

1 Outline

- Allows different entities to access different resources in a shared manner.
- The OS needs to control this sharing and provide an interface for allowing this access.
- Identification and authentication are required for access control.
- Focus on memory protection for now.

2 Protection in General Purpose Operating Systems

2.1 History

- Worth looking at how OSES have evolved until now.
- They now allow multiple users to use the same hardware, and added sequential abilities to run multiple *executives*.
- The OS should make resources available to users if required and permitted to by some policy.
- OSES also protect users from each other; attacks, mistakes, malware, and resource overconsumption are things to protect against, even if it is a single-user OS.

2.2 Protection Basics

- Some protected objects:
 - Memory
 - Data
 - CPU
 - Programs
 - I/O devices
 - Networks
 - The OS itself

2.3 Memory Protection

- We talk about memory protection first — a program should not be able to read the memory of another program unless permitted.
- This is important as a *lot* of things are stored in memory!
- One way to invoke security is separation — keep a user's objects separate from another's!
- We can do this:
 - Physically (easy but expensive and inefficient)
 - Temporally (execute different users' programs at different times)
 - Logically (users don't see other users)

- Cryptographically (make users unable to see other user's data unencrypted)
- But sometimes, users do want to share resources.
- For example, library routines or files/DB records.
- OSes should allow for flexible sharing — but this creates new questions:
 - Which files to share? What part?
 - Which users?
 - Which users can share objects further?
 - What uses are permitted? R/W/E perms? Maybe only specific bits of information (ie: only aggregate info, not individual entries of a DB).
 - How long?
- Often, for memory protection, the OS can exploit hardware support (virtual memory from CS 350) for a cheap solution.
- Memory addresses used are virtual and the MMU will manage this and translate to physical addresses.
- the OS maintains the mapping tables the MMU will use and deals with raised exceptions from the MMU.
- What kinds of hardware support is used for memory protection?
- The simplest is a fence register — exception if the memory access below addresses is in the fence register.
 - This protects the OS from user programs.
 - This only works for a single-user OS.
- Another technique is base/bound register pairs.
 - This throws an exception if memory access is below/above an address in the base/bound register.
 - Each user program has different values.
 - When a ctx switch occurs, the OS is responsible for moving the current base/bond registers to the currently executing user program.
 - We can extend this to have 2 base-bound pairs, one for code, one for data.
 - This approach is more flexible.
- The tagged architecture takes this to the extreme; each memory word has one or more extra bits that identify access rights to the word.
 - This is very flexible, but it incurs a large overhead and is not portable at all.
- Segmentation is a common approach (remember CS 350?).
 - Each program has multiple address spaces, or segments.
 - We have different segments for code, data, stack, though this can be split up even further.
 - The virtual address contains of two parts — the segment name, and the offset within the segment.
 - The OS will keeps mappings from the segment name to its base physical address in the segment table; one is made for each process.
 - The OS can relocate/resize segments and share them between processes.
 - The segment table can also keep protection attributes.
 - However, because of the resizing/relocation capabilities, this means every access requires a bounds check.

- In paging, we divide the virtual and physical address spaces into pages and frames respectively. Frame size is equal to page size.
 - The virtual address consists of the page number and offset within page.
 - The number of bits for the offset is $\log_2(\text{page size})$.
 - The OS keeps the mapping from page number to its base physical address in the page table.
 - Similarly to segmentation, protection attributes are kept in the page table.
 - Typically used; advantages are:
 - * Each address reference is checked for protection by hardware
 - * Prevent users from accessing unpermitted pages
 - * Less used pages can be moved to disk
 - * Users can share access to a page
 - Disadvantages:
 - * Internal fragmentation still is a problem
 - * Assigning different levels of protection of different classes of data items is not feasible
- The x86 architecture supports both segmentation and paging, though paging is more common.
- Memory protection bits can indicate no access, R/W access, or RO access.
- Processors now also typically include a no execute bit, or NX bit. This forbids execution of instructions stored in a page, which would make the stack/heap non-executable.
- Note that this does not stop buffer overflow attacks; see module 2 (see return oriented programming, where we don't inject new code but change existing code to do what we want).

2.4 Access Control

- Access control has 3 goals:
 1. Check every access
 2. Enforce the least privilege
 3. Verify acceptable use
- To describe the access control matrix, we first define some sets:
 - Set of protected objects: O
 - Set of protected subjects: S
 - Set of rights: R

The access control matrix consists of entries $a[s, o]$, where $s \in S$, $o \in O$, and $a[s, o] \subseteq R$.

- For example, we could make a table like:

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

- Often, access control matrices are not actually implemented as a matrix... the size and time would be awful to deal with.

- It's too sparse!
- Instead, typically they use access control lists, a set of capabilities, or some combination.
- An access control list, or ACL, is such that each object has a list of subjects and their access rights.
- We can quickly determine sets of allowed users per object, or revoking access to one specific object, but determining which objects a user can access or revoking a user's access to all objects is slow.
- For example, digital signatures.
- We can make tokens transferable.
- An example of combining usage of ACLs and capabilities:
 - In Unix, each file has an ACL, which is consulted when executing an `open()` call.
 - If approved, then the caller is given a capability listing type of access allowed in ACL.
 - Upon `read()` or `write()` call, the OS looks at capability to determine whether type of access is allowed.
 - This doesn't completely mediate; it does not check *every* access.
- Another way to do access control is role-based access control, or RBAC.
- Administrators assign users to roles and grant access rights to roles.
- This is similar to groups but groups are less flexible. A group is a set of subjects, while a role is as set of privileges to objects.
- When a user takes over a new role, they need to update only their role assignment, not all their access rights.
- Many commercial DBs do this.
- Also supports more complex access control scenarios:
 - Hierarchical roles — a manager is also an employee; this reduces the number of role/access rights assignments
 - Users can have multiple roles and assume/give up roles as required
 - Separation of duty — maybe a task needs two people of different roles that cannot be the same person

3 User Authentication

- Computers often have to identify and authenticate users before authorizing them.
- Difficult for computers to do for people both locally and remotely.
- Four classes of authentication factors:
 - Something the user knows
 - Something the user has
 - Something the user is (biometrics)
 - Something about the user's context (location, time, devices in proximity, etc.)
- Authentication factors may be combined for multi-factor authentication (ie: 2FA).
- Using multiple factors from the same class may not be a good idea; for example 2 passwords isn't really useful.
- Also, must make sure the factor cannot be changed — for example, a factor the user has can become a factor the user knows, like tokens that can be easily duplicated (mag strips) or SMS messages (ie: SIM-jacking).

3.1 Passwords

- A familiar and old authentication method.
- Enter an ID and password, potentially multiple attempts are allowed.
- Many usability problems:
 - Entering passwords may be inconvenient.
 - Password composition/change rules.
 - Forgotten passwords may not be recoverable; people tend to forget them.
 - If a password is shared then updating passwords is difficult.
- And security problems:
 - If the password is leaked, then an individual can immediately access the protected resource unless something like 2FA is used.
 - Shoulder surfing.
 - Keylogging.
 - Interface illusions/phishing.
 - Password re-use.
 - Password guessing (brute-forcing, for example).
- Some guessing attacks are exhaustive searching, dictionary attacks, etc.
- In online attacks, we can try to stop this by rate-limiting. This can still be defeated by social engineering mechanisms, though. For example, bypassing via secret questions/recovery methods.
- Password hygiene can be helped by things like password managers (though this now keeps all eggs in one basket), or passphrases that are easier to remember but still hard to guess.
- Don't reveal your passwords, don't use passwords on public computers, and don't re-use passwords.
- Advice for developers:
 - Don't use password composition rules.
 - At least 8 characters minimum length.
 - At least 64 characters max length.
 - Allow any characters.
 - Blacklist frequently used/compromised passwords.
 - Avoid password hints or secret questions.
 - Don't force users to change passwords.
 - Allow passwords to be copy-pasted.
 - Use 2FA (but avoid SMS-based).
- Attacks on password files — websites need to store passwords (or something) in a file to validate password inputs.
- Passwords in plaintext is a stupid idea as access to it from an intruder or... anyone really, could immediately screw over a lot of people.
- Cryptographic tools:

- Hash — compute a fixed-length, one-way output. Deterministic.
 - MAC — takes a secret key as an input value; otherwise a hash. Deterministic.
- Now, store a hash of the password in the password file.
- Compare the *hash of the input* with the stored hash (fingerprint).
- But this would still allow offline guessing attacks. . .
- Add a user-specific *salt* to the password fingerprint.
- So two users with the same password have different fingerprints.
- This makes pre-computed tables useless.
- Using an iterated hash function which is slow to compute is better than using a standard hash function that would be fast to compute.
- This would not be noticed when a user inputs a password, but much more noticeable during a brute-force attack.
- Then, to make it better, throw in a MAC instead of a cryptographic hash.
- This adds a secret key required to compute the password fingerprint.
- This makes guessing attacks based on the fingerprints alone useless, but would require some kind of secret key generation/storage, using software/hardware.
- If the key does leak, then the scheme remains as secure as a scheme based on solely a cryptographic hash.
- How do we do password recovery?
- Well, that would require storing the password in some way. . . that's a no-go.
- So, use password-resetting mechanisms instead!
- Interception attacks — intercept the password as it is being sent to the server.
- OTP (one-time passwords) make this attack useless; for example fobs, auth. apps, challenge-response protocols.
- A challenge-response protocol is using a random challenge to the client and the completion of the client + password computes a one-time password.
- Also, passwords (or their hashes) are usually encrypted via TLS to protect from interception attacks.
- Though sometimes, they're still transmitted via PT!
- There are also cryptographic protocols, like SRP, that make intercepted info useless to an attacker.
- Unlock patterns are an alternative.
- May have an issue; fingerprints/smudges may reveal the combination!
- Another graphical password is image selection; this has some issues with randomness and shoulder-surfing.
- Systems should authenticate the user, but the user should also authenticate the server is real!
- Biometrics are hailed as a way to get rid of problems with password and token-based authentication.
- Problems:
 - Observed fingerprint just has to be *close enough*, otherwise it would be too hard to use reliably.

- Could potentially be tricked if not done correctly (ie: face identification).
 - Not secret!
 - Cannot change!
- Authentication vs. identification — former is checking whether a captured trait corresponds to a particular stored trait, while the latter is if it corresponds to *any of the stored traits*.
- False positives can make biometrics-based identification useless (ie: Alice is accepted as Bob).
- Another potential example of the base rate fallacy — if the base rate is low and there are some false positives, it will seem like a huge failure.
- More issues with biometrics: privacy, accuracy.