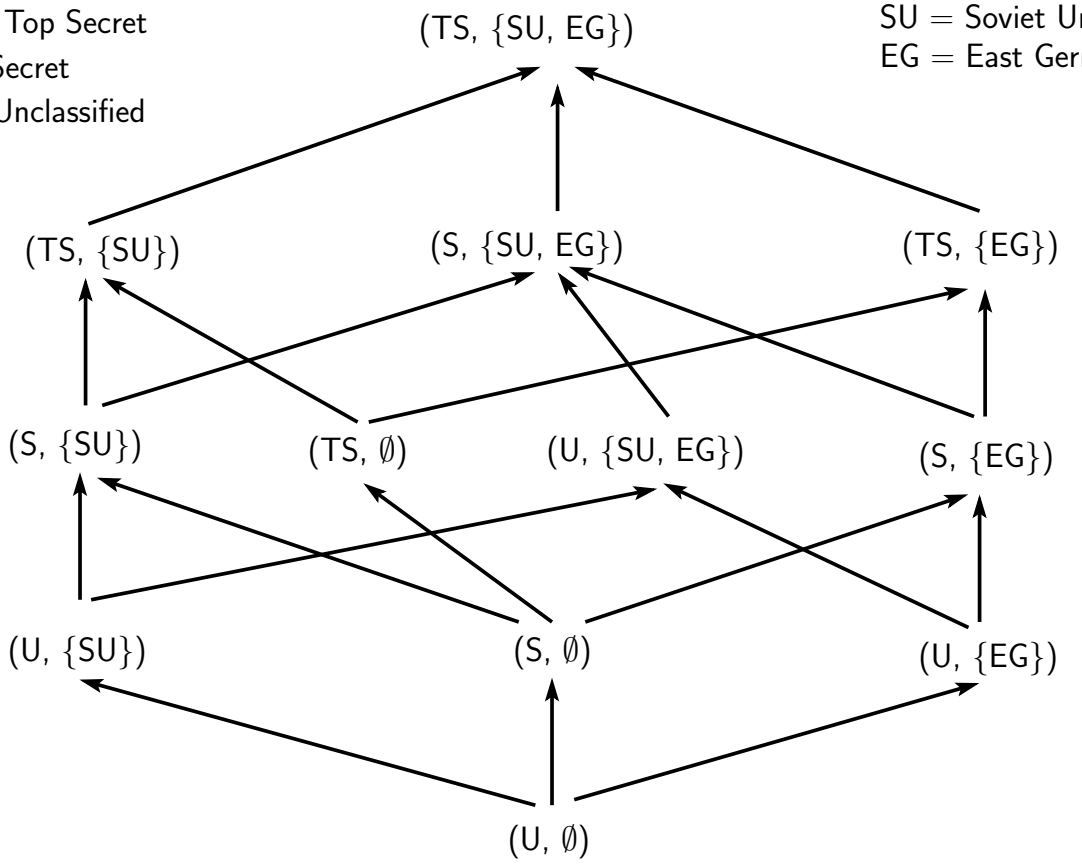
Lattices

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

Example lattice

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

Compartments:
SU = Soviet Union
EG = East Germany



transitive: $a \geq b \wedge b \geq c \Rightarrow a \geq c$

antisymetric: $a \leq b \wedge n \leq a \Rightarrow a = b$

relexive: $\forall a, a \geq a$

partially ordered set is called a poset. A graphical representation of a poset does not contain directed cycles. A **latice** is a poset where there exists a greatest lower bound an a lowest upper bound, and there exists a lowest and a highest element. You can test for this by finding the greatest lower bound and lowest upper bound.

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 $C(s) \geq C(o)$
- *-property ("no write down"): s should have write access to o only if $C(o) \geq C(s)$

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 Moneypenny 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 Moneypenny
  - 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

A subject can only read an object if their clearance is higher that the clearance of the object. A subject can then only write to an object if their clearance is lower than it (since we dont want people to share data that is secure to lower levels).

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 $I(s) \geq I(o)$
  - Unreliable person cannot modify file containing high integrity information
- Read access: s can read o only if $I(o) \geq I(s)$
  - Unreliable information cannot "contaminate" subject

This is a pair with BLP where it maintains integrity of the data. A subject can only write to an object if their integrity is higher than the objects. Like LBP you can then only read objects of higher integrity than you.

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) = glb(I(s), I(o)),
    where glb() = greatest lower bound
  - Object Low Watermark Property:
    If subject s modifies object o, then I(o) = glb(I(s), I(o))
- Integrity of subject/object can only go down, information flows down

A common modification to biba is to leverage the dynamic property of integrity levels. We decide a subject low watermark calculated by setting a subjects integrity to the greatest lower bound of their itegrity and the objects integrity when the read it (a high integrity person reading a low integrity object looses some integrity). An object low watermark changes an objects integrity when a subject writes to it to be the greatest lower bound of their integrity and the integrity of the subject writing to it.

A problem rises with this as over time everyone's integrity lowers until no one has any access.

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

MIDTERM GOES UP TO HERE