Objectives (features of security system)

Confidentiality: Prevent unauthorized author should not learn sensitive information, including secrecy, existence of secrecy, meta-data, anonymity & unlinkability
-> privacy-related properties

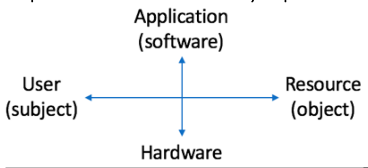
Integrity: Prevent unauthorized modification of information **Detect'n** of intentional & accidental modifications of transmitted data-checksums

Availability: The property of being accessible & usable upon demand by authorized entity. ('smurf' attack)

Accountability (Non-repudiation): It is concerned with attributes of an action (attack) to and entity(user)
Trace actions of entities; unforgeable evidence that a specific action occurred -> audit logs.
NR of origin: protects against a sender of data denying that data was sent. NR of delivery: -receiver Enforcement relies on public-key crypto techniques

[Reliability] the consequences of accidental errors **[Safety]** measure of the absence of catastrophic influence on the environment, especially on human life**[Dependability]** encompasses reliability + safety + security

[Dilemma] security unaware users with specific requirements but no security expertise

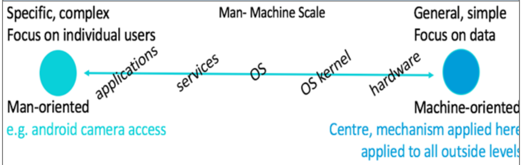


Design decisions

Where to focus the protection mechanism

[Data] format and content of data items. **[Operations]** what op allowed on data items. **[User]** who are allowed to access data item

Where to place the security mechanism



Complexity vs assurance

The location of a security mechanism on MMS is related to its complexity. Simple mechanism applied to everybody, but it may not meet specific requirements. However, to choose right features from rich menu is hard for security unware users.

Centralized vs decentralized security control

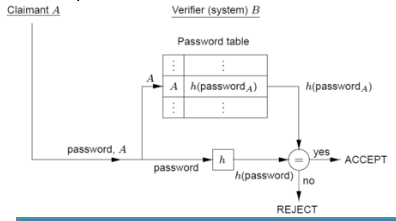
Within the domain of a security policy, the same control should be enforced. **[Centralized]** one entity in charge, easy to achieve uniform, BUT performance bottleneck. **[Distributed]** more efficient, BUT need to make sure policy consistency across different components.

Protection of the 'layer below'

[security perimeter] or security boundary, the parts of system that can disable the mechanism lie inside, those can malfunction without compromising the mechanism lie outside.
[E.g.] Forensic tools: restore data by reading memory/disks, it can circumvent logical access control as it does not care for the logical memory structure. **Unix treat I/O device like files**. **Buffer overruns**. **Object reuse**: data left behind in memory allocated to new process. **Backup**: whoever has access to a backup tape has access to all the data on it, logical access control does not help. **Core drums and system logs** may contain sensitive data.

[Data vs information] The meanings we assigned to data are called information; **Covert channel**: response time or memory usage may signal information**; Inference** in statistical database: combine statistical queries to get information on individual entries

[Password storage] plaintext or encrypted passwords
Hashed password verification



Attack & Defense

Exhaustive attacks (brute force)

Increase #char and #choices per char to increase entropy

$$H(x) = \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i}$$

if number of possible(equal) password is n

$$H(x) = \sum_{i=1}^{n} \frac{1}{n} \log_2 n = \log_2 n$$

Intelligent attacks: dictionary
Pre-compute hash table

Salting**[Salt]** a random string (not secret) each password has its own salt
Hash (password + salt)

Phishing: impersonate the system SMS
Spoofing: fake login

-Mutual authentication
-Trust path: make sure user communicate with the OS not spoofing page

[Password policies] aging, limit login, inform user
[Password file]: cryptography and access control
[alternative password] visual drawing pattern, picture passwords, one-time password, single sign-on (sign on chrome -> auto-sign on others)
[one-time password] -> reuse of stolen password. A list of password. E.g. Lamport's scheme (A send W3 to B, B has W4, and compute H(W3))

[Authentication protocol] weak -> password-based; unilateral: 1 entity proves its identity to the verifier strong -> mutual authentication: both parties take both the roles of claimant and verifier; challenge-response protocols: sequence of steps to prove knowledge of shared secrets.
[Relay attack] *Transferability*: B cannot reuse an id exchange with A to impersonate A to a third party C.
Impersonation: The probability that a third party B distinct from A, playing the role of A, can cause C to accept A's id is negligible
[Basis of authentication] what u know (password); what u have (token); who you are (fingerprints)
Authentication tokens and biometrics
Identification: 1: n, verification: 1:1

$$FMR = \frac{FP \text{（假的让通过）}}{FN + FP}, FNMR = \frac{TN \text{（真的被拒）}}{TP + TN}$$

EER: FMR = FNMR

$$FTA(aquire) = FTC + FTX \cdot (1 - FTC)$$
$$FA(accept)R = FMR \cdot (1 - FTA)$$
$$FR(reject)R = FTA + FNMR \cdot (1 - FTA)$$
$$FPI(identify)R = (1 - FTA) \cdot (1 - (1 - FMR)^n)$$

FP identification rate for a DB with *n* person
The **state** of a system is defined by (S, O, M)
Access matrix example

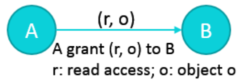
Subject \ object	M1	F1	P1
P1	R,W,E	Own, R, W	

HRU primitive commands (changes state of system)

create subject s	destroy subject s
create object o	destroy object o
enter r into $M_{s,o}$	delete r from $M_{s,o}$

```
command grant-read-file (p,f,q)
  if Own in M_{p,f} then enter Own into M_{q,f};
end
```

[Secure] R(reachable) ⊆ P (authorized) **[Precise]** R = P
Cascading revoke: A->B->C, revoke B -> revoke C
[Authorization graph]
Valid: nodes have access right can be traced back to source



Capabilities: rows of access control matrix
Access control list: columns of access control matrix

Discretionary	Mandatory
owner assign security policies	system wide policy ->who is allowed to access
refer to user ID	refer to security label
closed organizations	

Intermediate layer: e.g. group in UNIX or role
Role Based Access Control
More than 1 role; typical found in application level; role hierarchical RBAC: define relationships between roles
Constrained RBAC: add separation of duties
Negative permissions

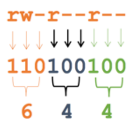
[Protection rings] each subject and object is assigned a number (0-OS kernel 1-OS 2-uilities 3-user processes)
Access control: compare the ring numbers
[Partial ordering (a ≤ b)] Reflexive a ≤ a; Transitive a ≤ b, b ≤ c => a ≤ c; Antisymmetric a ≤ b AND b ≤ a => a = b
[Hasse Diagram]
[VSTa ability] a list of integer; abilities are ordered by the prefix relation; empty string is the prefix of any ability
.1 ≤ .1.2 ≤ .1.2.4 but not .1 ≤ .4
CZ1002 student: .3.1.101 have access to object labelled .3.1.101 or .3.1 or .3 but not .3.1.105
[Lattices] partial ordering + l.u.b. + greatest lower bound
[System low] a label that is dominated by all other labels
When L is finite set, the element system low and high exist
[Multilevel security] m component => 2^m categories

------------------------------------ Unix ------------------------------------

root (UID 0): security check->off; become any user; change sys clock; can reset access right and password

User account/ passwords	`/etc/passwd`
username: password: UID: GID: name: homedir: shell	
password: crypt (3) 12-bit salt	
shell: program start after successful log in	
Groups	`/etc/group`
groupname: password: GID: list_of_users	
Shadow password	`/etc/shadow`
Also for aging and auto account locking	

[default permission] 666->new file, 777->new program
SetUID: U, G, S +octal representation
[umask] rights should be withheld
e.g. default: 666, umask: 077
umask (invert) -> 700; 666 AND 700 -> 600



Permissions for directories	
Read	Find files inside
Write	Add /remove files
Execute	cd / open files inside

Without r can still open the file, but cannot use ls
w + x for the directory (not the file)-> delete file
[Security Patterns]
1.SUID/SGID the user has the program's owner's privileges when running a SUID program
2.Deleting files copy a file: cp-> identical but independent ln -> a pointer to original file
If a process opens a file, which then is deleted by its owner, the file remains until that process closes the file
The deleted file left in memory
3.Protecting devices (Layer below)
Devices like files/ almost all devices should be unreadable and un-writable by 'other'
4.Mounting file system: when importing objects from outside, permissions must be redefined.
5. Env var: env USER='Bob' /home/alice/shell_program
6. Search path PATH in `.profile`
`PATH=.: \$HOME/bin:/usr/ucb:/bin`
The first entry '.' is current directory

Trojan horse: a program with same name, and be searched before the real one. Defense: call by full pathname
[Manage UNIX security]
Root account: separate the duties, if a special user is compromised, not all is lost; /etc/passwd and group must be write-protected, or attacker can change its UID to 0
Trusted hosts user from trusted host can login without password; only the same username on both hosts
Audit logs

------------------------Software security------------------------------

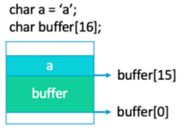
Command	Example
`more`	`more etc/passwd`
`chown`	`chown dieter:staff foo.txt`

`chmod`	`chmod 0754 filename`
	`chmod u+rwx, g+rx, g-w…`

C-String '\0' strcat: append

Input Validation
rlogin `rlogin –l –froot <machine>`
integer overflow | array overwrite
Security mechanism is case sensitive File system is case-insensitive
`../` -> step up (attack may hide '..' by UTF-8 encoding)

[Buffer Overruns]
STACK: return address; local variables and function arguments
HEAP: dynamically allocated memory



[format string] printf ->retrieve content of the stack
[Shellcode] overwrite return address so that execution jumps to the attack code ('shellcode') e.g. argv []-method
[Heap Overruns] overwrite filenames, functions pointers and mess up memory management
Type-safe languages(Java) guarantee the memory management is 'error free'
[Scripting] Cat file | mail to@me | rm –rf/
$sql = "SELECT * FROM client where name = Bob OR 1=1 --" the entire database is selected
[Shell] ';' sequential composition; '|' piping; '<','>' input/output redirection
shellshock: assign function to an environment variable env x=' () {:;}; malicious_code'
[Race conditions] change a value after it has been checked but before being used
[Logical errors] e.g. including typo or editing error

Hardware (Prevention)
1) A separate register for the return address
May not work on multi-threading program
2) Memory management unit configured to stop code from being executed at certain areas of RAM
Security measures may break existing code

Safer function (Prevention)
C unsafe string handling -> replace unsafe functions by func where the number of chars is handled e.g. strncpy
Still need to get the char count right

Filtering (Prevention)
1)Whitelisting: specify legal inputs; accept legal, block else
If you forget about some specific legal inputs, a legitimate action may be blocked
2)Blacklisting: specify forbidden inputs; block, accept
Forget -> attack may still get through
3)Taint analysis: mark inputs from untrusted sources as tainted, stop exe if a security critical functions gets tainted input, sanitizing tainted input and produce clean output
Type safety (Prevention) depends on definition of error

Canary (Detection)
Place a check value (canary) in memory just below the return address; before return, check if the canary changed Source code need to be re-compiled to insert placing and checking of the canary
Code Inspection (Detection) -> known problems
Untrustworthy sources -> sanitizing checks -> trust sink
If untrustworthy gets to sink without proper checks ->alarm
Testing (Detection) -> known problems Black-box testing
Least Privilege (Mitigation 减轻罪行)
Limit privileges required to run code; if code running with few privileges is compromised, the damage is limited
Keeping Up-to-date (Reaction)
Broken Abstraction: general patterns in insure software.
[State machine model]
1) define state set so that it captures 'security'; 2) check if all state transitions starting in a 'secure' state yield a 'secure' state; 3) check if initial state of system is 'secure'

---------------------Bell-LaPadula Model (BLP)---------------------

Subjects and objects with security levels that form a partial ordering, no information flow from 'high' to 'low' (confidentiality policy)
Access permission = access control matrix + security levels
[State set] access op: (s,o,a) subject, object, access right
assignment of security levels:
$f = (f_s, f_c, f_o)$, **F** the set of security level assignment
M: the set of access control matrix
State set: V = B×**M**×**F,** B:(S×O×A)
A state is: (b, **M**, f) b ⊆B
Discretionary security property
access must be allowed by access control matrix
Simple security property (no read-up) $f_s(s) \geq f_o(o)$
Start property (no write-down)
$f_s(s) \leq f_o(o) AND\ f_o(o') \leq f_o(o), all\ o' observed\ by\ s$
[Security Theorem]
a **state** is secure, if all current access tuples(s,o,a) are permitted by ss-, *-, ds-properties; a **state transition** is secure, if it goes from a secure state to another
Basic secure theorem: if the initial state is secure, and all state transitions are secure -> the system is secure
Tranquility: the security level of subjects and objects does not change. Optional: the security level of trusted subjects may change by following given rules.
Covert Channels are not detected by BLP modeling

Limitations restricted to confidentiality; no policy for changing access rights; a complete general downgrade is secure; BLP intended for systems with static security levels; it contains covert channels (detect existence <- deny access)

-----------------------------Windows-----------------------------

[SID] (globally unique across different computers in a domain) for authorization of access. **Random number** generators: generated SID when a new subject is created. manual generator -> complex as the domain size grows
A main **concern**: prevent a user to access objects belonging to another user in the same domain, including past users who have left. ><Unix security deals with a **single host** only and does not rely on globally unique SID to access resources in other computers in the network.
NULL DACL: everyone gets all access<=> empty DACL
[Static inheritance] ACEs inherited from the container(dir) when a new object is created; later changes to the container have no immediate effect on this object. (changes to take effect->propagat'n algo, re-run x change)

Inheritance flags indicate if the ACE is inherited	
`INHERITED_ONLY_ACE`	ACE only used for inheritance, not applied to the object itself
`NO_PROPAGATE_INHERIT`	Inherited by the next generation, not propagated
`OBJECT_INHERIT_ACE`	Inherited by all sub-objects that are not container
`CONTAINER_INHERIT_ACE`	Inherited by sub-objects that are container

InheritedObjectType specifies the object type that inherits the ACE; when an object is created, only ACEs with matching InheritedObjectType or without an IOT (NULL) are copied to its ACL.
SE_DACL_PROTECTED block ACE inherited from ancestor
Order: locally added ACEs -> inherited ACEs
ACEs from closer containers -> distant containers
A positive ACE can appear before a matching negative ACE
CREATOR_OWNER: put in inheritable ACEs; replaced with SID of the object's creator when the ACE is inherited
Owner Rights: used to limit access rights of owner
PRINCIPAL_SELF: for ACEs on objects representing a principal, e.g. user, group, replaced with SID of that principal during access check
ACE (Solution folder)

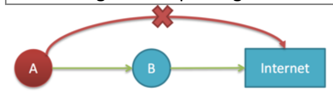
Type: access allowed (access denied) Access right: read, write Principal: Tutor (CREATOR_OWNER, NULL) InheritedObjectType: Solutions

-----------------------------Android-----------------------------

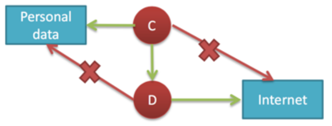
Intent filter: which implicit intent can receive (check by sys)

leaking information via logs
Android provide a centralized logging facility via log API Early -v: app with READ_LOGS permission can read the content of logs, now app can only read their own logs

leaking information via IPC
Any app can receive a broadcast that does not specify the target component. This is unsafe, if the Intent contains sensitive information, an malicious app with an appropriate filter may be able to intercept the intent
Unprotected Broadcast Receiver
App use broadcast receiver components to receive intent messages. 'Intern filters' defined by broadcast receivers are public. If the receiver is not protected by a permission, a malicious app can forge(伪造) message.
JNI & C/C++ library vulnerability
Android allow apps to call lib in C/C++, which may be prone to buffer overflow. If the libraries be called run under root privilege, a compromise could result in attacker gain root privilege.



"Confused deputy": B is not malicious, but used as a proxy to access internet.



Collusion: C and D are both malicious, but have different sets of permissions.

-------------------------------Others-------------------------------

An app can offer services. (This is done via the use of Intent Filter, which specify which Intents the app can handle. The app needs to implement a broadcast receiver to receive the intents) This app can impose the permissions required to access that service, but is not required to do so.

ACE2 //grant user permission to set their homepage Access rights: write Type: ACCESS_ALLOWED InheritedObjectType: {GUID for User Account Objects} ObjectType: {GUID for web homepage} Trustee (principal SID): PRINCIPAL_SELF

Need-to-know: an object is visible, only if the security label of the subject dominates the security label of the object.
Need-to-withhold: a list of compartments for which access needs to be withheld.