CS 458 Module 1

1. **What is security?**
   1. 3 properties of security:

* Confidentiality – access to systems/data that is only for authorised parties
* Integrity – ask for the “right” data
* Availability – System/data is there when required
  1. Security require “reliability
  2. Secure system allows
     1. Safeguard of personal data confidentially
     2. Allow authorised access to resources
     3. Give you correct and meaningful result when required

1. What is privacy?
   1. “Informational self-determination”

* user(s) get to control information about themselves
  1. CONTROL IS….
* Who gets to see/use
* What they can use/give it for/to

1. Who are the adversaries
   1. Trying to mess with user
   2. E.g.: Crackers, organised crime, government “cyber warriors”
2. Key terms
   1. Assets

* Things we want to protect – hardware, software, data
  1. Vulnerabilities
* Weakness in system
* May be able to be exploited
* Cause loss/harm
  1. Threats
* Loss/harm that might befall a system
* 4 categories
  + Interception
  + Interruption
  + Modification
  + Fabrication
* Need to state threat model
  + Set methods to defend
  + Whom do want to prevent?
  1. Attack
* Action that exploits a vulnerability to execute a threat
  1. Control
* Remove/reduce vulnerability
* Control by preventing attack and blocking threat

1. Methods of defence
   1. Prevent
   2. Deter
   3. Deflect
   4. Detect
   5. Recover (mitigate the effects)
   6. Cryptography

* Protect data by making it unreadable
* Authenticate users with digital signature
* Authenticate transaction with cryptographic protocol
* Ensure integrity of stored data
  1. Software Controls
* Password and other form of access control
* Separate each OS from each user
* Virus scanners
* Development controls enforce quality measure
* Personal firewalls
  1. Hardware Controls – protection not just hardware, but rather using separate hardware to protect the system
* Fingerprint readers
* Smart tokens
* firewall
  1. Physical Controls – protect hardware, and physical access to console, storage
* Locks
* Guards
* Off-site backups
  1. Policies and procedures – non-technical means to protect
* Rules about changing passwords
* Training in security practices

CS 458 Module 2 Program Security

**Hard to write secure programs**

1. Axiom (Murphy) – Programs have bugs
2. Corollary – Security-relevant programs have security bugs
   * Corollary - a proposition that follows from (and is often appended to) one already proved.
3. As long as human write software, there is still bugs
4. **Flaws, faults and failure**
   * Flaw is a problem with a program
   * Security flaw is a problem that affects security in some way
5. Confidentiality
6. Integrity
7. Availability
   * Comes as
     + Faults – mistakes behind the scene
       - Error in code, data, specification, process, etc.
       - Potential problem
     + Fix fault by
       - User experience a failure, work backwards to uncover the underlying fault
       - Intentionally try to cause failures, then proceed as above
         * **RMB to think like an attacker**
       - fix by making small edits (patches) to the program
         * aka penetrate and patch
         * e.g. Microsoft “Patch Tuesday” example
     + Failures – something goes wrong

* **Problem with patching**
  + Might make things **worst**
    - Pressure to patch a fault is often high, causing a narrow focus on the observed failure
    - May introduce new fault
* **Fuzzing** (other approach to patching)
  + Automated tools that understand common problem in system/code
  + Brute force bad inputs
  + Some are more intelligent
    - Behaviour based
    - Construct state machines and deviate from normal state flow
  + Can be used to identify faults early
* **Unexpected behaviour**
  + Spec list things program must do
    - I.e.: ls command must list the names of the files in directory
  + Most implementers wouldn’t care if it did additional things as well
    - Sorting list of filenames alphabetically before outputting them is fine
  + From security POV, extra behaviours could be bad
    - After displaying filenames

1. **Type of security flaws**

* Ways to divide up security flaws by where they are from
* Some flaws that are intentional
  + - **Malicious** flaws to attack system
      * **I.e.: Juniper (Router Company)**
    - **Non-malicious** (intentional flaws) often features that are meant to be in system and correctly implemented, but nonetheless can cause a failure used by an attacker
  + Most flaws are caused by unintentional program error
    - Heartbleed bug in OpenSSL
      * Missing bounds check in code
      * Attacker request a TLS server hand over a large size of its private memory space.
        + Same memory space where OpenSSL stores server private key material as well as TLS session keys.
    - Apple SSL/TLS Bug – Feb 2014
      * Occur in code that check validity of server signature used in an SSL/TLS connection
      * Exist in OSX 10.9, iOS 6.1 & 7
        + Active attacker could exploit this flaw to get a user to accept a fake key chosen by the attacker.
    - Buffer Overflows
      * Most common exploit
        + ‘gets’ & ‘strcpy’ functions don’t check string they are copying fit in buffer.
        + C will not notice the exception and continues its on way
        + If attacker can write data past the end of an array on the stack,

Can overwrite things like the saved return address.

When function returns -> jump to any address of her choosing.

* + - * + Targets on local machine that run super-user privileges.
        + Kinds of buffer overflow

Classic attack – overflow a bugger on stack to jump to shellcode

Attacks a single byte can be written past end of the buffer. (caused by a common off-by-one error)

Overflow of buffers on heap instead of stack

Jump to other parts of program or libraries.

* + - * + Defence

Use a language with bound checking – catch exceptions

Non-executable stack

Stack at random virtual address

Mainstream OS do this

“Canaries” – detect if stack has been overwritten before return from each function

* + - Integer Overflows
      * Machine integers can represent only a limit set of numbers
      * Program assume integer is always positive
        + Overflow will make signed integer wrap and become negative -> violate assumption
      * Attacker can pass values to program that trigger overflow
    - Format String Vulnerabilities
      * Unfiltered user input is used as format string in
        + Printf(), fprintf(), sprint().
      * Printf(buffer) instead of printf(“%”, buffer)
        + Will parse buffer for %’s and use what is on stack to process found format parameters.
      * Printf(“%s%s%s%s”) -> crash your program
      * Printf(“%x%x%x%x”) -> dumps parts of stack
      * %n will write to an address found on the stack
    - Incomplete mediation
      * Input to programs are specified by untrusted users.
        + E.g.: web based application
      * User mistype data in webform
        + Email: abc123#uwaterloo.ca
      * Web application must ensure user has entered a meaningful request -> MEDIATION
      * Incomplete mediation occurs when application accept incorrect data from user
      * Focus on catching entries that are clearly wrong
      * Security issue
        + Wrong data could result to buffer overflow
        + SQL injection
      * A need to make sure that any user supplied input falls within safe values.
    - Client side mediation
* web sites will validate checks on data entered.
  + Invalid data -> popup will prevent submitting
* Many sites rely on client to keep state for them.
  + Hidden fields in the form which are passed back server when user submit.
* Issues:
  + User turn off JavaScript -> user can send values to server (unmediated)
  + Can modify any client-side state.
* Defence against incomplete mediation
  + Careful checks on the values of all fields
* Make sure client has not modified the data in any way.
* TOCTTOU
  + Time of check to time of use
  + Known as race condition errors
  + Happens when
    - User request system to perform an action
    - System verifies user to allowed to perform
    - System performs the action
  + What you check and what you use might be different
  + Example:
    - Unix terminal program runs super-user privileges that allocate terminals to users
    - Support a command to write contents of terminal to log file
    - If first check user has permission to write -> open for writing
    - Attacker make symbolic link
    - Between “check and “open” attacker change it
  + State of system has been comprised during check for permission and execution of operation
  + Defence:
    - When performing privileged action on behalf of another party, ensure all information relevant to access control is constant
      * Keep private copy of request itself so request can’t be altered.
      * Use locks to ensure object is not changed during the race.

1. **Malicious code**

**Malware**

* Various forms of software written with malicious intent
* Common characteristic -> need to be executed to cause harm
  + Weakest link -> keyboard
* How malware get executed
  + User action
    - Download and run a malicious software
    - View a page containing malicious control
    - Opening an executable email attachment
  + Exploiting existing flaw in system
    - Buffer overflow in
      * network daemons (process)
      * email clients/web browser

**Virus**

* Kind of malware that infects other files
  + traditional -> infect only executable programs
  + nowadays -> more data document formats can contain executable code
* File executed -> virus activates -> infect other files with copies of itself
  + Spread between files / between computers

**Infection**

* Virus modify an existing program/document(host) in a way that executing will transfer control to virus
* For executable program
  + Modify other program and copy itself to beginning of targets’ program code.
* For documents with macros
  + Edit other documents to add itself a macro which starts automatically when opened
* Other than infecting other files, a virus will often try to infect computer itself.
  + When computer is booted
  + Put itself in boot sector of hard disk
  + Add itself to list of programs OS runs at boot time

**Spreading**

* Thru user:
  + send infected files to his friends
  + p2p network
* require some kind of user action in order to spread

**Payload**

* payload of an infected machine will activate and something will happen
  + erase your HD
  + corrupt your spreadsheets
  + install keystroke logger to capture your online banking password

**Spotting Virus**

* When files are added to computer
* Do scan entire state of computer over time to time
* Use signature-based protection or behaviour-based protection

**Signature-Based Protection**

* Keep a list of known virus
  + Each virus – stores some characteristic features aka signature– 100 million
    - Signature can be changed by inserting NOP
    - System use features of virus code itself
      * Infection code
      * Payload code
* To avoid being recognise by virus scanners, virus tend to behave polymorphic
  + Instead of making perfect copies of itself
  + Makes modified copy
  + Done by having most of code encrypted -> due to new random key is chose to encrypt it

**Behaviour-based Protection**

* Improvement from signature based protection (limitation)
  + Only scan for viruses that are in list
  + Several brand-new viruses identified per day
* looks for suspicious patterns of behaviour
  + rather than specific code fragments
  + run suspicious code in sandbox first

**False Negatives and Positives**

* any kind of test/scanner can have 2 types of error
  + false negative – fail to identify threat that is present
  + false positive- claim a threat is present when it’s not
* not reliable to reply on behaviour-based

**Trojan Horse**

* programs which claim to do something they usually do, but hide malicious behaviour
  + i.e: Dancing Pig
* gain control by getting user to run code of attacker’s choice
  + provide some code which user wants to run
* payload can be anything
* do not spread between computers
  + rely on multiple users to execute the “trojaned” software
  + i.e.: p2p software

**Scareware/Ransomware**

* demand ransom to return hostage’s resource(information)

**Logic Bombs**

* malicious code hiding in software waiting for a certain trigger to go off(execute its payload)
* written by insider, meant to be triggered sometime in the future
* payload is usually dire
  + usually trigger when insider is no longer an insider

**Spotting Trojan Horses and Logic Bomb**

* extremely tricky - > user is intentionally running the code
* don’t run code from untrusted sources

**Worms**

* self-contained code that can reproduce with little or no user involvement.
* Often use security flaws in software as path to infection
  + Worm exploit flaw -> infect it -> search for another computer on network to infect
  + May or may not be a payload
* E.g: Morris Worm 1988
  + Exploit buffer overflow
  + Use backdoor
  + dictionary attack on local user’s password
* E.g: Code Red Worm 2001
  + Exploit buffer overflow in Microsoft IIS web server
    - Deface its home page
    - Launch attacks on other web servers
    - Launch DOS on web sites
    - Install a back door and a Trojan horse to try to prevent disinfection
* E.g.: Slammer Worm 2003
  + Infect nearly all vulnerable machines in 15 minutes
  + Exploit buffer overflow in Microsoft’s SQL server
  + Vulnerable machine infected with a single UDP packet
* E.g.: Stuxnet 2010
  + Created by US & Israeli intelligence agencies
  + Target: Iranian uranium enrichment program
  + Target Siemens SCADA system installed on windows
* E.g.: Flame
  + Successor of Stuxnet
  + Most complicated malware found
  + Focus on middle eastern countries energy sectors.

1. **Other Malicious Code**

**Web Bugs**

* Object embedded in a web page
* Fetch from different server from the one that served the page itself
* Information about you can be sent to third parties without knowledge or consent
  + IP address
  + Contents of cookies
  + Any personal info the site has about you
* Track people over different website

**Why malicious code**

* Issue of privacy more than security
  + Web bug instruct browser to behave in a way contrary to principle of informational self-determination
* Leakage of identity
  + With cookies, advertiser can learn what website person is interested in
    - E.g.: cross site tracking – amazon
  + Advertiser cannot learn person identity
    - Unless place ads on social networking site

**Backdoors**

* aka trapdoor
* set of instruction designed to allow bypass the normal authentication mechanism
* allow acess to the system to anyone who knows back door exists
  + sometimes its useful -> for debugging purpose
* e.g.: Code Red Worm, port knocking
* Source of backdoors
  + Forget to remove
  + Intentionally leave for testing purposes
  + Intentionally leave for maintenance purposes
  + Intentionally leave for legal reasons
  + Intentionally leave for malicious purposes

**Salami Attacks**

* attack made of smaller, inconsequential attacks
* classic example: send fractions of round off error from many accounts to a single account owned by attacker
* Commonly in:
  + Credit Card thieves make small charges to many cards
  + Clerks slightly overcharge customer for products

**Privilege Escalation**

* most systems have concept of differing levels for different users
  + root and normal users
* an attack which raise the privilege level of the attacker
* source:
  + flaw occur in part of system that legitimately runs with higher privilege
  + buffer overflow in setuid programs or network daemons
  + attacker might trick system into thinking he is in fact legitimate higer-privileged user

**Rootkits**

* happens in Linux system
* tool used by “script kiddies”
  + system ships with backdoor
* 2 main parts
  + method for gaining unauthorized root/admin privileges
    - exploit known flaws in system
    - often leaves behind backdoor for attacker to go back later
  + way to hide its existence
    - clean up log messages that have been created by exploited
    - modify commands like ls and ps
      * don’t report files and processes belonging to rootkit
    - modify kernel -> no user program will learn
* **E.g.: Sony XCP**

**Keystroke logging**

* Install a keyboard logger to keep track of
  + Email/IM sent
  + Password typed
* Data can be accessed locally
* Or sent to a remote machine over internet
* Installed by malware
  + Capture password -> send to remote attacker
* Installed by family members
  + Spy on children/partners
* Kinds of keyboard logger
  + Application Specific
    - Record keystrokes tied to a particular app.
  + System logger
    - Record all keystrokes pressed
  + Hardware keyboard
    - Small piece of hardware that sits in between keyboard and computer
    - Undetectable in software

**Interface illusion**

* use of user interface to control computer
  + - * + fake login screen
* expected system to behave in certain way it interacts with standard user
* malicious code make it non-standard to trick users
* caused by Conficker worm
* Windows NT prevention feature at login page
  + - * + CTRL + ALT + DEL

**Phishing**

* Example of interface illusion
  + E.g. visit paypal site, but its fake
  + Advanced phishers make site look authentic, but is fake
  + Require to check SSL certificate/address bar

**Man-in-middle attacks**

* Interface illusion, phishing, keyboard logging are examples
* System communicate with you plus attacker
* Man in the middle(attacker) intercept the communication then pass it to the intended other party
  + User thinks nothing is wrong
* Attacker can see everything user’s doing
  + Capture password
  + Hijack session to insert malicious commands
* Won’t see it happen on screen
  + **Attacker can edit results**
    - **Rootkit**

1. **Non-malicious Flaws**

**Covert Channels**

* Attacker creates capability to transfer sensitive information thru a cahnnel
  + Not supposed to transmit that information
* Information can/cannot be transmitted through a channel
  + Determined by policy/guidelines/etc.
* E.G.
  + Eve can arrange for malicious code to run on Alice’s machine
    - Alice closely watches all internet traffic from her computer
  + Alice publish weekly report (non-sensitive)
  + Even can “hide” sensitive data in report
    - Modification to wording, stats

**Side Channels**

* more powerful attacks than covert channels
* E.G
  + Eve watch how Alice computer behaves
  + Eve can learn information about what Alice’s computer is doing
    - Power consumption
    - Audio emission
    - Shared CPU cache
    - Time to perform a computation
* Powerful attacks when Alice’s computer is a smart card
  + Store some kind of secret
  + Physically in Eve’s possession

**6) Controls against security flaws**

* Tough job to control threat

**Software Lifecycle**

* Several stages
* Important at **Design & Implementation** stage
  + **Why? ->** fixing bug/flaws at later stage is expensive
* Design
  + Modularity
    - * Break problem into smaller pieces
      * Each responsible for single subtask
      * Complexity = smaller => easier to check for flaws
      * Conclusion: modules = low coupling
  + Encapsulation
    - * Modules must be self-contained (cannot call private method from outside)
      * Share information only as necessary
      * Reduce coupling
      * Developer should not need to know how a different module is implemented
        + Only need to know another modules’ API
  + Information hiding
    - * Internals should not be visible to others
      * Stronger expression than encapsulation
      * Implementation & internal state should be hidden from developer
      * Prevent accidental reliance
  + Confinement
    - * Use of sandboxing
        + if module A calls module B

if B is untrustworthy = > confine it

* + - * useful if B is code downloaded from internet
* **Implementation**
  + **DON’T USE C**
  + Static Code analysis
    - * Look out for patterns that cause flaws in C, C++, Java, etc
        + Buffer overflow
        + TOCTTOU
      * Tools are not perfect
        + Meant to find suspicious things
        + Also miss things -> not only line of defence
  + Formal methods
    - * Costly - > require machines to run programme
      * Use prove to check code
        + Another approach compare to looking for patterns
      * Programmer will mark up code with assertions/hints to theorem proving program
      * Time consuming
  + Genetic diversity
    - * Reason worms/virus are propagating quickly - > many machines running the same vulnerable code
        + Malware exploits this code
      * **Use of different servers = less likely for common flaw**
        + If everyone grow the same crop, most probably be wiped by a single virus
* **Security Controls – Change Management**
  + project lifecycle is work with large group of people
    - * control process to track who wrote which code
  + Source Code & Configuration
    - * Track changes to source code/configuration information
        + Git, subversion, etc
* **Security Controls – Code Review**
  + Review codes = effective to find faults
  + Have people > code author to find flaws
  + Open source software
    - * Anyone can look at code
      * Might still have vulnerabilities (undiscovered)
  + Give the code to reviewers for code review
    - * Look it over
      * Spot problems
      * Open source model
    - Guided code review
      * More useful (guided walk through)
      * Author explains the code
      * Justifies why it was done in that approach
      * Useful for changes to code
        + Why each change made
        + What testing needs to be done
      * **TAKES MORE TIME**
      * **Important for safety critical system**
    - Easter egg code reviews
      * Prevent reviews to say “fine” after nothing to be found
      * Author inserts intentional flaws
        + Let reviewers look harder
        + More likely to find unintentional flaws
* **Security Controls – Testing**
  + test = implementation meets specification
  + 2 strategies
    - * make program do unspecified things by doing unusual things to it
      * make program do unspecified things by considering of design and implementation
    - Black Box
      * Only access to completed object
      * Fuzz testing – supply random data to object
        + Test an input in an API
        + As a data file
        + Cause program to crash surprising often

Violations of availability

* + - White Box
      * Taking to account knowledge of design and implementation
      * Tie to code review
      * Useful for regression testing
        + Make comprehensive set of tests
        + Ensure program pass them
* **Security Controls – Documentation**
  + write down choice made
    - * note down things that didn’t work
      * aid future developer
    - make checklist of things to be careful of
* **Security Controls – Maintenance**
  + Modifying programs
  + Change management, review, testing, documentation
  + Standards, Process, Audit
    - * Rules incorporate the controls
        + E.g: what kind of code review
        + E.g: what type of testing
      * Make formal process specifying how each standard should be implemented
      * Have audits to verifies user if he/she follow processes properly
      * **Does not guarantee flaw free code**

CS 458 Module 3 Operating System Security

**Operating System**

* Allows different user to access different resource
  + In shared way
* Controls sharing and provide interface to allow access
* Identification and authentication are required

**1) Protection in general-purpose OS**

**History**

* OS evolved to allow multiple user to use the same hardware
  + Sequentially -> executives
  + Interleaving -> monitor
* Make resources available to users if required by them
  + Permitted by some policy
* Protect user from each other’s
  + Attacks/mistakes
  + Resource consumption
* Protection against mistakes/malware

**Protected Objects**

* Memory
* CPU
* IO
* Programs
* Data
* Networks
* OS

**Separation**

* Keep user object separate from others
  + Physical
    - Use different physical resource for different users
    - Easy implement
    - Expensive and inefficient
  + Temporal separation
    - Execute different user’s programs at different times
  + Logical Separation
    - User is given the impression that no other users exist
  + Cryptographic
    - Encrypt data = intelligible to outsiders
    - Complex

**Sharing**

* Users might want to share resources
  + - * Files/DB records
    - OS needs to provide flexible sharing

**Memory & Address Protection**

* Prevent program from corrupting others
* OS can exploit hardware support (cheap)
* Memory protection is part of translation from virtual to physical address
  + Memory Management Unit(MMU) generate exception
    - If something is wrong with virtual address/request
  + OS maintain mapping tables used by MMU and deals with exceptions

**Protection Techniques**

* Fence Register
  + Exception if memory access below in fence register
  + Protects OS from user programs
  + Single-user OS only
* Base/Bounds register pair
  + Exception if memory access below/above in base/bound register
  + Different value for each user program
  + **Maintained by OS context switch**
  + Limited flexibility
* Tagged Architecture
  + Each memory word has one/more extra bit that identify access rights
  + Flexible
  + Con: large overhead
  + Not all hardware architecture support
* Segmentation
  + Each program = multiple address spaces (segments)
  + Different segments for code, data, stacks
  + Virtual addresses consist of two parts
    - <Segment name, offset within segment>
  + OS keeps mapping from segment name to its bases physical address in segment table
    - Table for each process
  + OS can relocate or resize segments and share them between processes
  + Segment table also keeps protection attributes

|  |  |
| --- | --- |
| **Advantage** | **Disadvantage** |
| Each address reference is checked for protection by hardware | External fragmentation |
| User can share access to segments = potentially different access rights | Dynamic length of segments = costly |
| User cannot access an unpermitted segment | Segment names are difficult to implement |

**Paging**

* program is divided into equal-sized chunks (pages)
* physical memory is divided into equal sized chunks (frames)
* frame size = page size
* virtual address consists of
  + <Page #, offset with page>
  + #bits for offset = log2(page size)
* OS keep mapping from page # to its base physical address with Page Table
* Keep memory protection attributes

|  |  |
| --- | --- |
| **Advantage** | **Disadvantage** |
| Address reference is checked for protection by hardware | Internal fragmentation |
| Users can share access to a page with potentially access rights | Assigning different levels of protection to different classes of data item not feasible |
| Users cannot access an unpermitted page |  |
| Unpopular pages can be moved to disk to free memory |  |

**x86 Architecture**

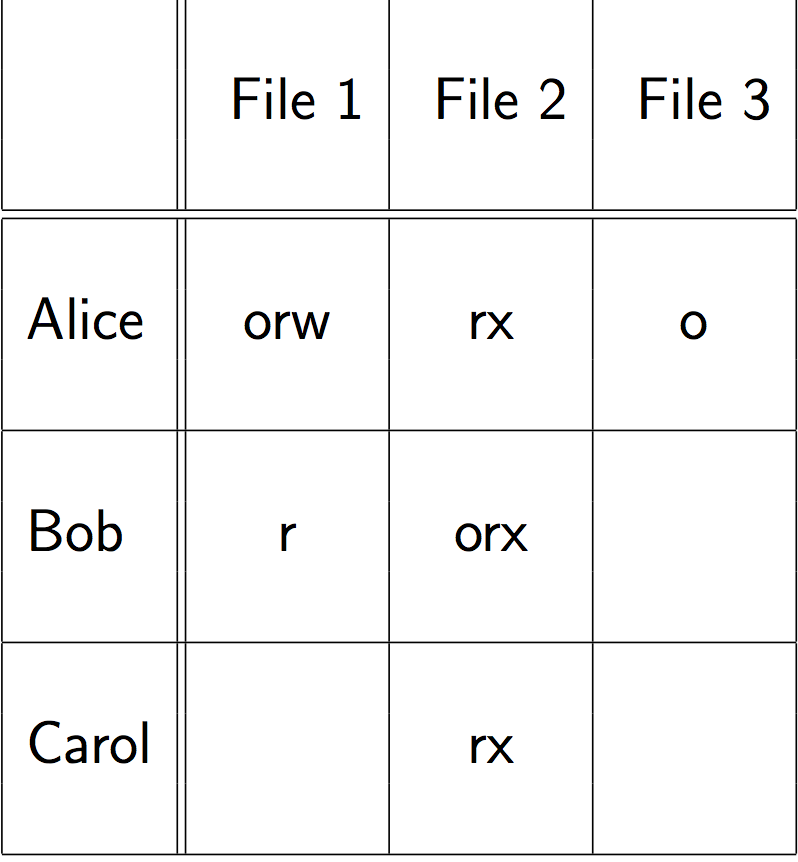
* Has both segmentation and paging
  + Windows and Linux
* Memory protection bits indicate
  + no access
  + read/write access
  + read only access

**2) Access Control**

* memory only one of the many objects for which OS must run access control
* access control has 3 goals
  + Check every access:
    - Else OS might fail to notice access has been revoked
  + Enforce least privilege
    - Grant program access only to SMALLEST # of objects to perform a task
  + Verify acceptable use
    - Limit types of activity that can be performed on an object
    - For integrity reasons (ADTs)

**Access control matrix**

* Set of protected objects: O
  + E.g. files or database records
* Sets of subjects: S
  + E.g. users, processes acting behalf of humans or group of humans/processes
* Set of rights: R
  + Read, write, execute, own
* Access control matrix consist of entries a[s , o]
  + Where s is element form Subject
  + Where o is element from Objects
  + **a[s , o]** is subset of Rights

****

**Implementing access control matrix**

* **rarely implemented** 
  + lots of memory would be unused.
* Implemented as
  + Set of access control list (column-wise representation)
  + Set of capabilities (row-wise representation)
  + Or both together

**Access Control List(ACL) \*\*\*\*\*\*MIDTERM QN\*\*\*\*\* Slide 3-24**

* Each object has list of subjects and their access right
* E.g:
  + File 1
    - Alice ORW
    - Bob R
  + File 2
    - Alice RX
    - Bob ORX
* ACL implemented in Windows File system (NTFS)
  + User entry denote entire user group
* Each file list its owner, group, and a third entry representing of other users
  + Each class have its own set of right
  + Groups are system wide

**Capabilities**

* Unforgeable token
  + Give owner access rights to an obkect
  + E.g.:
    - File 1: ORW
    - File 2: RX
    - File 3: O
  + unforgeability enforce by having OS store and maintain token
    - or by cryptographic manner
  + e.g.:
    - digital signature allow tokens to be handed out to process/user
    - OS detect tampering when they tries to get access with modified token
  + Token can be transferable

**Combined usage of ACL and capabilities**

* Use both ACL and Cap
  + UNIX file system -> each file has an ACL
    - Consulted when executing open() call
    - If approved -> caller is given capability listing type of access in ACL
    - Upon read()/write() -> OS look at capability first

**Role Based Access Control**

* Object that a user can access DO NOT depend on identity of user
  + Only on user’s job function within company
  + E.g: salesperson can access customer CC numbers
    - Marketing only customer names
* Administrator assign users to role
  + Grant access rights to role
  + Sounds familiar to group, but group is less flexible
* When user take a new role
  + Update role assignment, not her access right
* Used in commercial DB

RBAC Extensions

* Hierarchical roles
  + Manager = employee
    - Reduce number of role/access right assignments
  + User can have multiple role
* Separation of Duty
  + An object requires 2 different roles to write
    - Cannot be same person

**3) User Authentication**

* **System** need to identify and authenticate users
  + **Identification:** Who are you
  + **Authentication:** Prove it
* Difficult for computers to authenticate people

**Authentication factor**

* **4 classes**
* Something user knows
  + Password/Pin to secret question
* Something user has
  + Card/token
* Something user is
  + Biometrics
* Something about user context
  + Location
  + Time
  + Device in proximity

**Combination of Authentication Factor**

* Different classes of authentication = more solid
  + 2 factor
* same class = not better
* E.g.:
  + ATM – Something user knows (password) and has (Card/token)

**Password**

* oldest authentication method
* problem?
  + Forgotten password + not recoverable
  + Inconvenient
  + If disclosed = might give others access
  + Can be shared among people = update more difficult

**Attack on Password**

* Shoulder surfing
* Keystroke logging
* Interface illusion
* Password reuse across sites
* Password guessing

**Password Guessing**

* brute force – try all combination
* exhaustive search
* root take from dictionaries
  + password from previous leaks
* **type of password guessing attack**
  + Offline – on file
  + Online – web
    - Bank block access after 5 tries

**Choose good password**

* Use letters
* Numbers
* Special characters
* At least 8 characters
* Avoid guessable root
* Use pass phrase
  + Mix upper and lower case
  + Introduce spelling mistakes

**Password Hygiene**

* Write down better than store on networked computer
* Change regularly
  + Password recycling
* Have site-specific password
* Don’t reveal to others
* Don’t enter password on public computer

**Attack on password files**

* Web/com need to store information about password in order to validate
* Storing = dangerous
  + Password file might end up on backup tapes
  + Attacker to OS might get access

**Fingerprint password**

* Store digital fingerprint of password
  + Cryptographic hash
* Still allow offline guessing attacks when file leaks

**Defences against guessing attacks**

* UNIX use user-specific salt in password fingerprint
* Don’t use cryptographic hash (SHA-1) to compute stored fingerprint
  + Cheap to compute
* Use iterated hash function that is expensive to compute
* Additional defence is to use a MAC
  + Mix in secret key to compute password fingerprint
  + Protect secret key -> embed in resistant hardware

**Password Recovery**

* Cannot be recovered from a hash value
* Need to store an encrypted version in password file
* Need to keep encryption key away from attacker

**Case Study: Adobe Password Hack 2013**

* **130** million encrypted passwords revealed
* **encryption mechanism** 
  + append null to password
  + padded password were encrypted 8 character using a fixed key
    - weak encryption mode

**Interception Attacks**

* Attacker intercept password
  + While transmission
* OTP = intercepted password useless for later login
  + Challenge response protocols
    - Serer send client random challenge
    - Client use challenge and password to compute OTP
    - Client send OTP to server
    - Server validate
* Cryptographic protocols
  + make intercepted information useless to attacker

**Graphical password**

* Android slide to unlock
* Graphical than text based
* Multiple technique
  + Choose picture
  + Choose set of places in picture
* Issues similar to text based password
  + Choice of places is not random

**Server Authentication**

* System check user
* User should also check system
  + Else password might end up with attacker
* Classic Attack
  + Program display fake login (Paypal)
* Today attack
  + Phishing

**Biometrics**

* Way to get rid of problems with password/token based
* Have own problem
  + Authenticate user based on physical characteristic
    - Fingerprints, iris scan, voice
  + If trait is sufficiently close to store strait
    - Might accept user

**Local vs Remote authentication**

* Biometric suit local
  + Own finger on fingerprint scanner

**Authentication vs Identification**

* Authentication
  + Captured trait = stored traits?
* Identification
  + Captured trait match any of the stored straits
  + More expensive search problem
  + Made worst by biometric
  + Matches are based on closeness
  + Not on equality
* False positives can make biometric identification useless
  + False positive: Alice accepted as Bob
  + False Negative: Alice is incorrectly rejected

**Other Problem with Biometric**

* Privacy
  + Why employer/website have information of fingerprints
* Accuracy
  + False negative = annoying
* Secrecy
  + Some biometric are not secret
    - Face/fingerprints

**4) Security policies and models**

* Idea of certain components be trusted in what way
* Trust entity =
  + If entity misbehave
  + System fails
* Trust OS if we have confidence that it provides
  + Memory and file protection
  + Access control and user authentication

**Trusted OS**

* **Policy**
  + Rules outlining what is secure and why
* **Model**
  + Model that implement policy
  + Reasoning about policy
* **Design**
  + Specification how OS implement for model
* **Trust** 
  + Assurance is designed according to design

**Trusted Software**

* Software **=** rigorously developed and analysed
  + Reason to trust what code does
  + Expect nothing more
* Functional Correctness
  + Software works correctly
* Enforcement of integrity
  + Wrong input doesn’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
  + Based on experience

**Security Policy**

* OS security policy have their roots in military security policy
  + Where research funding came from
* Each object/subject has sensitivity/clearance level
  + Top secret
  + Secret
  + Confidential
  + Unclassified
* Each subject might be assigned to one or more compartments
* Subject can access object if and only if subject > object and compartments subject is a subset of compartments object

**Commercial Security Policy**

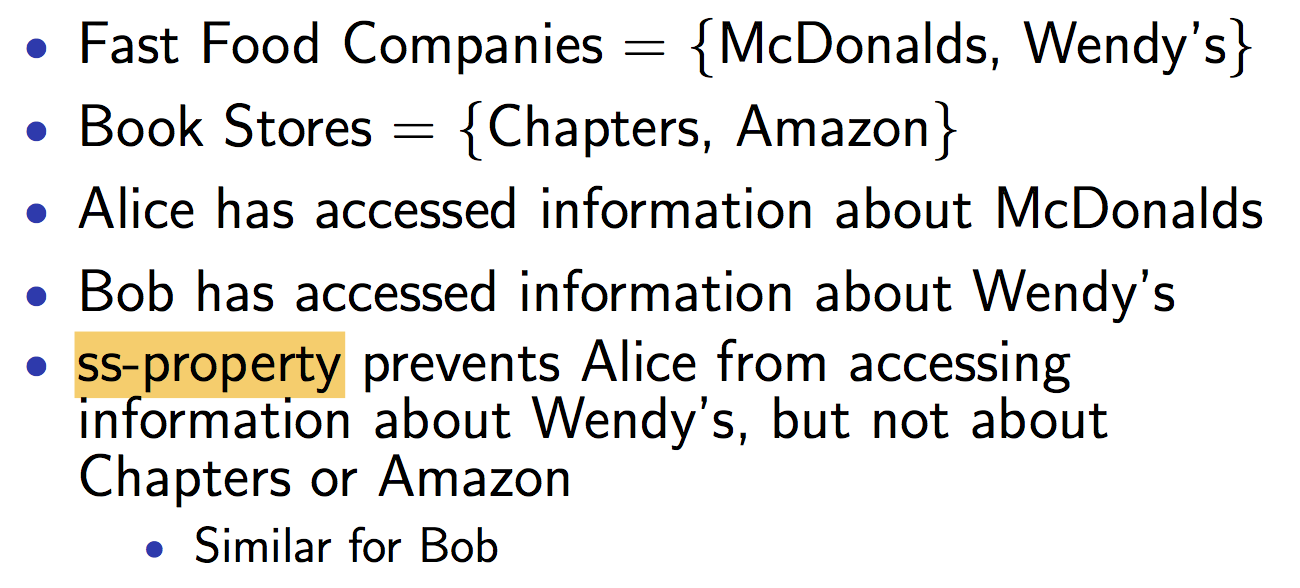
* rooted in Military security policies
* different classification levels for information
  + internal vs external
* different departments can call for need to know restriction
* assignment of people to clearance level not as formal defined
  + maybe ad hoc basis

**Other Security Policy**

* Integrity of information can be > confidentiality
* Clark-Wilson Security Policy
  + Based on well-formed transaction that transition system from consistent state to another
  + Support RBAC – separation of duty
* Dealing with conflict of interest
  + Chinese Wall Security Policy
  + Once decided a side of the wall, no easy way to get to other side

**Chinese Wall Security Policy**

* Once able to access information of one company
  + No longer access information about other companies of same kind
* **SS-property:** 
  + Subject s can access object o IF AND ONLY IF
    - Object previously accessed by s belongs to same company as o
    - Or belongs to a different company than o does

****

* **\*-property:**
  + for a write access to o by s
    - need to ensure all objects readable by s
    - belong to same company as o or have been sanitized

**Security Model \*\*\*\*\*Possible Exam Question\*\*\*\*\*\*\*\***

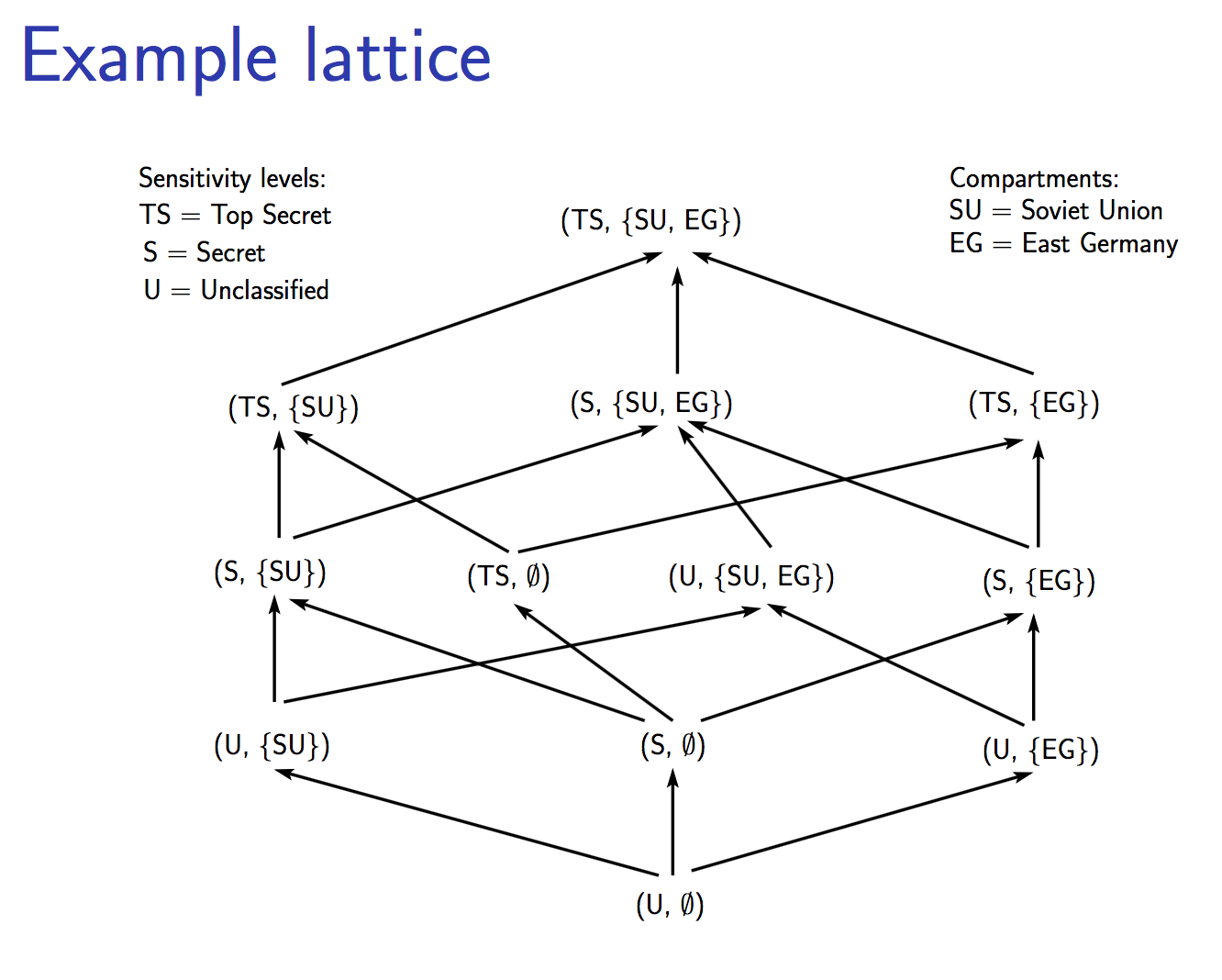
* Bell-La Padula Confidentiality Model
* Biba Integrity Model

**Targeted at multilevel security (MLS) policies**

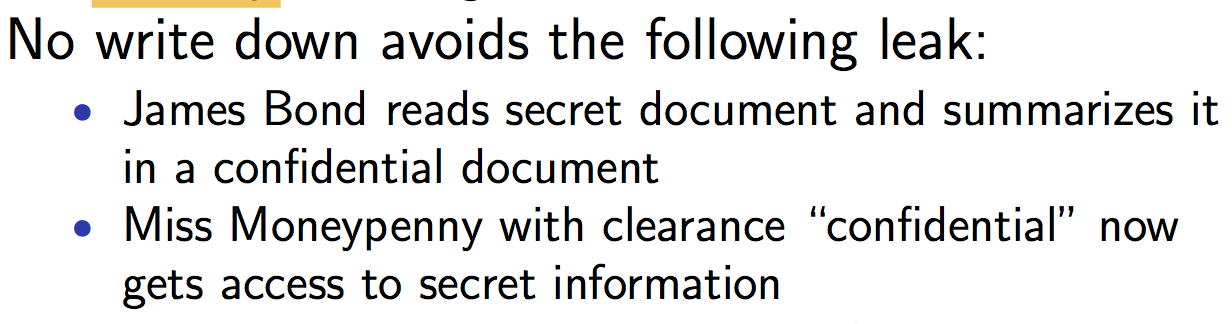
* Subject /objects have clearance level

**Lattices**

* Dominance relationship
  + Defined in military security model is transitive and antisymmetric
  + Defines a partial order
    - NEITHER
    - A dominate b
    - OR
    - B dominate a
* In a lattice, for every A & B
  + There is a unique lowest upper bound u
    - U bigger or equal to A
    - U bigger or equal to B
  + And unique greatest lower bound I
    - A bigger or equal to I
    - B bigger or equal to I
* There is also 2 elements U & L that
  + Dominate by all levels

****

**Bell-La Padula Confidentiality Model**

* Regulate information flow in MLS policies
  + Lattice-based
* Users get information only
  + According to their clearance
* Subject s with clearance C(s) have access to object o with sensitivity C(o)
  + Information can only flow up
* SS-property: No read up
  + S should have read access to o if C(s) >= C(o)
* \*-property: no write down
  + s should have write access to o if C(o) >= C(s)
* can write above me, but private can’t only write down
* no read up = can read only at current or lower level
* ****

**Biba Integrity Model**

* Prevent inappropriate modification of data
* Dual of Bell-La Padula Model
* Subject & Objects are ordered by an integrity classification scheme
  + I(s) and I(o)
* Write access:
  + S can modify o only if I(s) >= I(o)
* Read access
  + S can read o if I(o) >= I(s)
  + Unreliable information cannot contaminate subject

**Review of Security Model**

* Simple
* Makes it possible to prove properties
* Probably too simple for great practical benefit
  + Need declassification
  + Need both confidentiality and integrity
* Information leaks still possible through covert channels

**5) Trusted OS Design**

**Trusted System Design Elements**

* design = address which objects are accessed
* must be part of early design
  + hard to retrofit
    - Win95/98
* 8 design principles
* Least Privilege
  + Work with fewest privileges possible
* Economy of mechanism
  + Protect mechanism -> simple and straightforward
* Open Design
  + Avoid security by unknown
  + Secret key, not secret algorithms
* Complete Mediation
  + Every access attempt = checked
* Permission based/Fail safe defaults
  + Default should be denial of access
* Separation of privileges
  + Two or more condition must be met to get access
* Least common mechanism
  + Every shared method could be used as covert channel
* Ease of use
  + Cannot be too hard to use

**Features of Trusted OS**

* Identification and Authentication
* Access control
* Object reuse protection
* Completed mediation
* Trusted path
* Accountability and audit
* Intrusion detection

**Access Control**

* Mandatory Access Control(MAC)
  + Central authority establish who can access
  + Good for military
* Discretionary Access Control(DAC)
  + Owner of an object = control over who can access
  + Grant others access to your home directory
* RBAC
  + Not a MAC and DAC
* Possible to use combination of these mechanism

**Object reuse protection**

* Alice allocate memory from OS and store her password
* After using, return memory to OS
* OS should erase returned memory before handling it to others
* Defensive Programming
  + Erase sensitive data YOURSELF before return to OS

**Hidden Data**

* Related to Object reuse protection
* Think might be deleted, but still hidden
  + Mailbox
  + Deleting a file does not mean physically erase on disk

**Complete Mediation**

* All access must be checked
* Prevent access to OS memory if it is possible to swap space on disk

**Trusted Path**

* Give assurance to user
  + Keystrokes and mouse clicks are legitimate to receiver application
  + E.g. Fake login screen
  + Don’t run SUDO if you have untrusted application

**Accountability and Audit**

* **Integrity =** prevent attacker modify the log
* Keep audit log of security related events
  + Provide accountability
    - Who deleted data in DB?
* Audit log does not give accountability
  + If attacker can modify log
* Granularity
  + Might miss attack if not enough details

**Intrusion Detection**

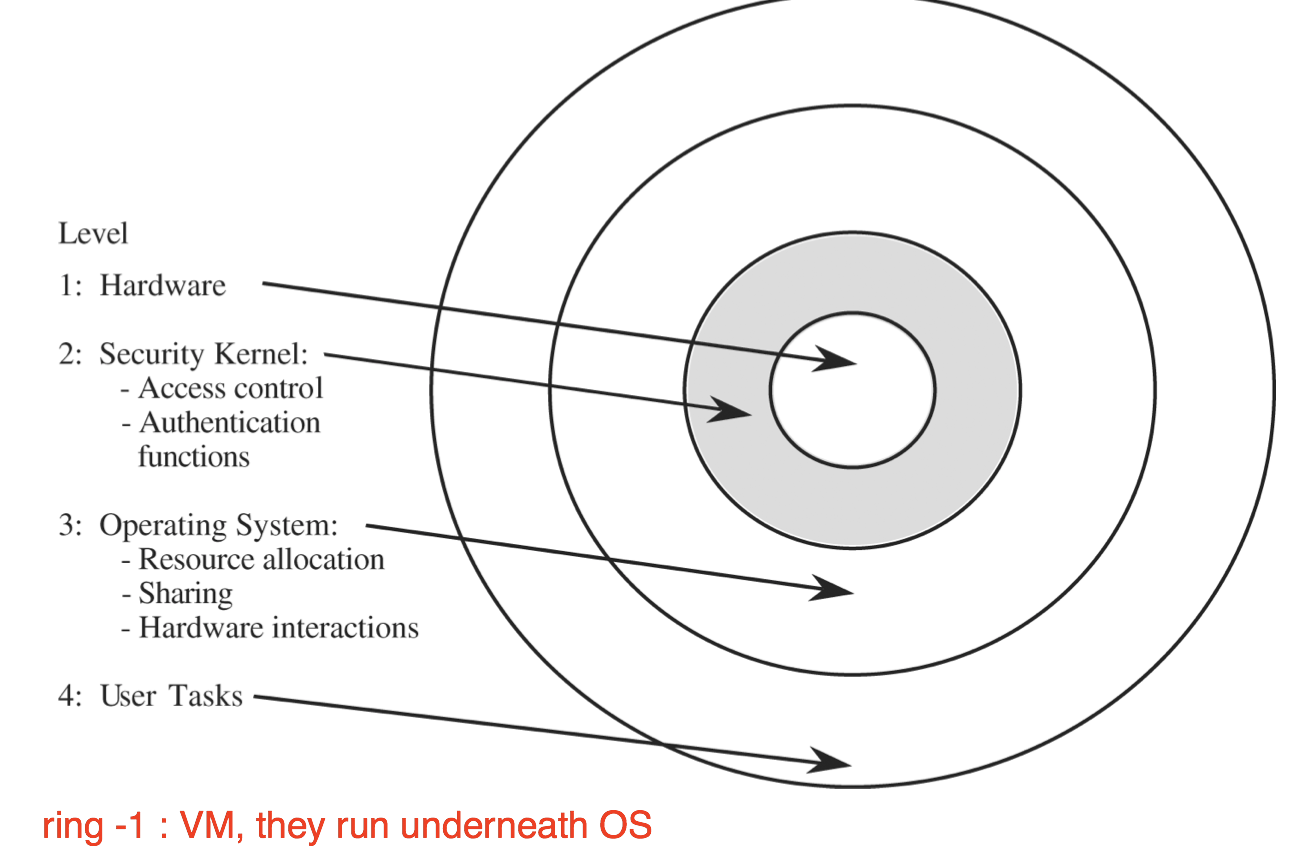
* Alarm if behaviour look abnormal
* No intrusion in a trusted OS
* Audit log to give information about an intrusion
* OS must detect if occur

**Trusted computing based (TCB)**

* part of trusted OS that is necessary to enforce OS security policy
* can be implemented either in different parts of OS
  + or in separate security kernel
  + security kernel runs below OS kernel
    - more difficult for attacker to attack

**Security Kernel**

* higher ring cannot access inner ring

****

**Rings**

* some processor support this kind of layering based on ring
  + code can access if
    - memory and instruction in rings >= n
  + access to ring < n
    - interrupt/exception by inner ring
    - deny/grant access

**Reference Monitor**

* pass in reference on some object, then check the object
* collection of access controls for devices,files, memory, IPC, etc.
* not necessarily a single piece of code
* must be tamperproof
* interacts with other security mechanism

**Virtualisation**

* way to provide logical separation
* different degrees of virtualisation
* Virtual Memory
  + Page mapping
    - Give each process impression of having separate memory space
* Virtual Machine
  + Virtualise I/O, files, printers
  + VMware
    - If browser in VM get attack
      * Limit to VM environment

**Application Insulation**

* Memory encryption allows application shielding from other apps
* Application is portioned into trusted and untrusted code.
* Trusted code is encrypted into memory
  + Using a key
  + Key live in secure hardware -> close to CPU
* Untrusted code
  + Communicate via compact API
* Trusted computing base = reduce secure hardware, CPU and small trusted code

**Least privilege in popular Oss**

* Traditional UNIX
  + Root process has access to anything
  + User process has full access to user data
* SELinux provide MAC for Linux
  + Allow implementation of least privilege
    - No more root user
    - Support both confidentiality and integrity
    - Difficult to set up
* Less invasive approaches for UNIX

**Chroot**

* Sandbox/jail command by changing its root directory
* Command cannot access files outside of jail
* Some commands/programs are difficult to run in a jail