CS 458 / 658
Computer Security and Privacy

Module 3
Operating System Security

Winter 2010
Operating systems

• An operating system allows different users to access different resources in a shared way

• The operating system needs to control this sharing and provide an interface to allow this access

• Identification and authentication are required for this access control

• We will start with memory protection techniques and then look at access control in more general terms
Module outline

1. Protection in general-purpose operating systems
2. User authentication
3. Security policies and models
4. Trusted operating system design
Module outline

1. Protection in general-purpose operating systems
2. User authentication
3. Security policies and models
4. Trusted operating system design
History

- Operating systems evolved as a way to allow multiple users use the same hardware
  - Sequentially (based on executives)
  - Interleaving (based on monitors)
- OS makes resources available to users if required by them and permitted by some policy
- OS also protects users from each other
  - Attacks, mistakes, resource overconsumption
- Even for a single-user OS, protecting a user from him/herself is a good thing
  - Mistakes, malware
Protected objects

- CPU
- Memory
- I/O devices (disks, printers, keyboards,...)
- Programs
- Data
- Networks
Separation

- Keep one user’s objects separate from other users
  - Physical separation
    - Use different physical resources for different users
    - Easy to implement, but expensive and inefficient
  - Temporal separation
    - Execute different users’ programs at different times
  - Logical separation
    - User is given the impression that no other users exist
    - As done by an operating system
  - Cryptographic separation
    - Encrypt data and make it unintelligible to outsiders
    - Complex
Sharing

• Sometimes, users do want to share resources
  • Library routines (e.g., libc)
  • Files or database records

• OS should allow flexible sharing, not “all or nothing”
  • Which files or records? Which part of a file/record?
  • Which other users?
  • Can other users share objects further?
  • What uses are permitted?
    • Read but not write, view but not print (Feasibility?)
    • Aggregate information only

• For how long?
Memory and address protection

- Prevent 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
  - See CS 350 memory management slides
- 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
Protection techniques

- **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

- 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

- 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:
  - \texttt{<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
Segment table

(Protection attributes are missing)
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

- 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:
  - \(<\text{page } \#, \text{ offset within page}>\)
  - \# 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

Source: CS 350 slides
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

- **Disadvantages:**
  - Internal fragmentation
  - Assigning different levels of protection to different classes of data items not feasible
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

- Recent x86 processors also include NX (No eXecute) bit, forbidding execution of instructions stored in page
  - Enabled in Windows XP SP 2 and some Linux distros
  - E.g., make stack/heap non-executable

- Does this avoid all buffer overflow attacks?
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
    - Access to additional objects might be harmless under normal circumstances, but disastrous in special cases
  - **Verify acceptable use**: Limit types of activity that can be performed on an object
    - E.g., for integrity reasons (ADTs)
Access control matrix

- **Set of protected objects**: $O$
  - E.g., files or database records
- **Set of subjects**: $S$
  - E.g., humans, 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

<table>
<thead>
<tr>
<th></th>
<th>File 1</th>
<th>File 2</th>
<th>File 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>orw</td>
<td>rx</td>
<td>o</td>
</tr>
<tr>
<td>Bob</td>
<td>r</td>
<td>orx</td>
<td></td>
</tr>
<tr>
<td>Carol</td>
<td></td>
<td>rx</td>
<td></td>
</tr>
</tbody>
</table>
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
Access control lists (ACLs)

- Each object has a list of subjects and their access rights
  - 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
Capabilities

- A capability is an unforgeable token that gives its owner some access rights to an object
- 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 transferrable
- 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
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
  • Capability is stored in memory space of OS
• If approved, caller is given a capability listing type of access allowed in ACL (read or write)
  • Upon read()/write() call, OS looks at capability to determine whether type of access is allowed
• Problem with this approach?
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 a 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”
Module outline

1. Protection in general-purpose operating systems
2. User authentication
3. Security policies and models
4. 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
 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
  • 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
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
    - Malware can intercept number, see optional reading
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
Password guessing attacks

- **Brute-force**: Try all possible passwords using exhaustive search
- Can test 350,000 Microsoft Word passwords per second on a 3-GHz Pentium 4
- For passwords of length 8 consisting only of letters, there are about $2 \cdot 10^{11}$ possibilities
- Takes 600,000 seconds or 166 hours to test them
  - Expected wait till success is 83 hours
- Easy to buy more hardware if payoff is worth it
  - Parallelizing search and running it on Graphics Processing Unit can achieve a speedup of 25
- Can make attack harder by including digits and special characters in password
- However,...
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, Password Recovery Toolkit (PRTK) assumes that a password consists of a root and a pre- or postfix appendage
  - “password1”, “abc123”, “123abc”
- Root is taken from dictionaries (names, English words, . . .)
- Appendage is two-digit combination, date, single symbol, . . .
- PRTK could have cracked 55% of 34,000 leaked MySpace passwords in 8 hours
  - Even though passwords turned out to better than passwords from previous studies
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 ≤ 5)
  - But! How can an attacker circumvent this lockout?
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

• **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 traveling, 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
Defending against attacks

- 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 guessing attacks when password file leaks
Defending against 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 attacks

- Store an **encrypted version** of the password in the password file
- Need to keep encryption key away from attacker
- As opposed to fingerprints, this approach allows system to (easily) re-compute password if necessary
  - E.g., have system email password to predefined email address when user forgets password
  - Has become the norm for many websites
  - In fact, some people use this reminder mechanism whenever they want to log in to a website
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 plain
  - 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

http://www.usenix.org/events/sec07/tech/thorpe.html
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
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
    - MythBusters demonstrated how easy it is to fool a fingerprint scanner
  - 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,...
Module outline

1. Protection in general-purpose operating systems
2. User authentication
3. Security policies and models
4. 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
- 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) ≥ level(o) and compartments(s) ⊇ compartments(o)
  - s dominates o, short “s ≥ 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
  • Maybe on a temporary/ad hoc basis
• Assignment of people to clearance levels typically not as formally defined as in military
  •
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, 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
- 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
- We’ll focus on two prominent models
  - Bell-La Padula Confidentiality Model
  - Biba Integrity Model
  - See text for others
- 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
• In a lattice, neither $a \geq b$ nor $b \geq a$ might hold for two levels $a$ and $b$
• However, for every $a$ and $b$, there is a lowest upper bound $u$ for which $u \geq a$ and $u \geq b$ and a 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
  • In example,
    $U = \{ \text{“Top Secret”}, \{ \text{“Soviet Union”}, \text{“East Germany”} \} \}$
    $L = \{ \text{“Unclassified”}, \emptyset \}$
Example lattice

\[
\text{("Top Secret", \{ "Soviet Union", "East Germany" \})}
\]

\[
\text{("Top Secret", \{ "Soviet Union" \})}
\]

\[
\text{("Secret", \{ "Soviet Union", "East Germany" \})}
\]

\[
\text{("Secret", \{ "Soviet Union" \})}
\]

\[
\text{("Secret", \{ "East Germany" \})}
\]

\[
\text{("Unclassified", \{ \})}
\]
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 classification $C(o)$?
- 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 $C(o) \geq C(s)$
  - Comment: Version in textbook is a little bit less strict, but this is how *-property is typically defined
Example

- No read up is straightforward
- No write down avoids the following leak:
  - James Bond reads top-secret document and summarizes it in a confidential document
  - Miss Moneypenny with clearance “confidential” now gets access to top-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 top-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 $I(s) \geq I(o)$
    - 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
Low Watermark Property

- Biba’s access rules are very restrictive, a subject cannot ever view 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
- What kind of property does Bell-La Padula imply?
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
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.)

- See text for other sample statements
- Input and output variables of program each have a (lattice-based) security classification $S()$ associated with them
- For each program statement, compiler verifies whether information flow is secure
  - For example, $x := y + z$ is secure only if $S(x) \geq \text{lub}(S(y), S(z))$, where lub() is lowest upper bound
- Program is secure if each of its statements is secure
Module outline

1. Protection in general-purpose operating systems
2. User authentication
3. Security policies and models
4. Trusted operating system design
Trusted system design elements

- Design must address which objects are accessed how and which subjects have access to what
  - As defined in security policy and model
- Security must be part of design early on
  - Hard to retrofit security, see Windows 95/98
- Eight design principles for security
  - Least privilege
    - Operate using fewest privileges possible
  - Economy of mechanism
    - Protection mechanism should be simple and straightforward
  - Open design
    - Avoid security by obscurity
    - Secret keys or passwords, but not secret algorithms
Security design principles (cont.)

- Complete mediation
  - Every access attempt must be checked
- Permission based / Fail-safe defaults
  - Default should be denial of access
- Separation of privileges
  - Two or more conditions must be met to get access
- Least common mechanism
  - Every shared mechanism could potentially be used as a covert channel
- Ease of use
  - If protection mechanism is difficult to use, nobody will use it or it will be used in the wrong way
Security features of trusted OS

- Identification and authentication
  - See earlier
- Access control
- Object reuse protection
- Complete mediation
- Trusted path
- Accountability and audit
- Intrusion detection
Access control

- **Mandatory access control (MAC)**
  - Central authority establishes who can access what
  - Good for military environments
  - For implementing Chinese Wall, Bell-La Padula, Biba

- **Discretionary access control (DAC)**
  - Owners of an object have (some) control over who can access it
  - You can grant others access to your home directory
  - In UNIX, Windows, ...

- **RBAC is neither MAC nor DAC**
- Possible to use combination of these mechanisms
Object reuse protection

- Alice allocates memory from OS and stores her password in this memory
- After using password, she returns memory to OS
  - By calling free() or simply by exiting procedure if memory is allocated on stack
- Later, Bob happens to be allocated the same piece of memory and he finds Alice’s password in it
- OS should erase returned memory before handing it out to other users
- Defensive programming: Erase sensitive data yourself before returning it to OS
  - How can compiler interfere with your good intentions?
- Similar problem exists for files, registers and storage media
Hidden data

- Hidden data is related to object reuse protection

- You think that you deleted some data, but it is still hidden somewhere
  - Deleting a file will not physically erase file on disk
  - Deleting an email in GMail will not remove email from Google’s backups
  - Deleting text in MS Word might not remove text from document
  - Putting a black box over text in a PDF leaves text in PDF
  - Shadow Copy feature of Windows Vista keeps file snapshots to enable restores
Complete mediation / trusted path

- Complete mediation
  - All accesses must be checked
  - Preventing access to OS memory is of little use if it is possible to access the swap space on disk

- Trusted path
  - Give assurance to user that her keystrokes and mouse clicks are sent to legitimate receiver application
  - Remember the fake login screen?
  - Turns out to be quite difficult for existing desktop environments, both Linux and Windows
    - Don’t run sudo if you have an untrusted application running on your desktop
Accountability and audit

- Keep an audit log of all security-related events
  - Provides accountability if something goes bad
    - Who deleted the sensitive records in the database?
    - How did the intruder get into the system?
- An audit log does not give accountability if attacker can modify the log
- At what granularity should events be logged?
  - For fine-grained logs, we might run into space/efficiency problems or finding actual attack can be difficult
  - For coarse-grained logs, we might miss attack entirely or don’t have enough details about it
Intrusion detection

- There shouldn’t be any intrusions in a trusted OS
- However, writing bug-free software is hard, people make configuration errors, ...
- Audit logs might give us some information about an intrusion
- Ideally, OS detects an intrusion as it occurs
- Typically, by correlating actual behaviour with normal behaviour
- Alarm if behaviour looks abnormal
- See later in Network Security module
Trusted computing base (TCB)

- Part of a trusted OS that is necessary to enforce OS security policy
  - Changing non-TCB part of OS won’t affect OS security, changing its TCB-part will
  - TCB better be complete and correct

- TCB can be implemented either in different parts of the OS or in a separate security kernel

- Separate security kernel makes it easier to validate and maintain security functionality

- Security kernel runs below the OS kernel, which makes it more difficult for an attacker to subvert it
Security kernel

1: Hardware
2: Security Kernel:
   - Access control
   - Authentication functions
3: Operating System:
   - Resource allocation
   - Sharing
   - Hardware interactions
4: User Tasks
Rings

- Some processors support this kind of layering based on “rings”
- If processor is operating in ring $n$, code can access only memory and instructions in rings $\geq n$
- Accesses to rings $< n$ trigger interrupt/exception and inner ring will grant or deny access
- x86 architecture supports four rings, but Linux and Windows use only two of them
  - user and supervisor mode
  - i.e., don’t have security kernel
- Some research OSs (Multics, SCOMP) use more
Reference monitor

- Crucial part of the TCB
- Collection of access controls for devices, files, memory, IPC, . . .
- Not necessarily a single piece of code
- Must be tamperproof, unbypassable and analyzable
- Interacts with other security mechanism, e.g., user authentication
Virtualization

- Virtualization is a way to provide logical separation (isolation)
- Different degrees of virtualization
- Virtual memory
  - Page mapping gives each process the impression of having a separate memory space
- Virtual machines
  - Also virtualize I/O devices, files, printers,...
  - Currently very popular (VMware, Xen, Parallels,...)
  - If Web browser runs in a virtual machine, browser-based attacks are limited to the virtual environment
  - On the other hand, a rootkit could make your OS run in a virtual environment and be very difficult to detect (“Blue Pill”)
Least privilege in popular OSs

- Pretty poor
- Windows pre-NT: any user process can do anything
- Windows pre-Vista: fine-grained access control. However, in practice, many users just ran as administrators, which can do anything
  - Some applications even required it
- Windows Vista
  - Easier for users to temporarily acquire additional access rights (“User Account Control”)
  - Integrity levels, e.g., Internet Explorer is running at lowest integrity level, which prevents it from writing up and overwriting all a user’s files
Least privilege in popular OSs (cont.)

- Traditional UNIX: a root process has access to anything, a user process has full access to user’s data
- SELinux and AppArmor provide Mandatory Access Control (MAC) for Linux, which allows the implementation of least privilege
  - No more root user
  - Support both confidentiality and integrity
  - Difficult to set up
- Other, less invasive approaches for UNIX
  - Chroot, compartmentalization, SUID (see next slides)
- What about the iPhone?
Chroot

- **Sandbox/jail** a command by changing its root directory
  - chroot /new/root command

- Command cannot access files outside of its jail

- Some commands/programs are difficult to run in a jail

- But there are ways to break out of the jail
Compartmentalization

• Split application into parts and apply least privilege to each part

• OpenSSH splits SSH daemon into a privileged monitor and an unprivileged, jailed child
  • Confusingly, this option is called UsePrivilegeSeparation. But this is different from Separation of Privileges (see earlier)

• Child receives (maybe malicious) network data from a client and might get corrupted

• Child needs to contact monitor to get access to protected information (e.g., password file)
  • Small, well-defined interface
  • Makes it much more difficult to also corrupt monitor

• Monitor shuts down child if behaviour is suspicious
setuid/suid bit

• In addition to bits denoting read, write and execute access rights, UNIX ACLs also contain an suid bit

• If suid bit is set for an executable, the executable will execute under the identity of its owner, not under the identity of the caller
  • /usr/bin/passwd belongs to root and has suid bit set
  • If a user calls /usr/bin/passwd, the program will assume the root identity and can thus update the password file

• Make sure to avoid “confused deputy” attack
  • Eve executes /usr/bin/passwd and manages to convince the program that it is Alice who is executing the program. Eve can thus change Alice’s password
Assurance

- How can we convince others to trust our OS?
- Testing
  - Can demonstrate existence of problems, but not their absence
  - Might be infeasible to test all possible inputs
  - Penetration testing: Ask outside experts to break into your OS
- Formal verification
  - Use mathematical logic to prove correctness of OS
  - Has made lots of progress recently
  - Unfortunately, OSs are probably growing faster in size than research advances
Assurance (cont.)

- Validation
  - Traditional software engineering methods
  - Requirements checking, design and code reviews, system testing
Evaluation

- Have trusted entity evaluate OS and certify that OS satisfies some criteria
- Two well-known sets of criteria are the “Orange Book” of the U.S. Department of Defense and the Common Criteria
- Orange Book lists several ratings, ranging from “D” (failed evaluation, no security) to “A1” (requires formal model of protection system and proof of its correctness, formal analysis of covert channels)
  - See text for others
  - Windows NT has C2 rating, but only when it is not networked and with default security settings changed
  - Most UNIXes are roughly C1
Common criteria

- Replace Orange Book, more international effort
- Have **Protection Profiles**, which list security threats and objectives
- Products are rated against these profiles
- Ratings range from EAL 1 (worst) to EAL 7 (best)
- Windows XP has been rated EAL 4+ for the Controlled Access Protection Profile (CAPP), which is derived from Orange Book’s C2
  - Interestingly, the continuous release of security patches for Windows XP does not affect its rating
Recap

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