Protection & Security

- Intro to computer/information security
- OS Protection
- OS Security

Introduction to security

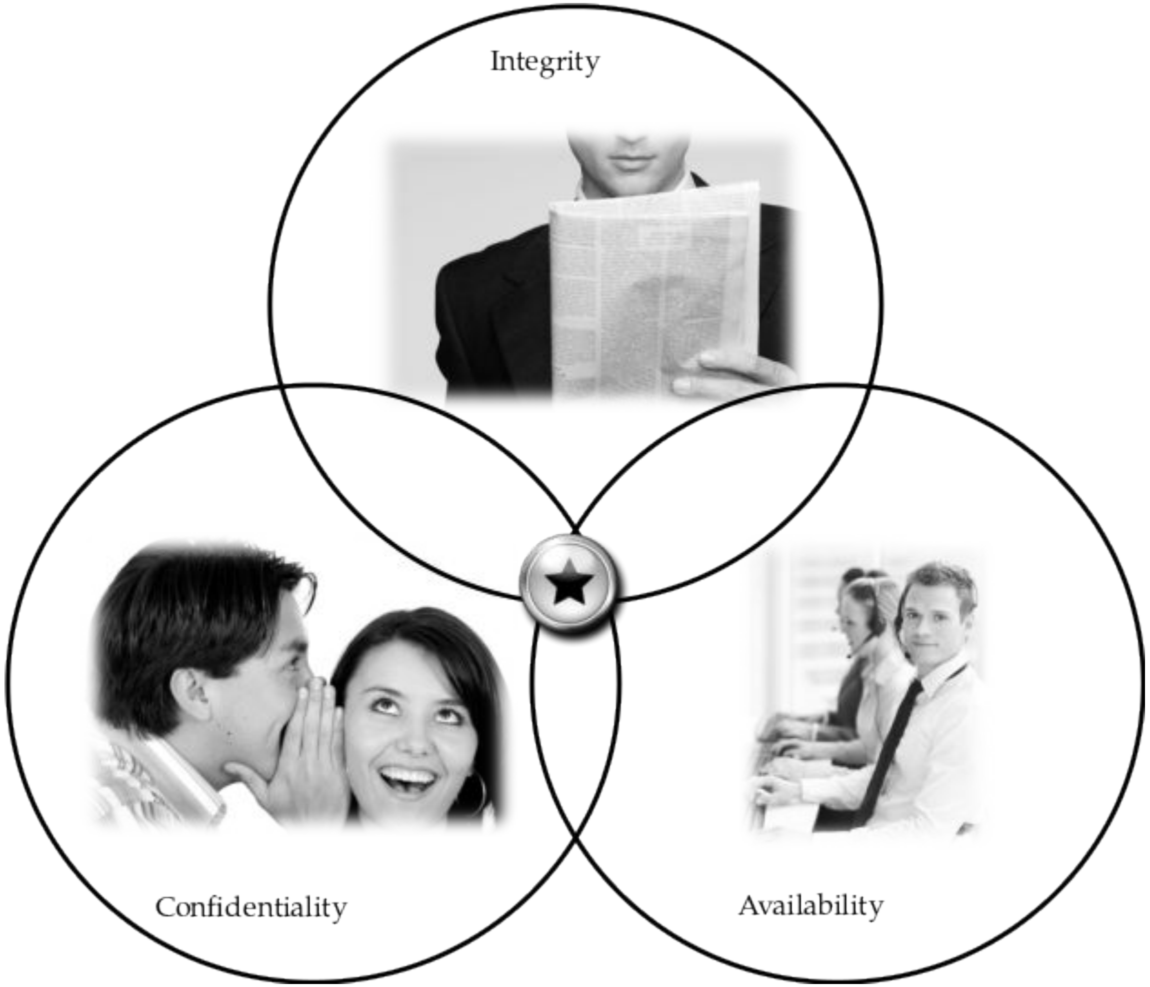
this part of slides are based on https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Introduction to Computer Security : Bishop, Matt

- The security of a system, application, or protocol is always relative to
    - A set of desired properties
    - An adversary with specific capabilities
- For example, standard file access permissions in Linux and Windows are not effective against an adversary who can boot from a CD

What to protect?

Basic security components: (CIA)



What to protect?

Basic security components: (CIA)

Confidentiality

the concealment of information or resources

- You cannot enter some offices in university
- You cannot see information of other students in ubys.medeniyet.edu.tr

Access control mechanisms support confidentiality.

- Example: **a cryptographic key:** scramble data so only people with the key can read/understand

Integrity

the trustworthiness of data or resources

- data integrity
- origin integrity
    - **Authentication:** source of data

Integrity mechanisms

- prevention mechanisms
    - Blocking any unauthorized attempts to change the data
    -
- detection mechanisms.
    - report that the data's integrity is no longer trustworthy

Availability

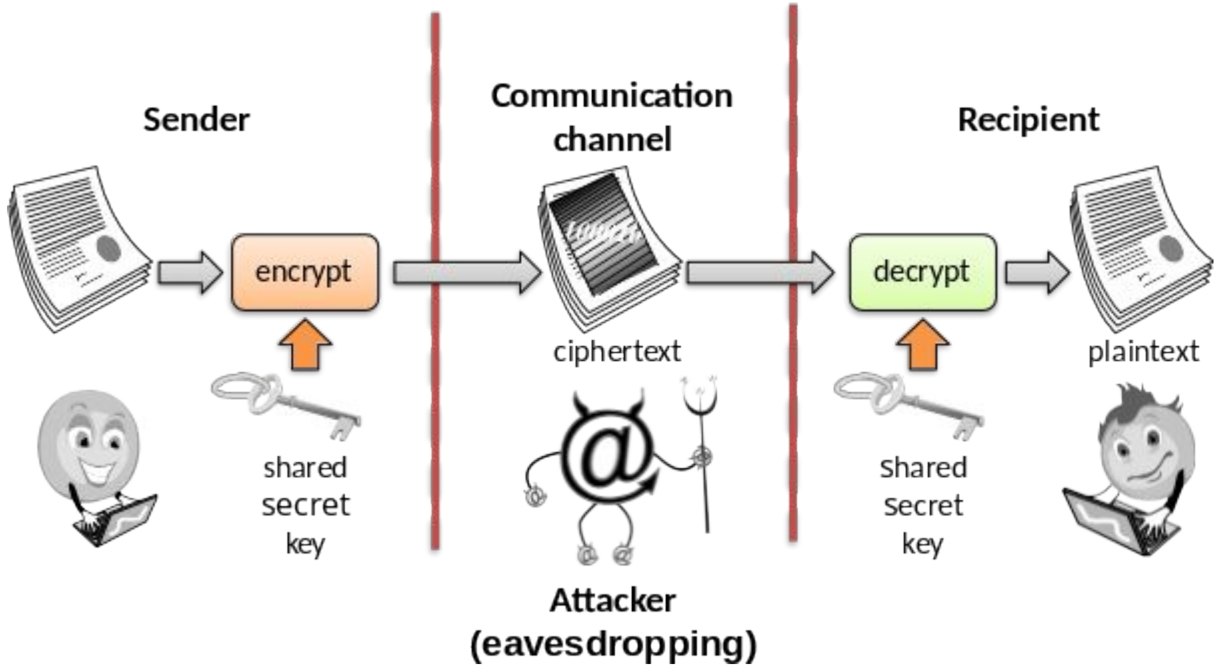
the ability to use the information or resource desire

Attempts to block availability, called **denial of service attacks**

Tools for Confidentiality

Encryption: the transformation of information using a secret,

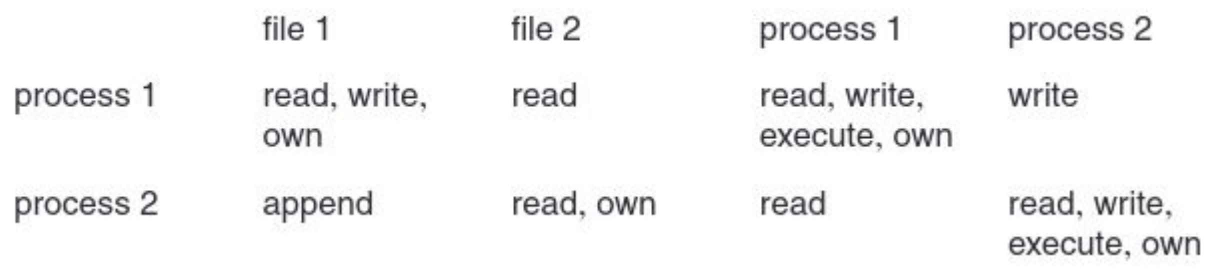
- the transformed information can only be read using another secret, called the **decryption key**



https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Tools for Confidentiality

- **Access control:** rules and policies that limit access to confidential information to those people and/or systems with a **"need to know."**

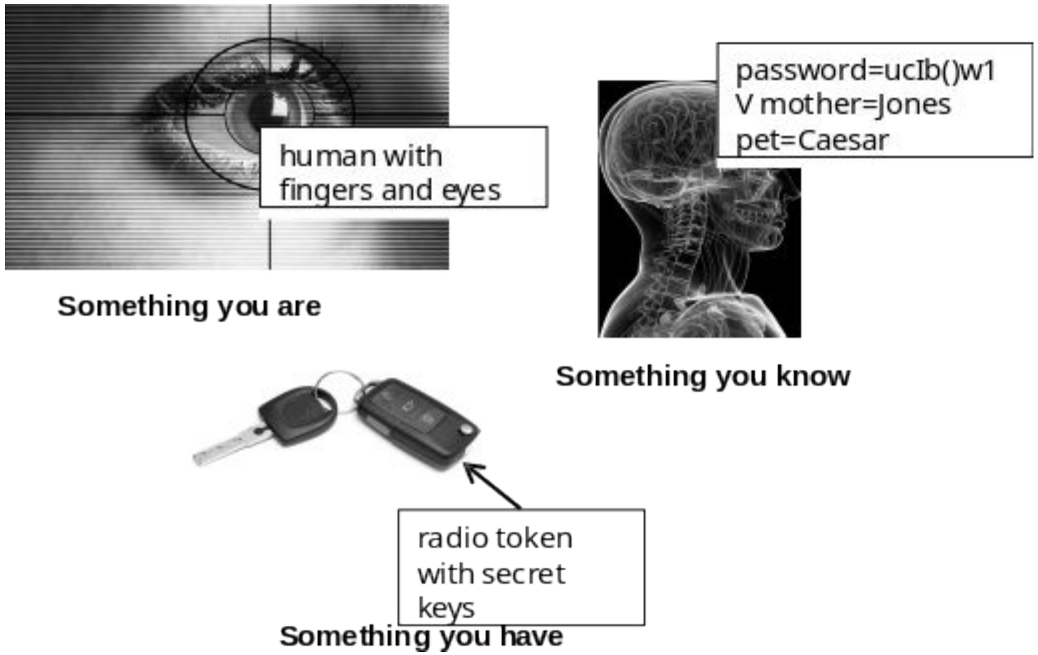


	file 1	file 2	process 1	process 2
process 1	read, write, own	read	read, write, execute, own	write
process 2	append	read, own	read	read, write, execute, own

https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Tools for Confidentiality

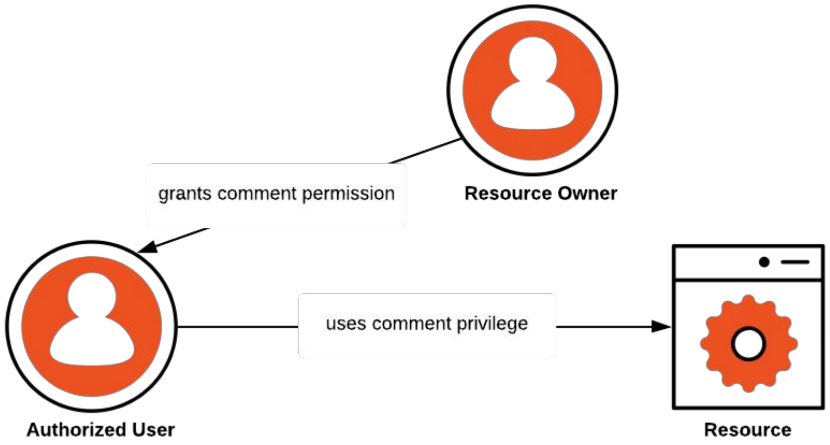
- **Authentication:** the determination of the identity or role that someone has. This determination can be done in a number of different ways
  - something the person has (like a smart card or a radio key fob storing secret keys),
  - something the person knows (like a password),
  - something the person is (like a human with a fingerprint).



https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Tools for Confidentiality

- **Authorization:** the determination if a person or system is allowed access to resources, based on an access control policy.
  - Such authorizations should prevent an attacker from tricking the system into letting him have access to protected resources.
- **Physical security:** the establishment of physical barriers to limit access to protected computational resources.
  - Such barriers include locks on cabinets and doors, the placement of computers in windowless rooms, the use of sound dampening materials, and even the construction of buildings or rooms with walls incorporating copper meshes (called Faraday cages) so that electromagnetic signals cannot enter or exit the enclosure.



https://auth0.com/intro-to-iam/what-is-authorization

Tools for Confidentiality

- **Authorization:** the determination if a person or system is allowed access to resources, based on an access control policy.
  - Such authorizations should prevent an attacker from tricking the system into letting him have access to protected resources.
- **Physical security:** the establishment of physical barriers to limit access to protected computational resources.
  - Such barriers include locks on cabinets and doors, the placement of computers in windowless rooms, the use of sound dampening materials, and even the construction of buildings or rooms with walls incorporating copper meshes (called Faraday cages) so that electromagnetic signals cannot enter or exit the enclosure.

Working of
Authentication and Authorization
Authentication
Authorization
Confirms users are who they say they are.
Gives users permission to access a resource

https://www.geeksforgeeks.org/difference-between-authentication-and-authorization/

e.g.: OpenID OAuth 2.0

https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Integrity and integrity tools

Integrity: the property that information has not be altered in an unauthorized way.

Tools:

Backups: the periodic archiving of data.

Checksums: the computation of a function that maps the contents of a file to a numerical value. A checksum function depends on the entire contents of a file and is designed in a way that even a small change to the input file (such as flipping a single bit) is highly likely to result in a different output value.

Data correcting codes: methods for storing data in such a way that small changes can be easily detected and automatically corrected.

https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Availability and Tools

Availability: the property that information is accessible and modifiable in a timely fashion by those authorized to do so.

Tools:

Physical protections: infrastructure meant to keep information available even in the event of physical challenges.

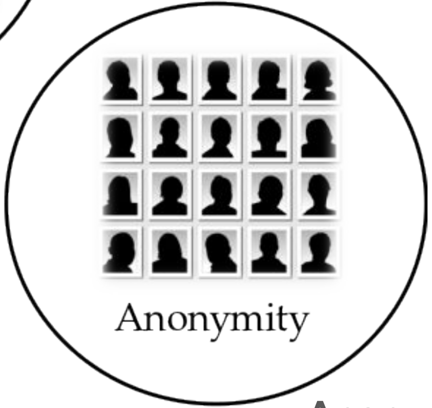
Computational redundancies: computers and storage devices that serve as fallbacks in the case of failures.

https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Other security concepts

Authenticity is the ability to determine that statements, policies, and permissions issued by persons or systems are genuine.

Assurance refers to how trust is provided and managed in computer systems.



Anonymity: the property that certain records or transactions not to be attributable to any individual.

https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Assurance refers to how trust is provided and managed in computer systems.

Trust management depends on:

- **Policies,** which specify behavioral expectations that people or systems have for themselves and others.
    - the designers of an online music system may specify policies that describe how users can access and copy songs.
- **Permissions,** which describe the behaviors that are allowed by the agents that interact with a person or system.
    - permissions for limited access and copying to people who have purchased certain songs.
- **Protections,** which describe mechanisms put in place to enforce permissions and policies.
    - prevent people from unauthorized access and copying of its songs.

Authenticity is the ability to determine that statements, policies, and permissions issued by persons or systems are genuine.

Primary tool:

- **digital signatures.** These are cryptographic computations that allow a person or system to commit to the authenticity of their documents in a unique way that achieves **nonrepudiation**, which is the property that authentic statements issued by some person or system cannot be denied.

Anonymity: the property that certain records or transactions not to be attributable to any individual.

Tools:

- **Aggregation:** the combining of data from many individuals so that disclosed sums or averages cannot be tied to any individual.
- **Mixing:** the intertwining of transactions, information, or communications in a way that cannot be traced to any individual.
- **Proxies:** trusted agents that are willing to engage in actions for an individual in a way that cannot be traced back to that person.
- **Pseudonyms:** fictional identities that can fill in for real identities in communications and transactions, but are otherwise known only to a trusted entity.

https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Threats to security

A threat is a potential violation of security.

- The violation need not actually occur for there to be a threat

The fact that the violation **might occur** means that

- those actions that could cause it to occur must be guarded against (or prepared for)

Attacks: Those actions

Attackers: Those who execute such actions

Many threats fall into 4 main classes

1. **Disclosure,**
    - unauthorized access to information;
2. **Deception**
    - acceptance of false data
3. **Disruption**
    - interruption or prevention of correct operation
4. **Usurpation**
    - unauthorized control of some part of a system

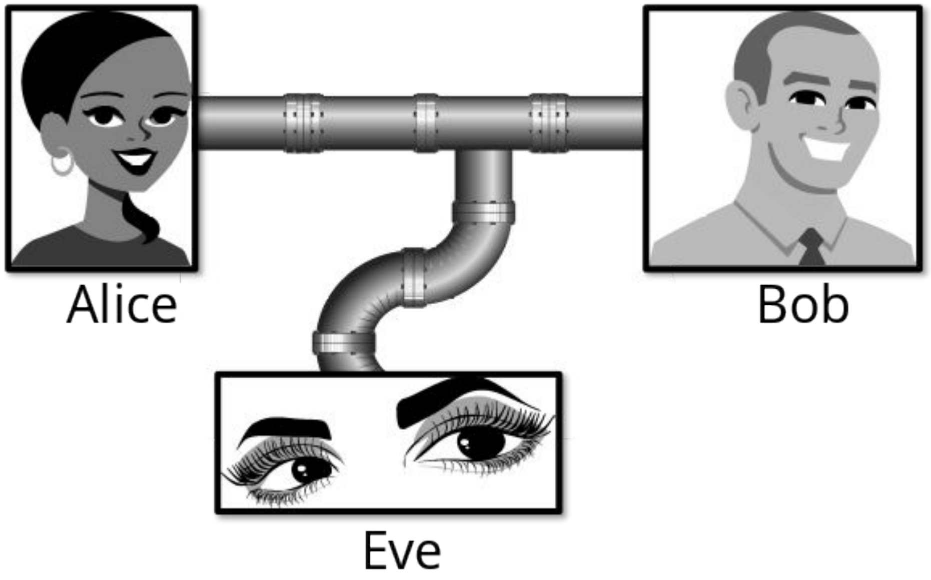
Threats:Snooping (Eavesdropping)

The unauthorized interception of information, is a form of disclosure

It is passive, suggesting simply that some entity is listening to (or reading) communications or browsing through files or system information

Wiretapping, or **passive wiretapping**, is a form of snooping in which a network is monitored

Confidentiality services counter this threat



https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Threats:Modification or alteration

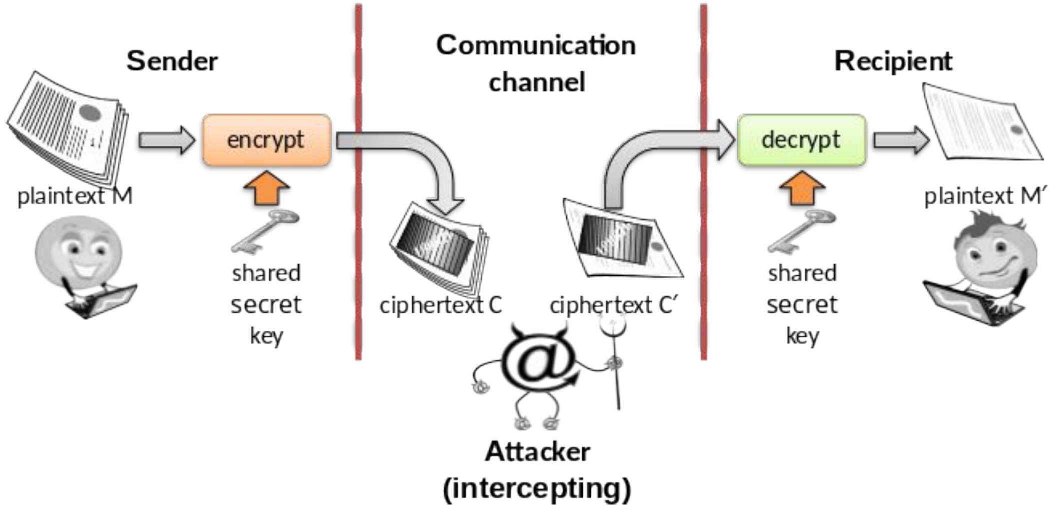
an unauthorized change of information,

- covers three classes of threat

The goal may be deception

Unlike snooping, modification is active

- **man-in-the-middle attack**: an intruder reads messages from the sender and sends (possibly modified) versions to the recipient
  - Integrity services counter this threat



https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Introduction to Computer Security : Bishop, Matt

Threats: Masquerading or spoofing

an impersonation of one entity by another,

- is a form of both deception and usurpation.

It lures a victim into believing that the entity with which it is communicating is a different entity

Delegation(an allowed form of masquerading): one entity authorizes a second entity to perform functions on its behalf

if a user tries to log into a computer across the Internet but instead reaches another computer that claims to be the desired one,

- **the user has been spoofed!**

"From: Alice" (really is from Eve)

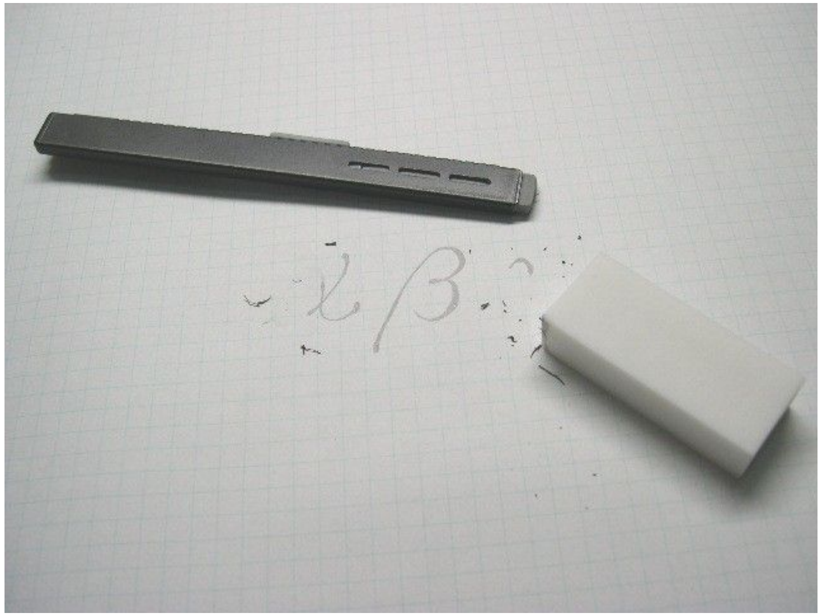
https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Introduction to Computer Security : Bishop, Matt

Threats: Repudiation of origin

a false denial that an entity sent (or created) something,

- a form of deception



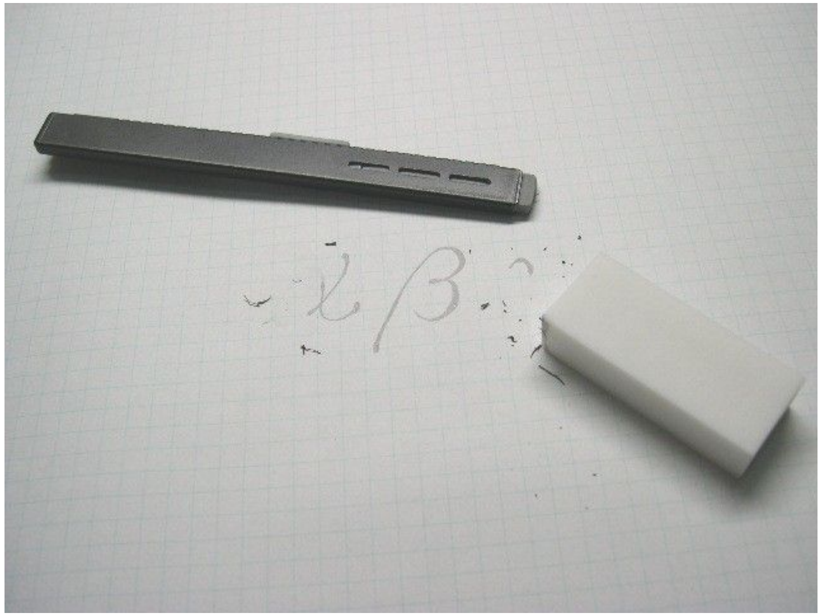
https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Introduction to Computer Security : Bishop, Matt

Threats: Denial of receipt

a false denial that an entity received some information or message,

- a form of deception.



https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Threats: Delay

a temporary inhibition of a service

- a form of usurpation
- can play a supporting role in deception.

Let’s say: delivery of a message or service requires some time t;

- **delay**: an attacker can force the delivery to take more than time t,

If an entity is waiting for an authorization message that is delayed,

- it may query a secondary server for the authorization.
  - The attacker may be unable to masquerade as the primary server,
    - may masquerade as that secondary server and supply incorrect information

Threats: Denial of service

a long-term inhibition of service

- a form of usurpation
- often used with other mechanisms to deceive

denial may occur

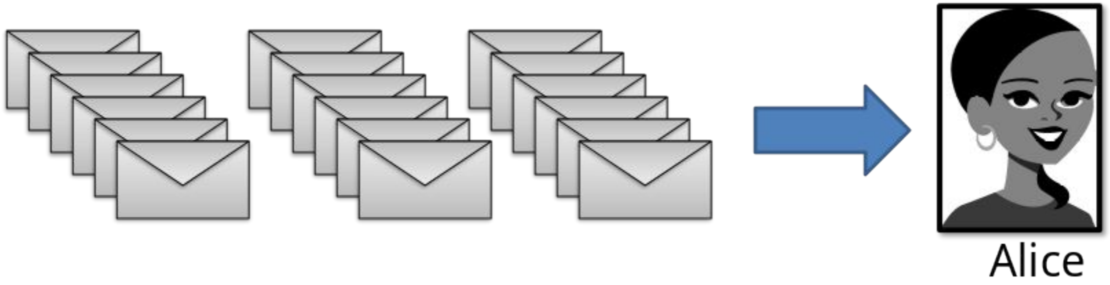
- at the source
  - preventing the server from obtaining the resources needed to perform its function
- at the destination
  - blocking the communications from the server
- or along the intermediate path
  - discarding messages from either the client or the server, or both

Availability mechanisms counter this threat.

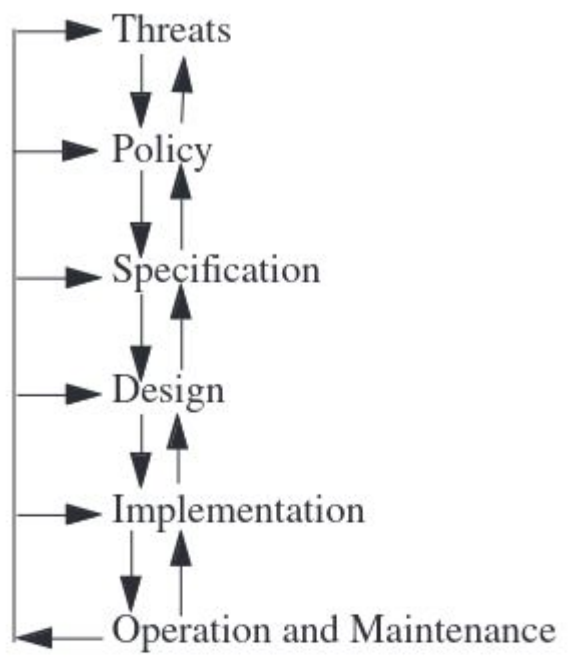
Denial of service or delay may result from **direct attacks** or from **non-security related** problems

- we view it as an attempt to breach system security
  - it compromises system security,
  - or is part of a sequence of events leading to the compromise of a system,

email spam to fill-up a mail queue



Security life cycle

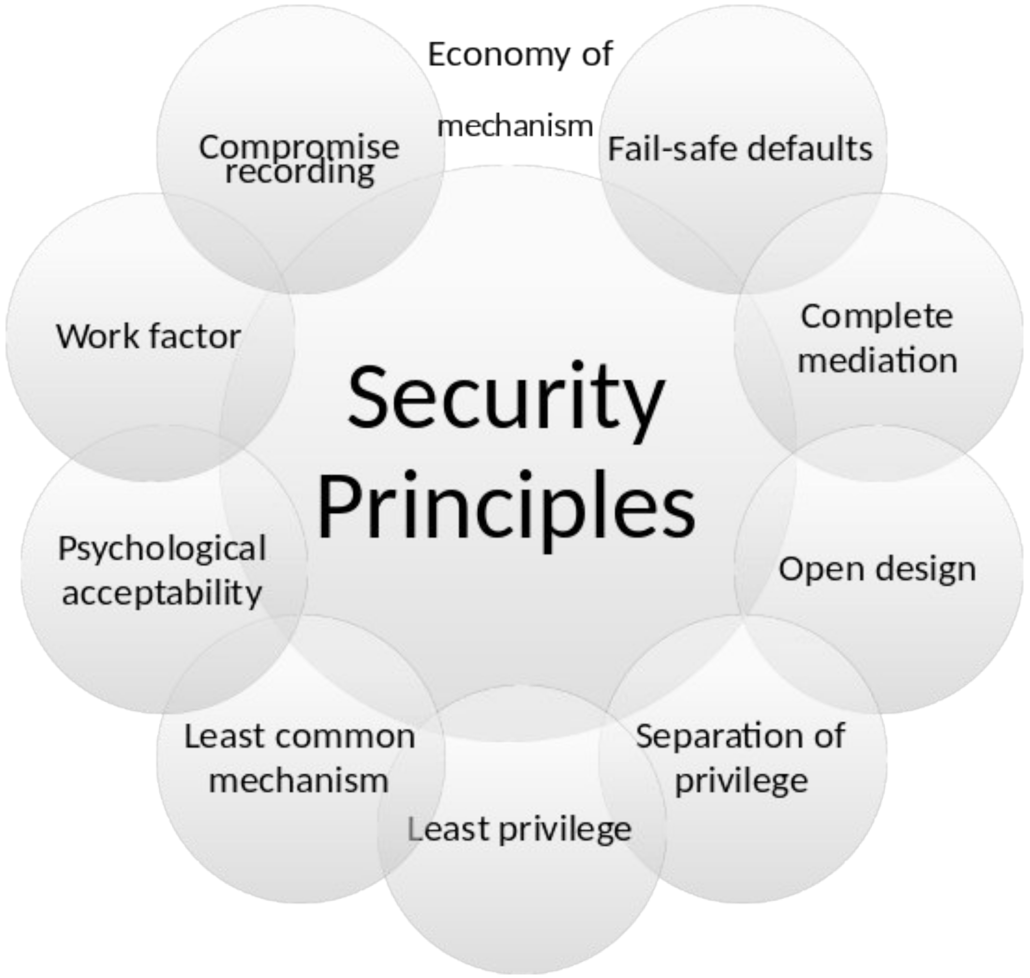


Human issues pervade each stage of the cycle and each cycle feeds info back

Example:A major corporation decided to improve its security.

- It hired consultants,
- determined the threats,
- and created a policy.
- From the policy, the consultants derived several specifications that the security mechanisms had to meet.
- They then developed a design that would meet the specifications
- During the implementation phase,
  - the company discovered that employees could connect modems to the telephones without being detected.
  - The design required all incoming connections to go through a firewall
  - **The design had to be modified to** divide systems into two classes:
    - systems connected to "the outside," which were put outside the firewall;
    - and all other systems, which were put behind the firewall.
  -
- **The operation and maintenance stage is critical to the life cycle**
  - The company discovers that several "trusted" hosts (those allowed to log in without authentication) were physically outside the control of the company.
    - **This violates policy!**

The Ten Security Principles



Introduction to Computer Security : Bishop, Matt

https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Access Control

Users and groups

Authentication

Passwords

File protection

Access control lists

Which users can read/write which files?

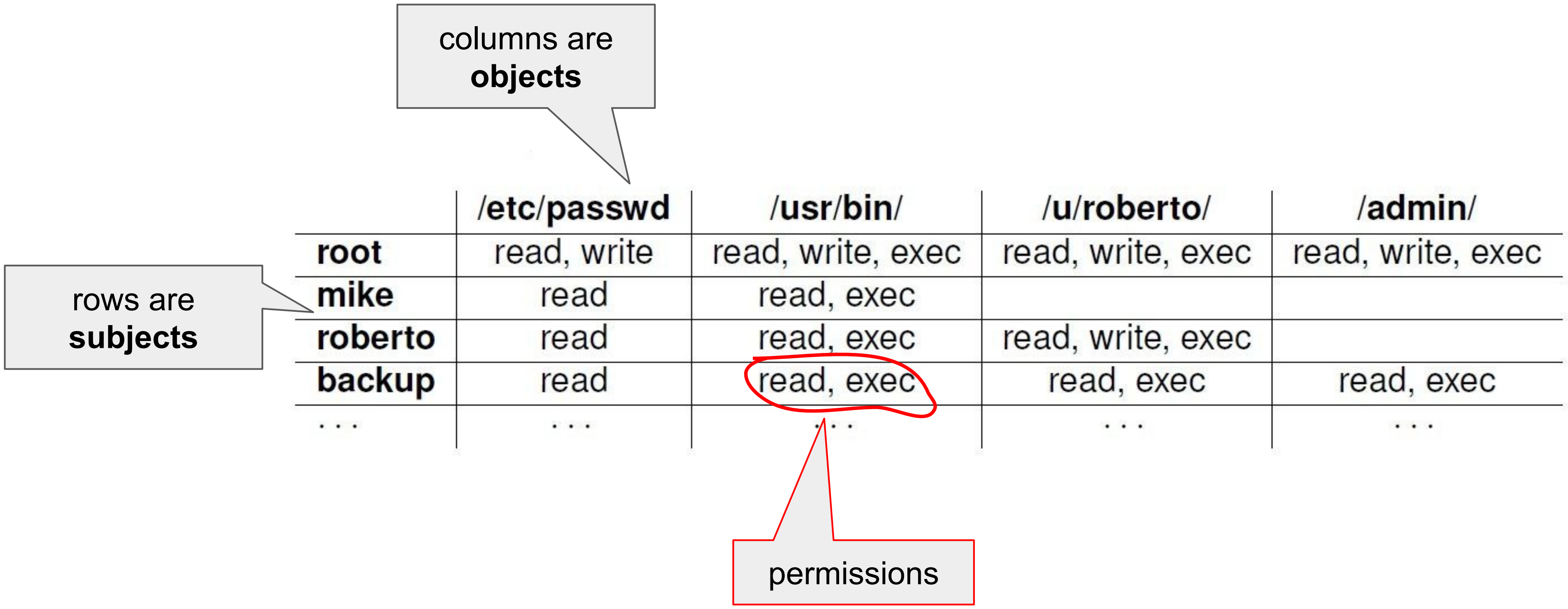
Are my files really safe?

What does it mean to be root?

What do we really want to control?

https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Access Control Matrices

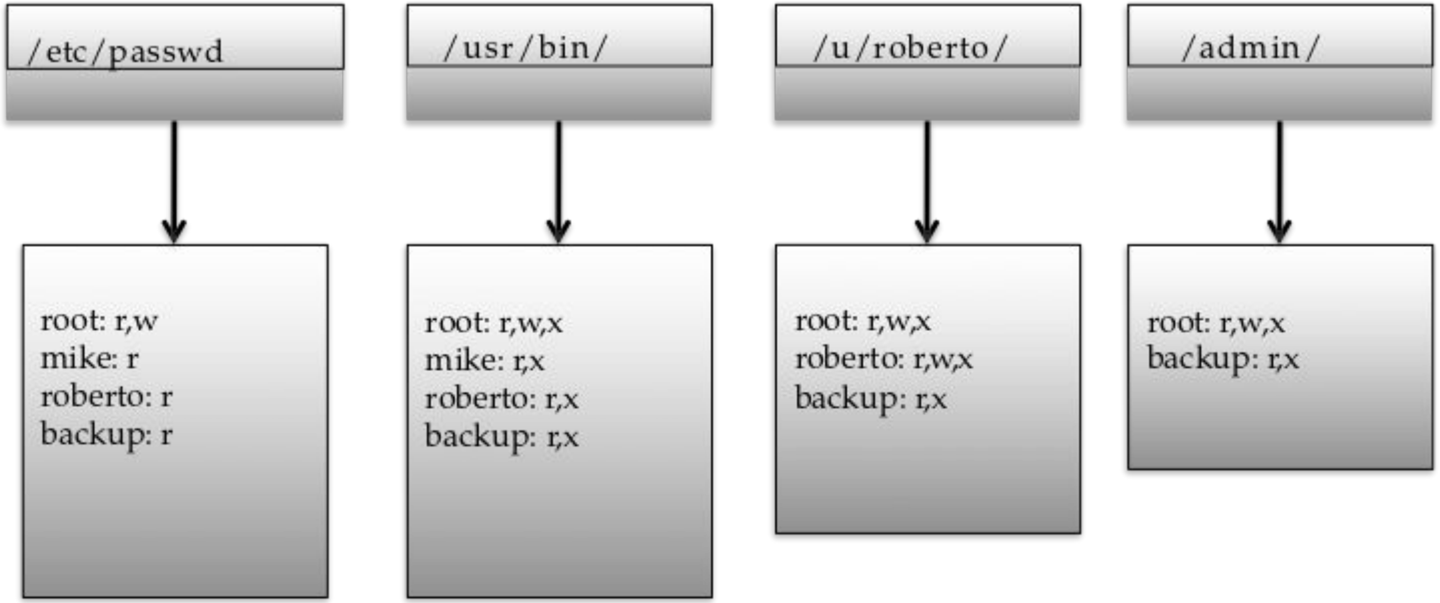

columns are **objects**
rows are **subjects**
permissions

	/etc/passwd	/usr/bin/	/u/roberto/	/admin/
root	read, write	read, write, exec	read, write, exec	read, write, exec
mike	read	read, exec		
roberto	read	read, exec	read, write, exec	
backup	read	read, exec	read, exec	read, exec
. . .	. . .	. . .	. . .	. . .

https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Access Control Lists

It defines, for each object, o, a list, L, called o's access control list, which enumerates all the subjects that have access rights for o and, for each such subject, s, gives the access rights that s has for object o.

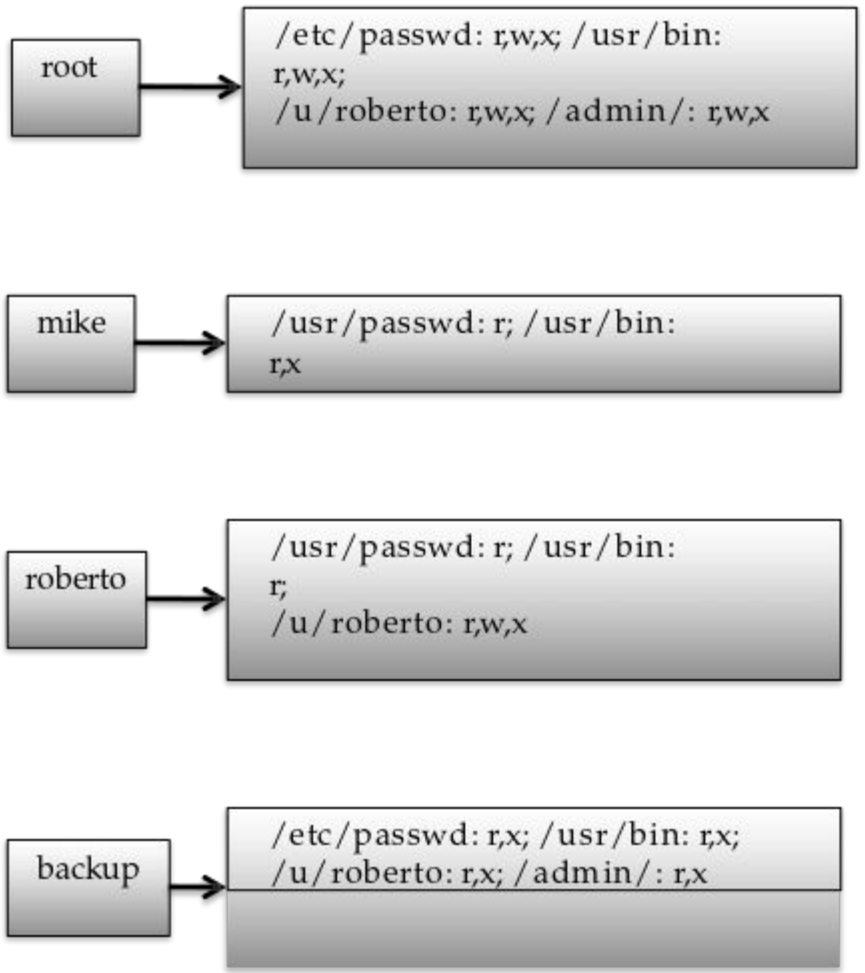


https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Capabilities

Takes a subject- centered approach to access control.

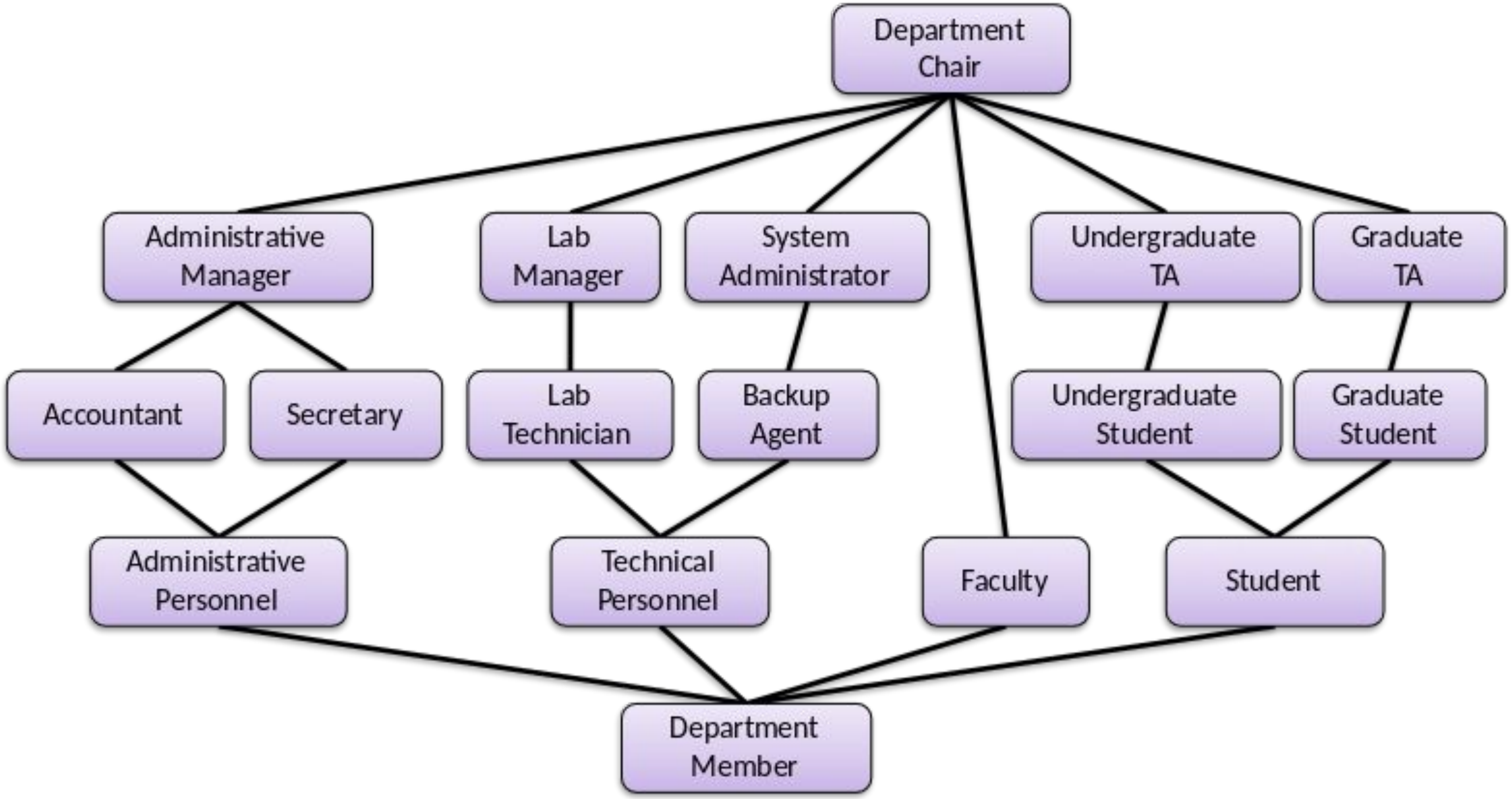
It defines, for each subject s, the list of the objects for which s has nonempty access control rights, together with the specific rights for each such object.


root
/etc/passwd: r,w,x; /usr/bin: r,w,x;
/u/roberto: r,w,x; /admin/: r,w,x
mike
/usr/passwd: r; /usr/bin: r,x
roberto
/usr/passwd: r; /usr/bin: r;
/u/roberto: r,w,x
backup
/etc/passwd: r,x; /usr/bin: r,x;
/u/roberto: r,x; /admin/: r,x

https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Role-based Access Control

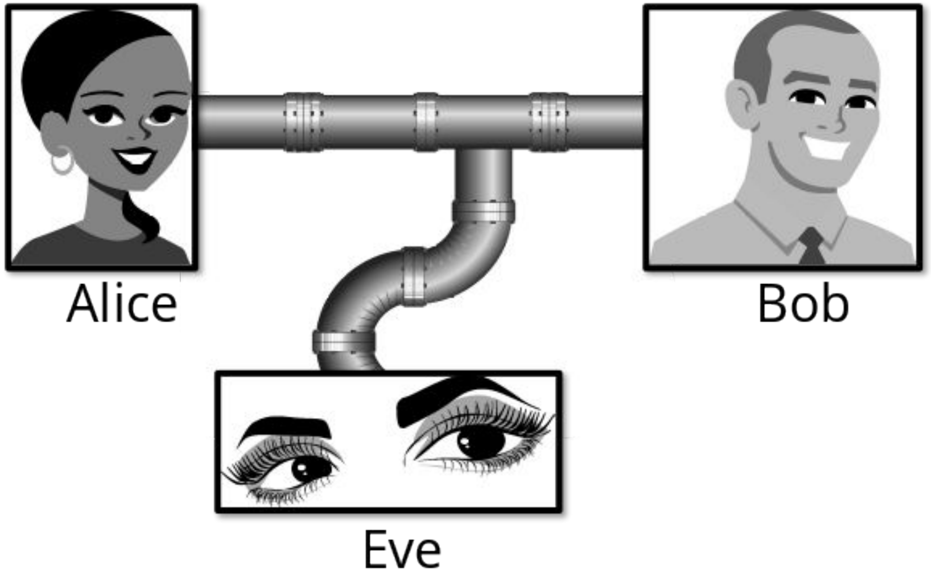
Define roles and then specify access control rights for these roles, rather than for subjects directly



https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Cryptographic concepts

Encryption: a means to allow two parties, customarily called Alice and Bob, to establish confidential communication over an insecure channel that is subject to eavesdropping.



https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Encryption and Decryption

The message M is called the **plaintext**.

Alice will convert **plaintext M** to an encrypted form

- Alice uses an **encryption algorithm E** that outputs **a ciphertext C** for M.

As equations:

C = E(M)

M = D(C)

The encryption and decryption algorithms are chosen so that it is infeasible for someone other than Alice and Bob to determine plaintext M from ciphertext C.

ciphertext C can be transmitted over an insecure channel that can be eavesdropped by an adversary.

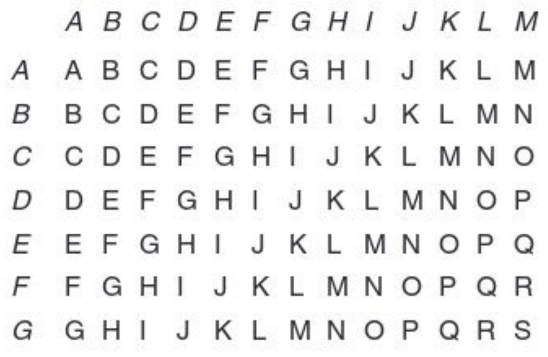
https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Classical cryptosystems (single-key or symmetric cryptosystems)

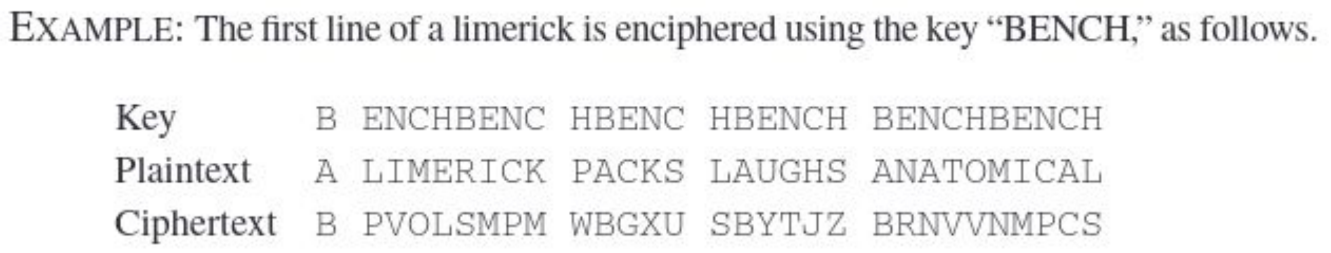
Substitution Ciphers

A substitution cipher changes characters in the plaintext to produce the ciphertext.

- **Transposition ciphers**
- **Cesar cipher**
- **Vigenère cipher**
    - **A longer key, uses tableau**



	A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*
A	A	B	C	D	E	F	G	H	I	J	K	L	M
B	B	C	D	E	F	G	H	I	J	K	L	M	N
C	C	D	E	F	G	H	I	J	K	L	M	N	O
D	D	E	F	G	H	I	J	K	L	M	N	O	P
E	E	F	G	H	I	J	K	L	M	N	O	P	Q
F	F	G	H	I	J	K	L	M	N	O	P	Q	R
G	G	H	I	J	K	L	M	N	O	P	Q	R	S



EXAMPLE: The first line of a limerick is enciphered using the key "BENCH," as follows.

Key	B	ENCHBENC	HBENC	HBENCH	BENCHBENCH
Plaintext	A	LIMERICK	PACKS	LAUGHS	ANATOMICAL
Ciphertext	B	PVOLSMPM	WBGXU	SBYTJZ	BRNVVNMPCS

One-Time Pad: if the key as long as the text

Introduction to Computer Security : Bishop, Matt

Public Key Cryptography

Two keys: encipherment and decipherment keys

Public (encipherment) **key** is public!

Private (decipherment) **key** is know only to owner!

RSA

$$c = m^e \bmod n$$

$$m = c^d \bmod n$$

- **φ(n)**
  - **the number of numbers less than n with no factors in common with n**
- **Choose e: e < n relatively prime to φ(n).**
- **Find d: ed mod φ(n) = 1**

The public key is (e, n)

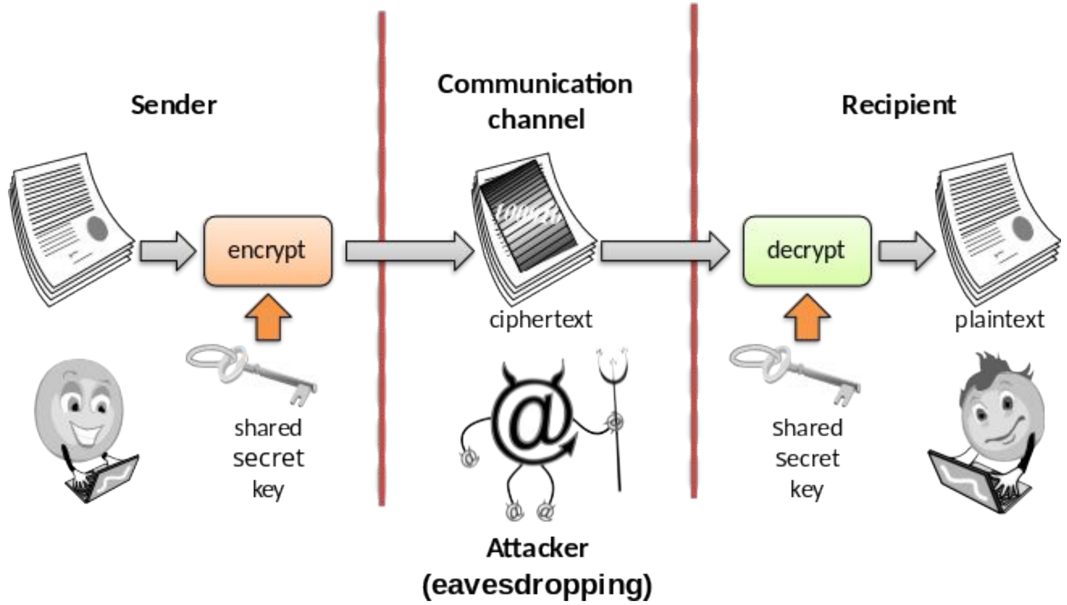
the private key is d

n = pq, p and q primes

In addition to confidentiality, RSA can provide data and origin authentication.

Symmetric cryptosystems

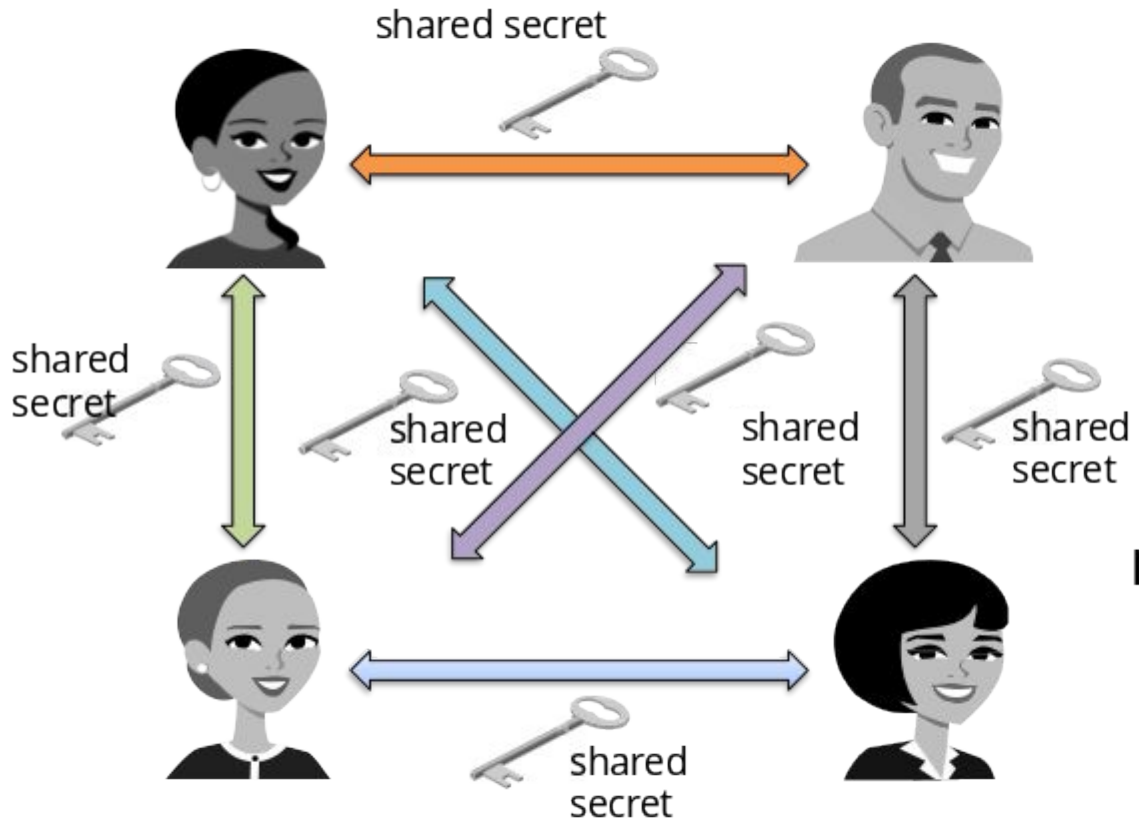
Alice and Bob share a secret key, which is used for both encryption and decryption.



https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Symmetric key Distribution

Requires each pair of communicating parties to share a (separate) secret key.

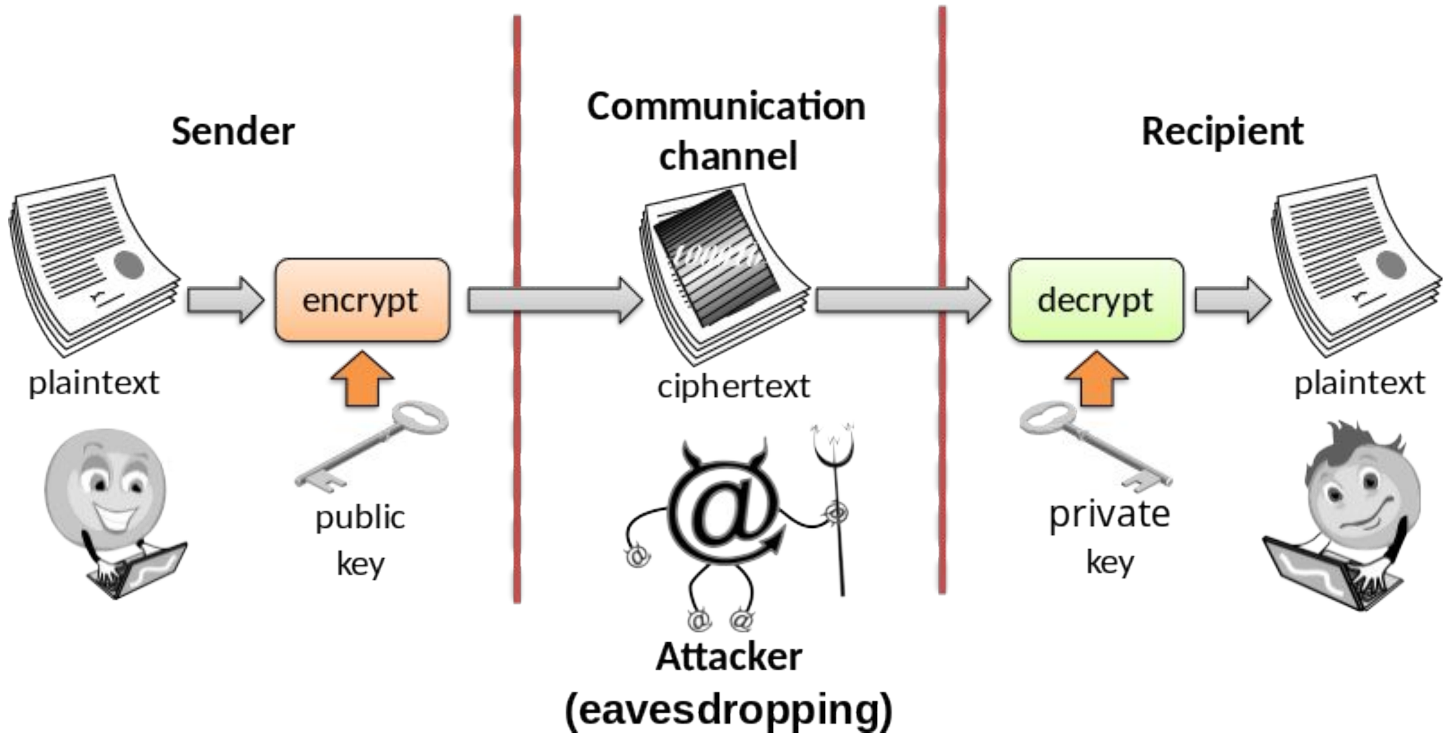


n (n-1)/2 keys

https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Public-Key Cryptography

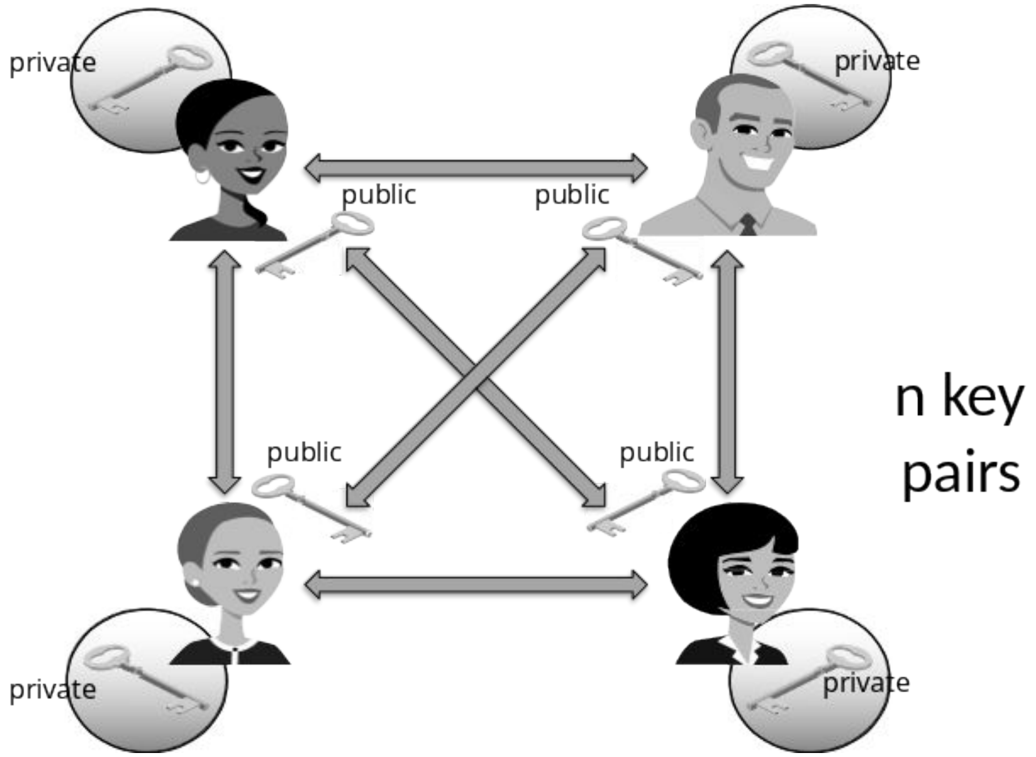
Separate keys are used for encryption and decryption.



https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Public Distribution

Only one key is needed for each recipient



Digital Signatures

Public-key encryption provides a method for doing digital signatures

To sign a message, M,

- Alice just encrypts it with her private key, SA, creating **$C = E_{SA}(M)$.**

Anyone can decrypt this message using Alice's public key,

- $M' = D_{PA}(C)$, and compare that to the message M.

Cryptographic Hash Functions

A checksum on a message, M, that is:

One-way:

- it should be easy to compute Y=H(M),
- but hard to find M given only Y

Collision-resistant:

- it should be hard to find two messages, M and N, such that H(M)=H(N).

Examples: SHA-1, SHA-256.

https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

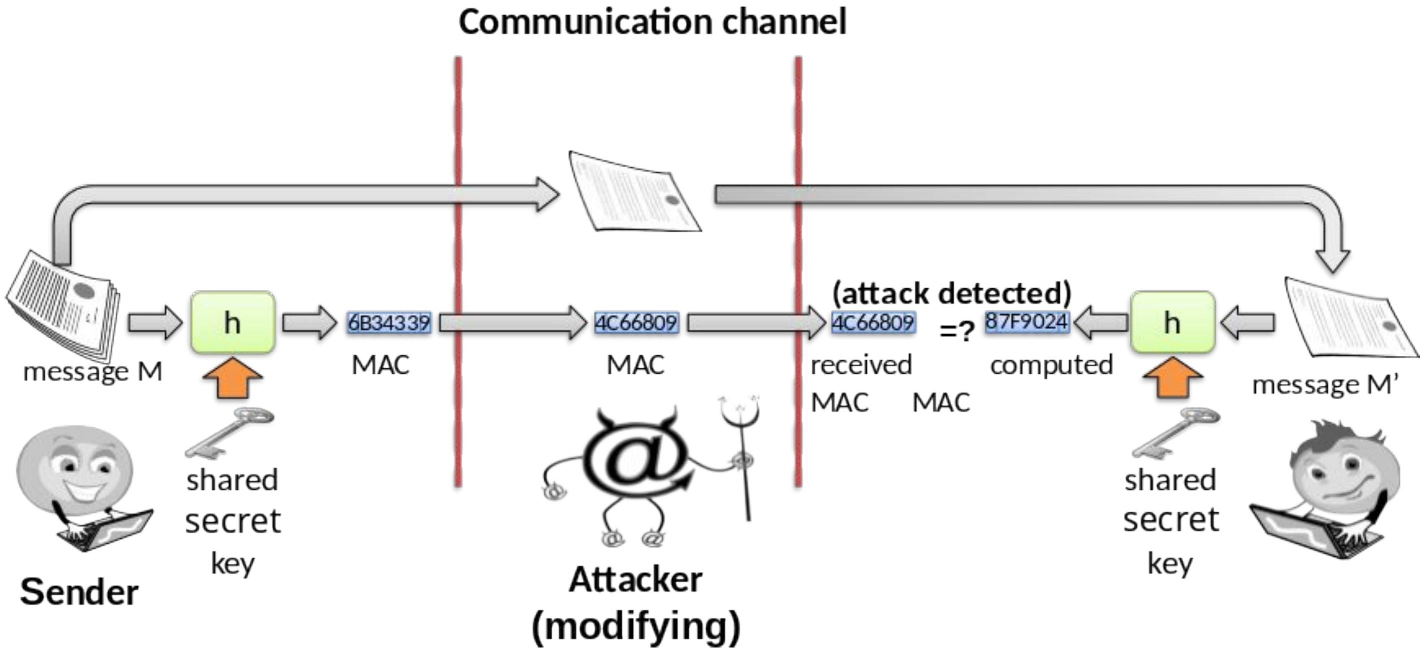
Message Authentication Codes

Allows for Alice and Bob to have data integrity, if they share a secret key.

Given a message M,

Alice computes H(K||M)

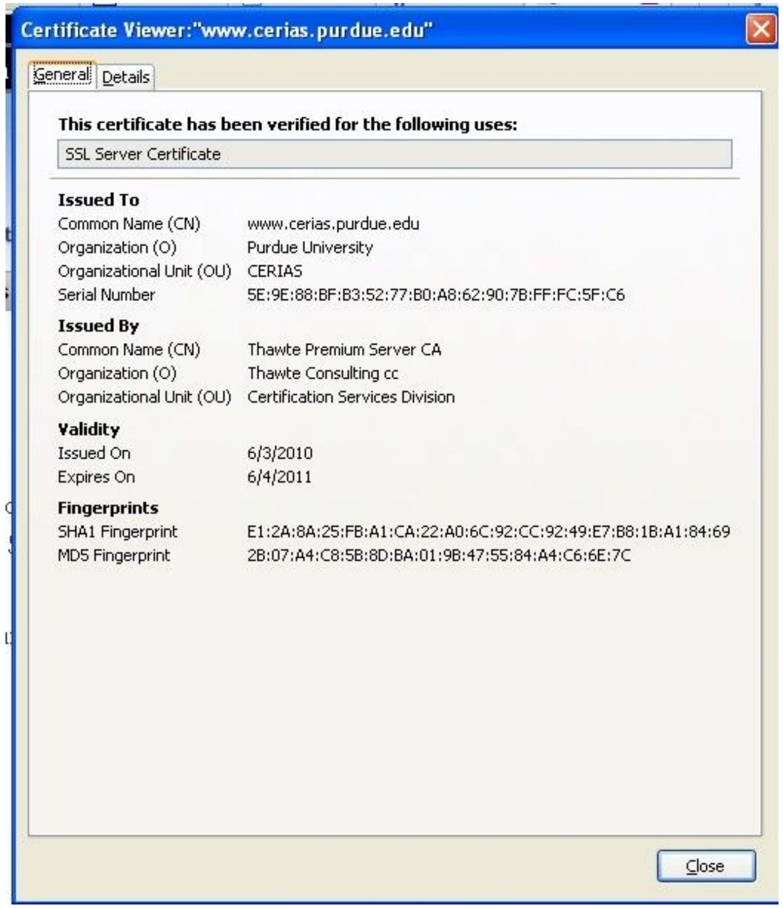
and sends M and this hash to Bob.



https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Digital Certificates

certificate authority (CA) digitally signs a binding between an identity and the public key for that identity.



This certificate has been verified for the following uses:

SSL Server Certificate

Issued To

Common Name (CN)	www.cerias.purdue.edu
Organization (O)	Purdue University
Organizational Unit (OU)	CERIAS
Serial Number	5E:9E:88:BF:B3:52:77:B0:A8:62:90:7B:FF:FC:5F:C6

Issued By

Common Name (CN)	Thawte Premium Server CA
Organization (O)	Thawte Consulting cc
Organizational Unit (OU)	Certification Services Division

Validity

Issued On	6/3/2010
Expires On	6/4/2011

Fingerprints

SHA1 Fingerprint	E1:2A:8A:25:FB:A1:CA:22:A0:6C:92:CC:92:49:E7:B8:1B:A1:84:69
MD5 Fingerprint	2B:07:A4:C8:5B:8D:BA:01:9B:47:55:84:A4:C6:6E:7C

https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Passwords

A short sequence of characters used as a means to authenticate someone via a secret that they know.

Userid:

Password:

What is a strong password

- UPPER/lower case characters
- Special characters
- Numbers

When is a password strong?

Seattle1

M1ke03

P@$$w0rd

TD2k5secV

https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Password complexity

Odd characters make pass strong

A fixed 6 symbols password:

- Numbers
  - 10^6 = 1,000,000
- UPPER or lower case characters
  - 26^6 = 308,915,776
- UPPER and lower case characters
  - 52^6 = 19,770,609,664
- 32 special characters
  - (&, %, $, £, “, |, ^, §, etc.)
  - 32^6 = 1,073,741,824

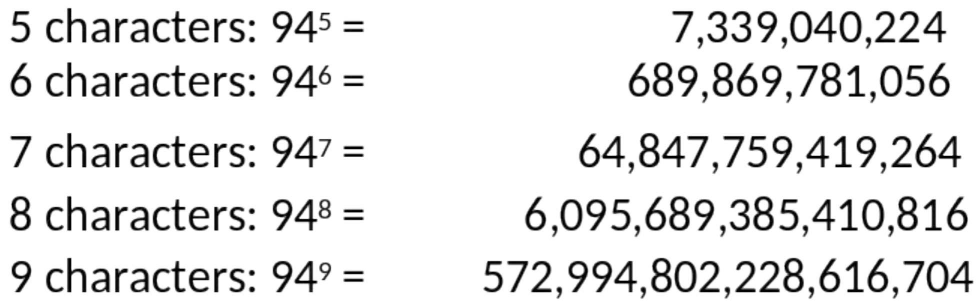
94 practical symbols available

- 94^6 = 689,869,781,056

ASCII standard 7 bit 2^7 =128 symbols

- 128^6 = 4,398,046,511,104

Password length



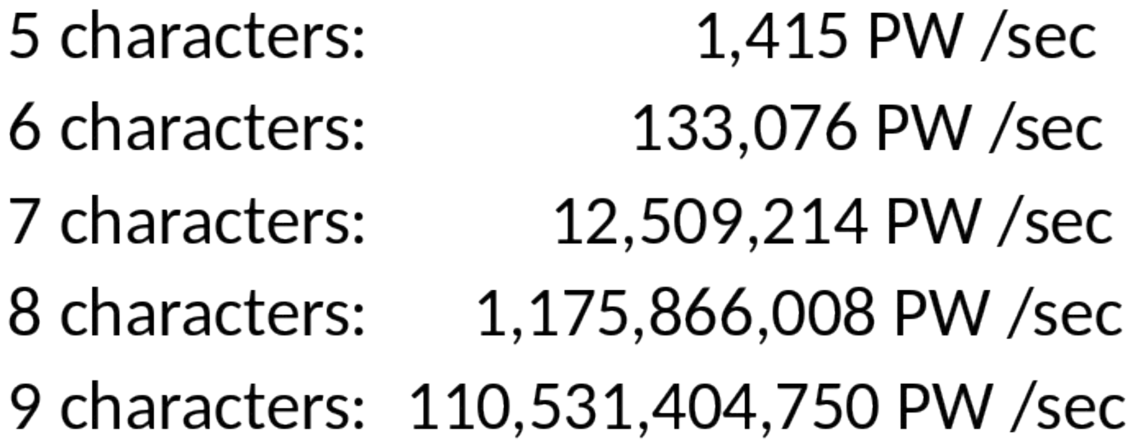
5 characters: 94^5 =	7,339,040,224
6 characters: 94^6 =	689,869,781,056
7 characters: 94^7 =	64,847,759,419,264
8 characters: 94^8 =	6,095,689,385,410,816
9 characters: 94^9 =	572,994,802,228,616,704

https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Password Validity: Brute Force Test

Password does not change for 60 days

how many passwords should I try for each second?

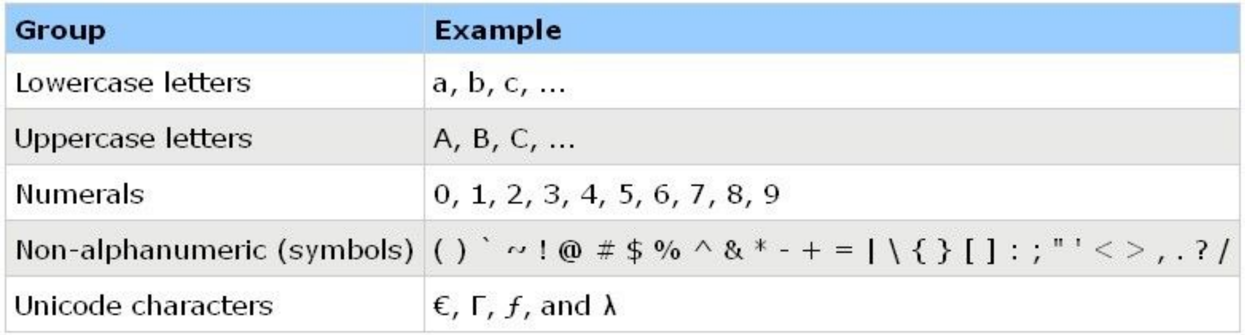


5 characters:	1,415 PW /sec
6 characters:	133,076 PW /sec
7 characters:	12,509,214 PW /sec
8 characters:	1,175,866,008 PW /sec
9 characters:	110,531,404,750 PW /sec

https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

Secure Passwords

A strong password includes characters from at least three of the following groups:

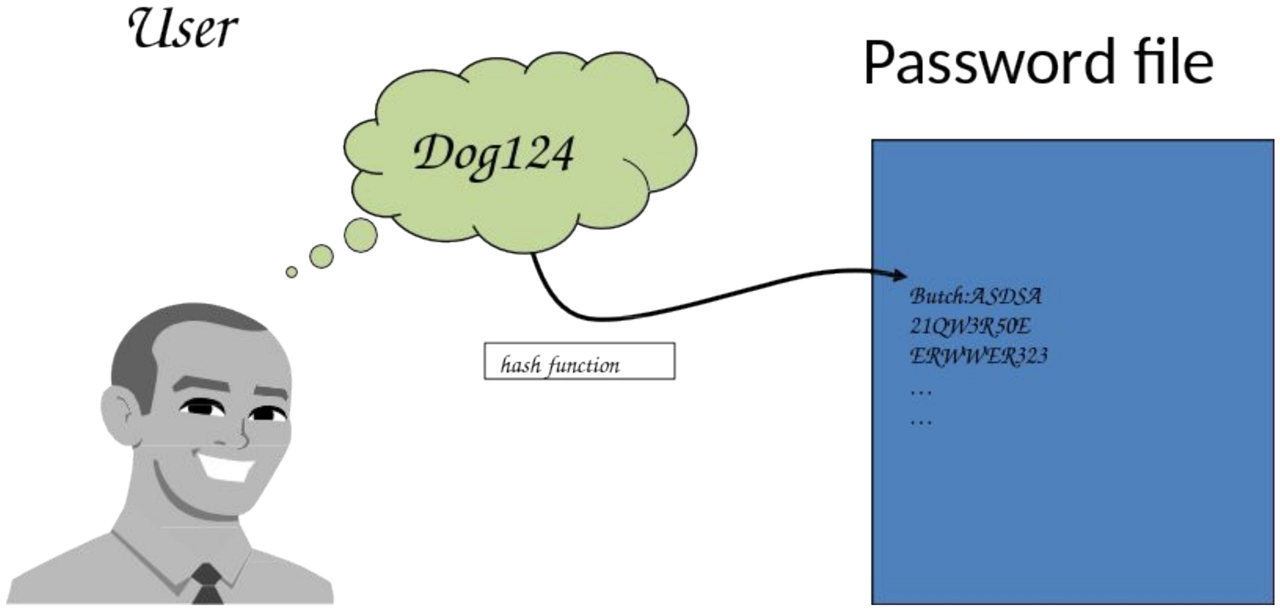


Group	Example
Lowercase letters	a, b, c, ...
Uppercase letters	A, B, C, ...
Numerals	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Non-alphanumeric (symbols)	( ) ` ~ ! @ # $ % ^ & * - + = | \ { } [ ] : ; " ' < > , . ? /
Unicode characters	€, Γ, ƒ, and λ

eg. "I re@lly want to buy 11 Dogs!"

https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

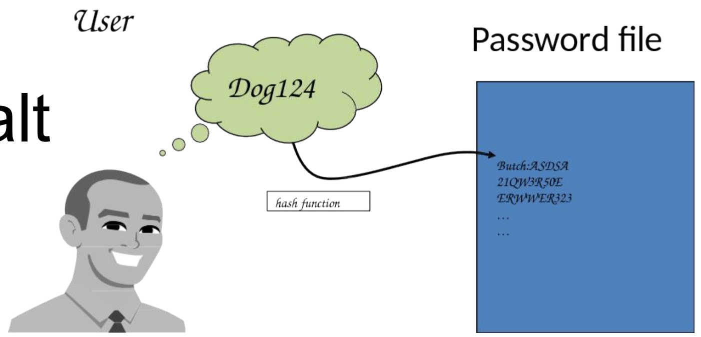
How a password is stored?



- ★ Add “salt” to avoid the same password being encrypted to the same value
  - H(dog124+salt)
  - Save salt+ h(dog124+salt)

https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

How a password is stored: random salt



Username	String to be hashed	Hashed value = SHA256
user1	**password123**	EF92B778BAFE771E89245B89ECBC08A44A4E166C06659911881F383D4473E94F
user2	**password123**	EF92B778BAFE771E89245B89ECBC08A44A4E166C06659911881F383D4473E94F

Username	Salt value	String to be hashed	Hashed value = SHA256 (Password + Salt value)
user1	D;%yL9TS:5PalS/d	**password123**D;%yL9TS:5PalS/d	9C9B913EB1B6254F4737CE947EFD16F16E916F9D6EE5C1102A2002E48D4C88BD
user2	)<,-<U(jLezy4j>*	**password123**)<,-<U(jLezy4j>*	6058B4EB46BD6487298B59440EC8E70EAE482239FF2B4E7CA69950DFBD5532F2

https://en.wikipedia.org/wiki/Salt_(cryptography)

https://ics.uci.edu/~goodrich/teach/cs201P/notes/Ch01-Introduction.pdf

OS protection

book slides for chapter 17 edited

- A **protection system** describes the conditions under which a system is secure.
- In one protection model, computer consists of a collection of objects, hardware or software
- Each object has a unique name and can be accessed through a well-defined set of operations
- **Protection Goal -** ensure that each object is accessed correctly and only by those processes that are allowed to do so

Principles of Protection

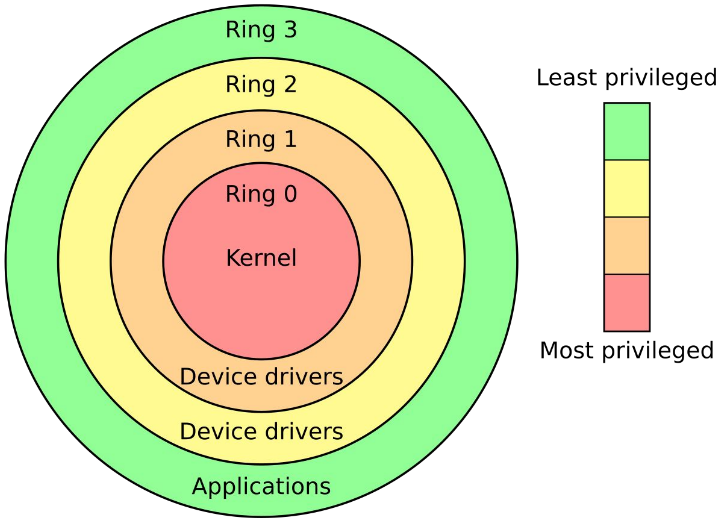
- Guiding principle – **principle of least privilege**
  - Programs, users and systems should be given just enough **privileges** to perform their tasks
  - Properly set **permissions** can limit damage if entity has a bug, gets abused
  - Can be static (during life of system, during life of process)
  - Or dynamic (changed by process as needed) – **domain switching, privilege escalation**
  - **Compartmentalization** a derivative concept regarding access to data
    - Process of protecting each individual system component through the use of specific permissions and access restrictions

Principles of Protection (Cont.)

- Must consider “grain” aspect
  - Rough-grained privilege management easier, simpler, but least privilege now done in large chunks
    - For example, traditional Unix processes either have abilities of the associated user, or of root
  - Fine-grained management more complex, more overhead, but more protective
    - File ACL lists, RBAC
- Domain can be user, process, procedure
- **Audit trail** – recording all protection-orientated activities, important to understanding what happened, why, and catching things that shouldn’t
- No single principle is a panacea for security vulnerabilities – need **defense in depth**

Protection Rings

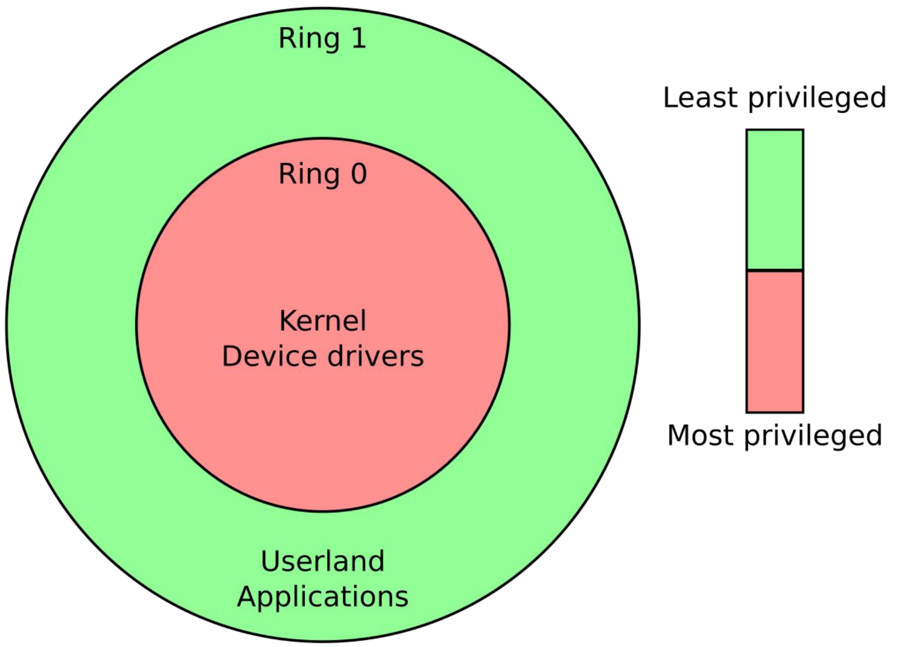
- introduced by the Multics operating system
- hierarchical protection domains
  - This privilege separation **requires hardware support**
  - Many modern CPU architectures (including Intel x86 architecture) include some form of ring protection
    - although most OSes do not fully utilize this feature.
  - Also traps and interrupts
  - **Hypervisors** introduced the need for yet another ring
  - ARMv7 processors added **TrustZone**(**TZ**) ring to protect crypto functions with access via new **Secure Monitor Call** (**SMC**) instruction
    - Protecting NFC secure element and crypto keys from even the kernel



https://en.wikipedia.org/wiki/Protection_ring

X86

- x86 has 4 protection rings,
- it is more common for architectures to only have two.
- x86-64 still uses the same 2-bit (4 level) privilege level mechanism
- Even on x86, most OS only use ring 0 and 3.



https://en.wikipedia.org/wiki/Protection_ring

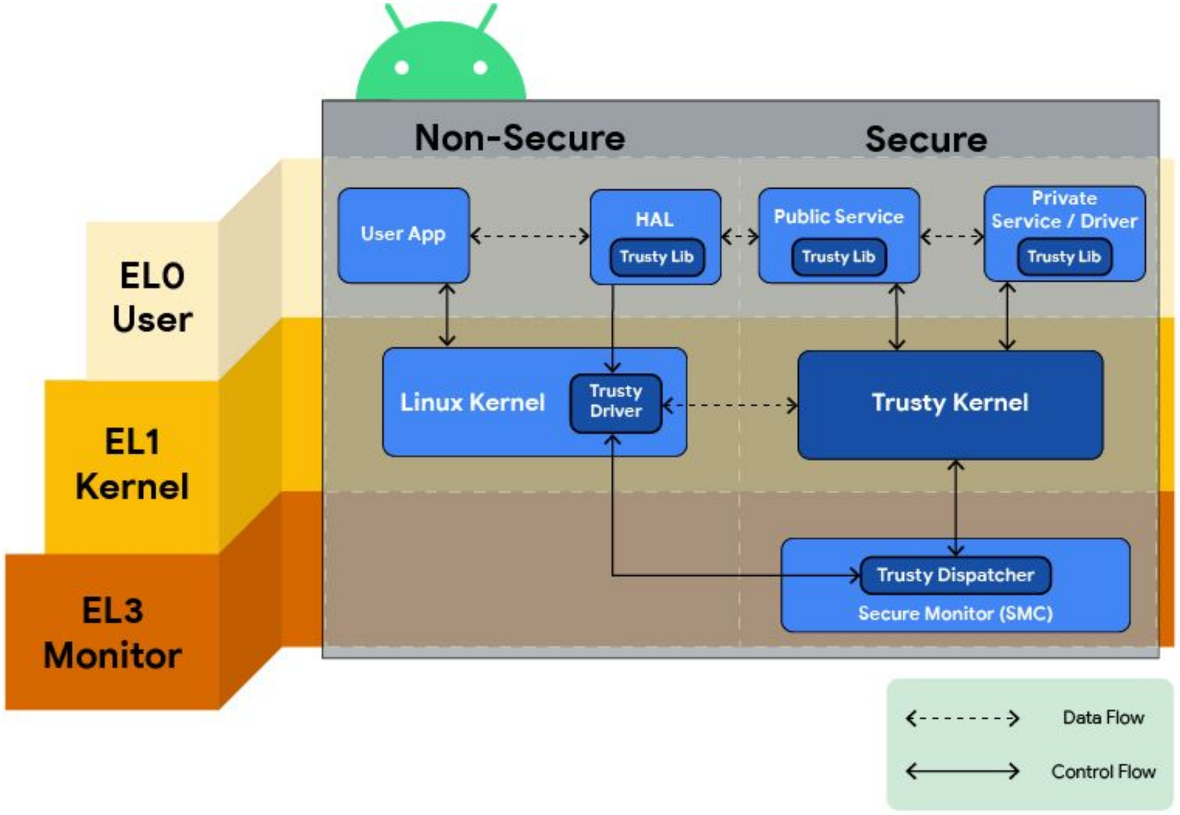
Example: Android Trusty: A Trusted OS for Android devices

Trusty is a secure Operating System (OS) that provides a Trusted Execution Environment (TEE) for Android.

The Trusty OS runs on the same processor as the Android OS, but Trusty is isolated from the rest of the system by both hardware and software.

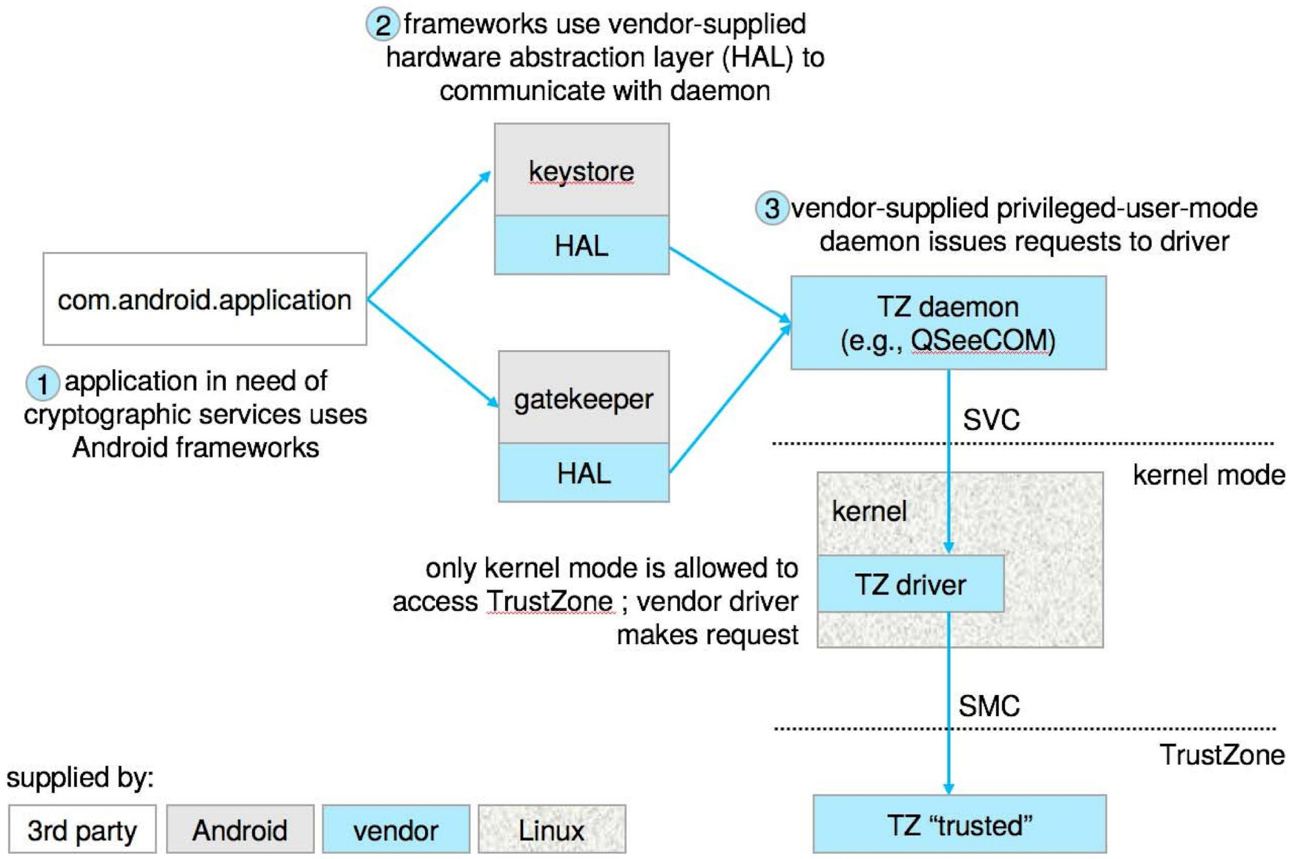
Trusty and Android run parallel to each other.

Trusty works upon the hardware isolation provided by **Arm TrustZone** to create software-level isolation – completing the TEE Trusted Execution Environment on Android.

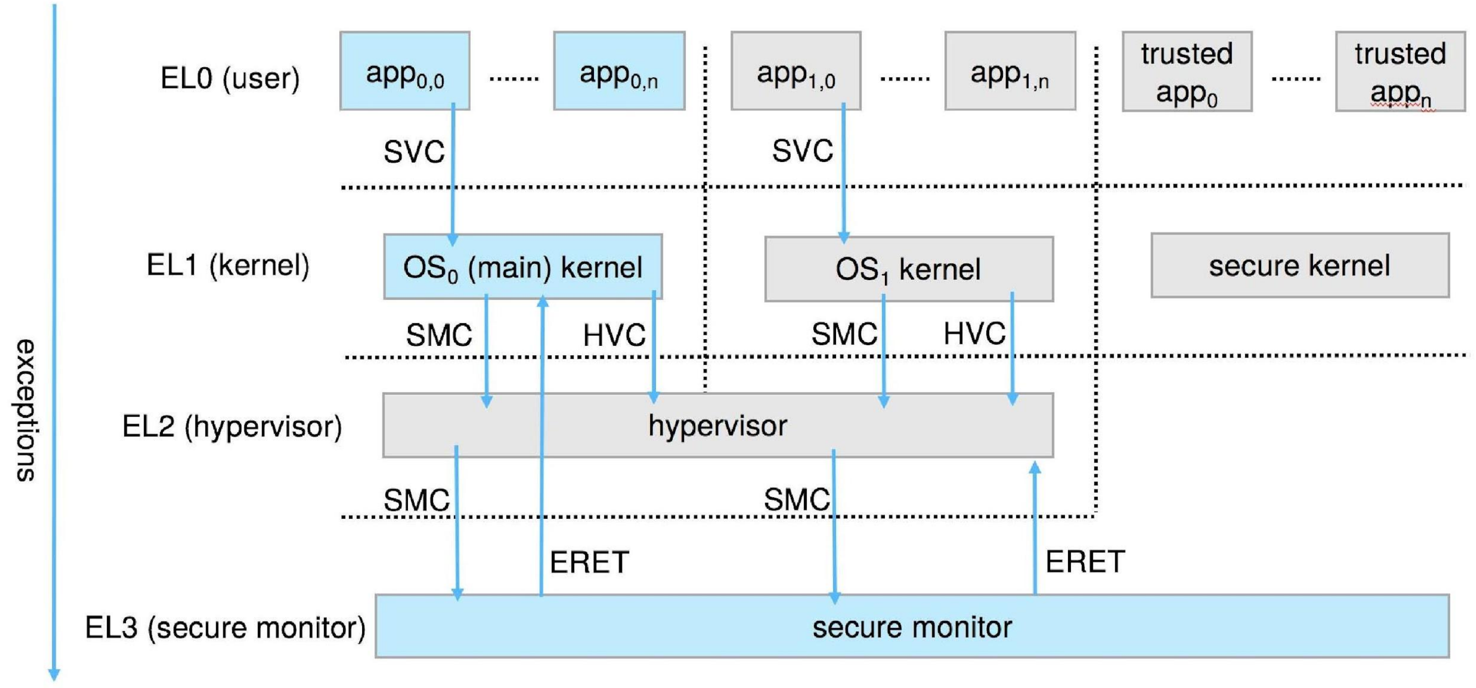


Trusty TEE | Android Open Source Project
Android TEE: The vault of Android security

Android TrustZone: Implementing TEE through hardware


2 frameworks use vendor-supplied hardware abstraction layer (HAL) to communicate with daemon
keystore
HAL
3 vendor-supplied privileged-user-mode daemon issues requests to driver
com.android.application
TZ daemon (e.g., QSeeCOM)
1 application in need of cryptographic services uses Android frameworks
gatekeeper
HAL
SVC
kernel mode
kernel
only kernel mode is allowed to access TrustZone ; vendor driver makes request
TZ driver
SMC
TrustZone
supplied by:
3rd party
Android
vendor
Linux
TZ "trusted"

ARM CPU Architecture



Domain of Protection

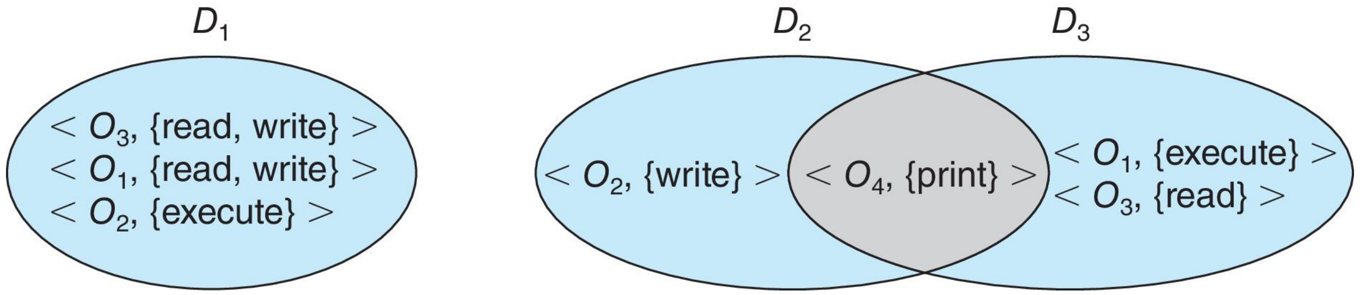
- Rings of protection separate functions into domains and order them hierarchically
- Computer can be treated as processes and objects
  - **Hardware objects** (such as devices) and **software objects** (such as files, programs, semaphores
- Process for example should only have access to objects it currently requires to complete its task – the **need-to-know** principle

Domain of Protection (Cont.)

- Implementation can be via process operating in a **protection domain**
  - Specifies resources process may access
  - Each domain specifies set of objects and types of operations on them
  - Ability to execute an operation on an object is an **access right**
    - <object-name, rights-set>
  - Domains may share access rights
  - Associations can be **static** or **dynamic**
  - If dynamic, processes can **domain switch**

Domain Structure

- Access-right = <*object-name*, *rights-set*>
  where *rights-set* is a subset of all valid operations that can be performed on the object
- Domain = set of access-rights



Domain Implementation (UNIX)

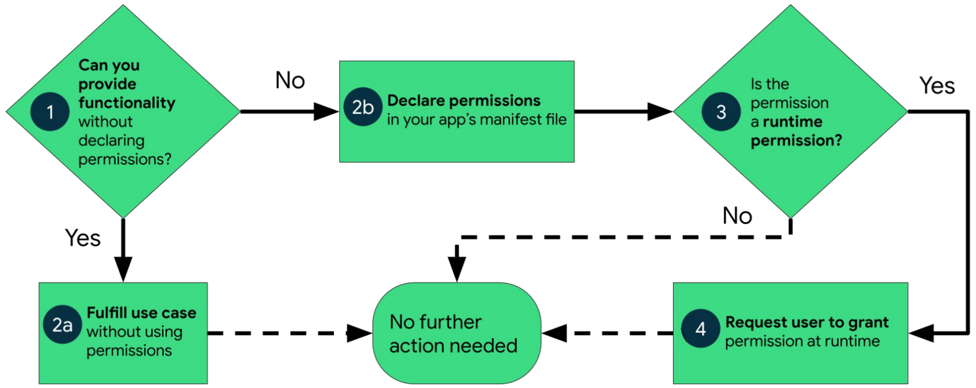
- Domain = user-id
- The Unix and Linux access rights flags **setuid** and **setgid** allow users to run an executable with the file system permissions of the executable's owner or group respectively and to change behaviour in directories.
- Domain switch accomplished via file system
  - Each file has associated with it a domain bit (setuid bit)
  - When file is executed and setuid = on, then user-id is set to owner of the file being executed
  - When execution completes user-id is reset
- Domain switch accomplished via passwords
  - `su` command temporarily switches to another user’s domain when other domain’s password provided
- Domain switching via commands
  - `sudo` command prefix executes specified command in another domain (if original domain has privilege or password given)

Domain Implementation (Android App IDs)

App permissions help support user privacy by protecting access to the following:

- **Restricted data**, such as system state and users' contact information
- **Restricted actions**, such as connecting to a paired device and recording audio

Work with user ID, FIDs and GUIDs

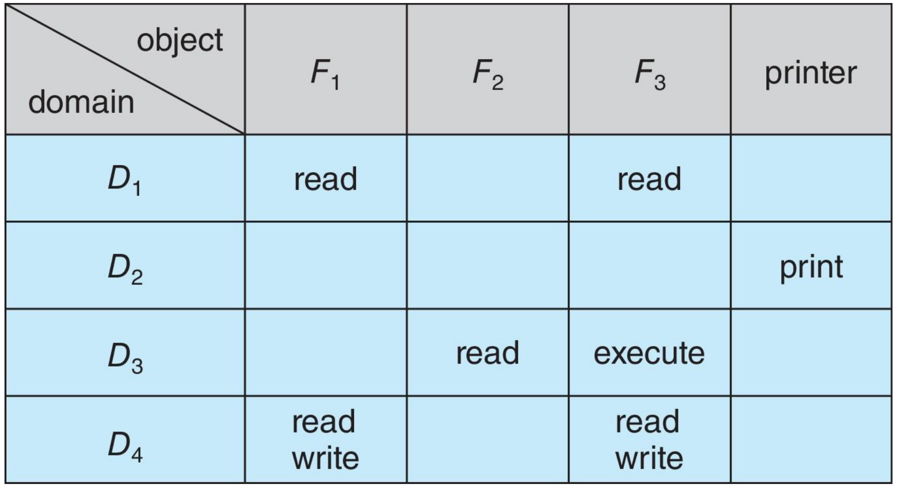

1 Can you provide functionality without declaring permissions?
No
Yes
2b Declare permissions in your app's manifest file
3 Is the permission a runtime permission?
Yes
No
2a Fulfill use case without using permissions
No further action needed
4 Request user to grant permission at runtime

```
<permission
        android:name="com.example.myapp.permission.DEADLY_ACTIVITY"
```

[Best practices for unique identifiers | Identity | Android Developers]
[Permissions on Android]
[Define a custom app permission | Android Developers]

Access Matrix

- View protection as a matrix (**access matrix**)
- Rows represent domains
- Columns represent objects
- **`Access(i, j)`** is the set of operations that a process executing in Domain$_i$ can invoke on Object$_j$



object domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Use of Access Matrix

- If a process in Domain *D_i* tries to do “op” on object *O_j*, then “op” must be in the access matrix
- User who creates object can define access column for that object
- Can be expanded to dynamic protection
  - Operations to add, delete access rights
  - Special access rights:
    - *owner of O_i*
    - *copy op from O_i to O_j (denoted by “*”)*
    - *control – D_i can modify D_j access rights*
    - *transfer – switch from domain D_i to D_j*
  - *Copy* and *Owner* applicable to an object
  - *Control* applicable to domain object

Use of Access Matrix (Cont.)

- **Access matrix** design separates mechanism from policy
  - Mechanism
    - Operating system provides access-matrix + rules
    - If ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced
  - Policy
    - User dictates policy
    - Who can access what object and in what mode
- But doesn’t solve the general confinement problem

Access Matrix of Figure A with Domains as Objects

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Access Matrix with *Copy* Rights

object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

Access Matrix With *Owner* Rights

object domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

object domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

Modified Access Matrix

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Implementation of Access Matrix

- Generally, a sparse matrix
- **Option 1 – Global table**
  - Store ordered triples **`<domain, object, rights-set>`** in table
  - A requested operation M on object O_j within domain D_i -> search table for < D_i, O_j, R_k >
    - with $M \in R_k$
  - But table could be large -> won't fit in main memory
  - Difficult to group objects (consider an object that all domains can read)

Implementation of Access Matrix (Cont.)

- **Option 2 – Access lists for objects**
  - Each column implemented as an access list for one object
  - Resulting per-object list consists of ordered pairs **`<domain, rights-set>`** defining all domains with non-empty set of access rights for the object
  - Easily extended to contain default set -> If M ∈ default set, also allow access

- Each column = Access-control list for one object
  Defines who can perform what operation

  Domain 1 = Read, Write
  Domain 2 = Read
  Domain 3 = Read

- Each Row = Capability List (like a key)
  For each domain, what operations allowed on what objects

  Object F1 – Read

  Object F4 – Read, Write, Execute

  Object F5 – Read, Write, Delete, Copy

Implementation of Access Matrix (Cont.)

- **Option 3 – Capability list for domains**
  - Instead of object-based, list is domain based
  - **Capability list** for domain is list of objects together with operations allows on them
  - Object represented by its name or address, called a **capability**
  - Execute operation M on object O_j, process requests operation and specifies capability as parameter
    - Possession of capability means access is allowed
  - Capability list associated with domain but never directly accessible by domain
    - Rather, protected object, maintained by OS and accessed indirectly
    - Like a “secure pointer”
    - Idea can be extended up to applications

Implementation of Access Matrix (Cont.)

- **Option 4 – Lock-key**
  - Compromise between access lists and capability lists
  - Each object has list of unique bit patterns, called **locks**
  - Each domain as list of unique bit patterns called **keys**
  - Process in a domain can only access object if domain has key that matches one of the locks

Comparison of Implementations

- Many trade-offs to consider
    - Global table is simple, but can be large
    - Access lists correspond to needs of users
        - Determining set of access rights for domain non-localized so difficult
        - Every access to an object must be checked
            - Many objects and access rights -> slow
    - Capability lists useful for localizing information for a given process
        - But revocation capabilities can be inefficient
    - Lock-key effective and flexible, keys can be passed freely from domain to domain, easy revocation

Comparison of Implementations (Cont.)

- Most systems use combination of access lists and capabilities
  - First access to an object -> access list searched
    - If allowed, capability created and attached to process
      - Additional accesses need not be checked
    - After last access, capability destroyed
    - Consider file system with ACLs per file

Revocation of Access Rights

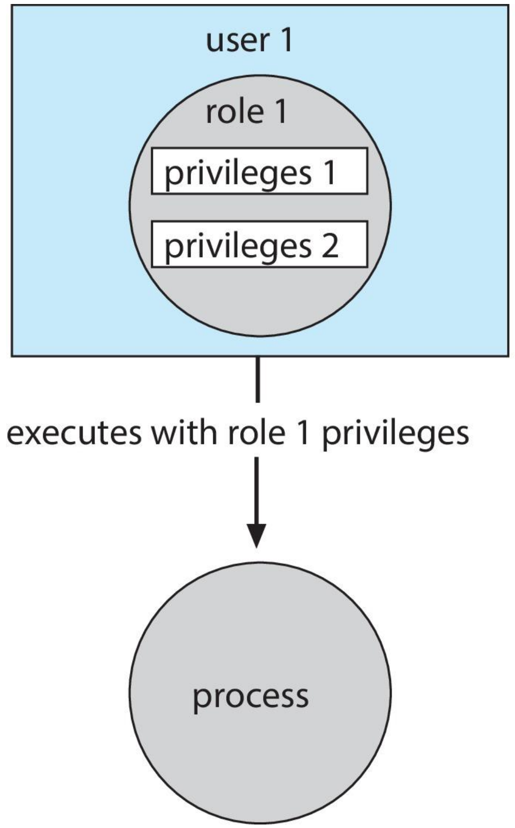
- Various options to remove the access right of a domain to an object
    - **Immediate vs. delayed**
    - **Selective vs. general**
    - **Partial vs. total**
    - **Temporary vs. permanent**
- **Access List** – Delete access rights from access list
    - **Simple** – search access list and remove entry
    - **Immediate**, **general** or **selective**, **total** or **partial**, **permanent** or **temporary**

Revocation of Access Rights (Cont.)

- **Capability List** – Scheme required to locate capability in the system before capability can be revoked
  - **Reacquisition** – periodic delete, with require and denial if revoked
  - **Back-pointers** – set of pointers from each object to all capabilities of that object (Multics)
  - **Indirection** – capability points to global table entry which points to object – delete entry from global table, not selective (CAL)
  - **Keys** – unique bits associated with capability, generated when capability created
    - Master key associated with object, key matches master key for access
    - Revocation – create new master key
    - Policy decision of who can create and modify keys – object owner or others?

Role-based Access Control

- Protection can be applied to non-file resources
- Oracle Solaris 10 provides **role-based access control** (**RBAC**) to implement least privilege
    - ***Privilege*** is right to execute system call or use an option within a system call
    - Can be assigned to processes
    - Users assigned ***roles*** granting access to privileges and programs
        - Enable role via password to gain its privileges
    - Similar to access matrix



Mandatory Access Control (MAC) vs DAC

- Operating systems traditionally had **discretionary access control (DAC)** to limit access to files and other objects (for example UNIX file permissions and Windows access control lists (ACLs))
  - Discretionary is a weakness – users / admins need to do something to increase protection

- Stronger form is mandatory access control, which even root user can't circumvent
  - Makes resources inaccessible except to their intended owners
  - Modern systems implement both MAC and DAC, with MAC usually a more secure, optional configuration (Trusted Solaris, TrustedBSD (used in macOS), SELinux), Windows Vista MAC)
- At its heart, labels assigned to objects and subjects (including processes)
  - When a subject requests access to an object, policy checked to determine whether or not a given label-holding subject is allowed to perform the action on the object

Capability-Based Systems

- Hydra and CAP were first capability-based systems
- Now included in Linux, Android and others, based on POSIX.1e (that never became a standard)
    - Essentially slices up root powers into distinct areas, each represented by a bitmap bit
    - Fine grain control over privileged operations can be achieved by setting or masking the bitmap
    - Three sets of bitmaps – permitted, effective, and inheritable
        - Can apply per process or per thread
        - Once revoked, cannot be reacquired
        - Process or thread starts with all privs, voluntarily decreases set during execution
        - Essentially a direct implementation of the principle of least privilege
- An improvement over root having all privileges but inflexible (adding new privilege difficult, etc.)

Capability example

`/etc/passwd`

- this identifies a unique object on the system, it does not specify access rights and hence is not a capability.
-

`/etc/passwd`

`O_RDWR`

- This pair identifies an object along with a set of access rights.
- The pair, however, is **still not a capability** because the user process's *possession* of these values says nothing about whether that access would actually be legitimate.

```
int fd = open("/etc/passwd",
O_RDWR);
```

- The variable `fd` now contains the index of a file descriptor in the process's file descriptor table.
- This file descriptor *is* a capability.

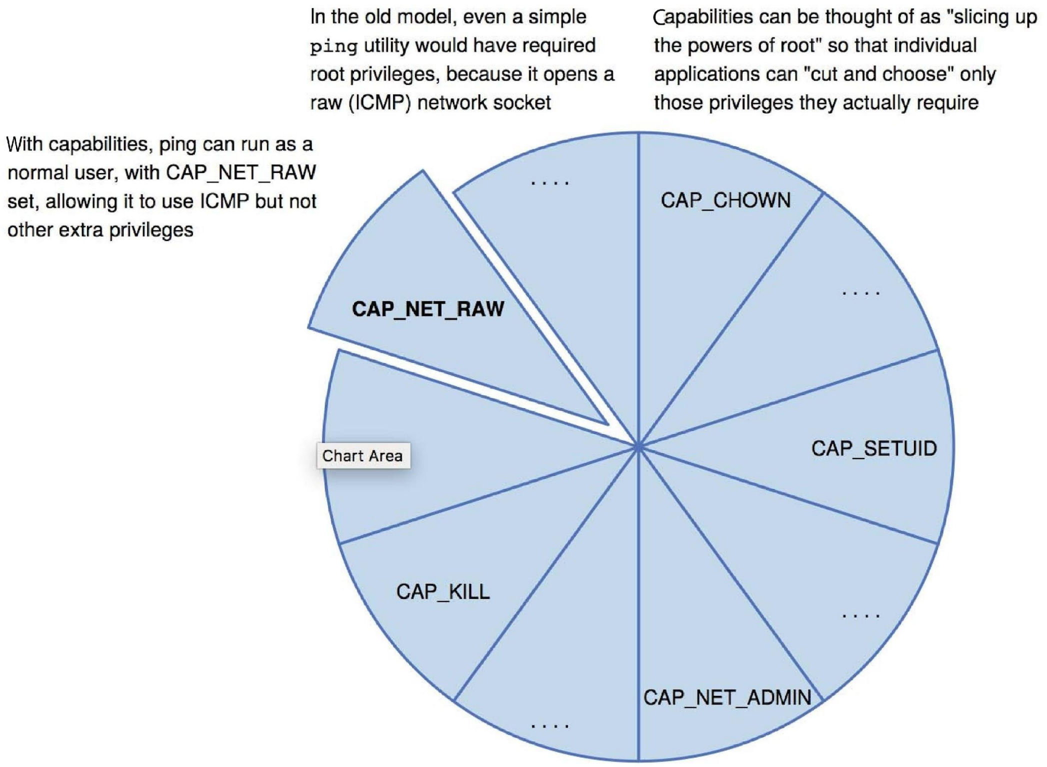
https://en.wikipedia.org/wiki/Capability-based_security

Capabilities in POSIX.1e

A POSIX capability is not associated with any object;

a process having CAP_NET_BIND_SERVICE capability can listen on any TCP port under 1024.

This system is found in Linux.
[capabilities(7) - Linux manual page]


In the old model, even a simple ping utility would have required root privileges, because it opens a raw (ICMP) network socket
Capabilities can be thought of as "slicing up the powers of root" so that individual applications can "cut and choose" only those privileges they actually require
With capabilities, ping can run as a normal user, with CAP_NET_RAW set, allowing it to use ICMP but not other extra privileges
. . . .
CAP_CHOWN
. . . .
CAP_NET_RAW
CAP_SETUID
Chart Area
CAP_KILL
. . . .
CAP_NET_ADMIN
. . . .

Other Protection Improvement Methods

- System integrity protection (SIP)
  - Introduced by Apple in macOS 10.11
  - Restricts access to system files and resources, even by root
  - Uses extended file attribs to mark a binary to restrict changes, disable debugging and scrutinizing
  - Also, only code-signed kernel extensions allowed and configurably only code-signed apps
- System-call filtering
  - Like a firewall, for system calls
  - Can also be deeper –inspecting all system call arguments
  - Linux implements via SECCOMP-BPF (Berkeley packet filtering)

Other Protection Improvement Methods (Cont.)

- Sandboxing
  - Running process in limited environment
  - Impose set of irremovable restrictions early in startup of process (before `main()`)
  - Process then unable to access any resources beyond its allowed set
  - Java and .net implement at a virtual machine level
  - Other systems use MAC to implement
  - Apple was an early adopter, from macOS 10.5's "seatbelt" feature
    - Dynamic profiles written in the Scheme language, managing system calls even at the argument level
    - Apple now does SIP, a system-wide platform profile

Other Protection Improvement Methods (Cont.)

- Code signing allows a system to trust a program or script by using crypto hash to have the developer sign the executable
  - So code as it was compiled by the author
  - If the code is changed, signature invalid and (some) systems disable execution
  - Can also be used to disable old programs by the operating system vendor (such as Apple) cosigning apps, and then invaliding those signatures so the code will no longer run

Language-Based Protection

- Specification of protection in a programming language allows the high-level description of policies for the allocation and use of resources
- Language implementation can provide software for protection enforcement when automatic hardware-supported checking is unavailable
- Interpret protection specifications to generate calls on whatever protection system is provided by the hardware and the operating system

Protection in Java 2

- Protection is handled by the Java Virtual Machine (JVM)
- A **class** is assigned a protection domain when it is loaded by the JVM
- The protection domain indicates what operations the class can (and cannot) perform
- If a library **method** is invoked that performs a privileged operation, the stack is **inspected** to ensure the operation can be performed by the library
- Generally, Java’s load-time and run-time checks enforce **type safety**
- Classes effectively **encapsulate** and protect data and methods from other classes

Stack Inspection

protection domain:	untrusted applet	URL loader	networking
socket permission:	none	*.lucent.com:80, connect	any
class:	gui: ... get(url); open(addr); ...	get(URL u): ... doPrivileged { open('proxy.lucent.com:80'); } <request u from proxy> ...	open(Addr a): ... checkPermission (a, connect); connect (a); ...

Examples

copied from

- https://www.scs.stanford.edu/24wi-cs212/notes/protection.pdf
- https://ics.uci.edu/~goodrich/teach/cs201P/notes/01_SetUID.pdf

Unix protection and security holes

Microarchitectural attacks

Example Unix protections

Each process has a User ID & one or more group IDs

System stores with each file (in the inode):

- User who owns the file and group file is in

- Permissions for user, any one in file group, and other

$ ls -l or $ `stat -c "%a %A" ~/test/`

`-rwsr-Sr-t`

which has *setuid*, *setgid* and *sticky* attributes set.

Each triad

first character

`r`: readable

second character

`w`: writable

third character

`x`: executable

`s` or `t`: setuid/setgid or sticky (also executable)

`S` or `T`: setuid/setgid or sticky (not executable)

Directories have permissions too

drwxr-xr-x 56 root wheel 4096 Apr 4 10:08 /etc

- Directory writable only by root, readable by everyone
- Means non-root users cannot directly delete files in /etc

Non-file permissions in Unix

```
$ ls -l /dev/tty1

crw--w---- 1 root tty 4, 1 Dec 30 08:35 /dev/tty1
```

Other access controls not represented in file system; must usually be root to do the following:

- Bind any TCP or UDP port number less than 1024
- Change the current process's user or group ID
    - `usermod -a -G newgroup username`
- Mount or unmount most file systems
    - `mount /dev/sdb1 /mnt/media`
- Create device nodes (such as /dev/tty1) in the file system
    - `mknod <node> <mode> <major> <minor>`
- Change the owner of a file
    - `chown USER FILE`
- Set the time-of-day clock; halt or reboot machine
    - `date -s "12 OCT 2030 08:00:00"`
    - `reboot`
    - `suspend`

Example login runs as root

List of Unix users with accounts typically stored in files in /etc

- Files **passwd**, **group**, and often **shadow** or **master.passwd**

For each user, files contain:

- Textual username (e.g., “dm”, or “root”)

- Numeric user ID, and group ID(s)

- One-way hash of user’s password: {salt,H(salt, passwd)}

- Should have tunable difficulty d: {d, salt,Hd(salt, passwd)}

- Other information, such as user’s full name, login shell, etc.

/usr/bin/login runs as root

- Reads username & password from terminal

- Looks up username in **/etc/passwd**, etc.

- Computes H(salt,typed password) & checks that it matches

- If matches, sets group ID & user ID corresponding to username

- Execute user’s shell with execve system call

how should users change their passwords?

Stored in root-owned /etc/passwd & /etc/shadow files

$ls -l /etc/shadow

-rw-r----- 1 root shadow 994 Jul 9 13:37 /etc/shadow

How would non-root users change their password?

Solution: Setuid/setgid programs

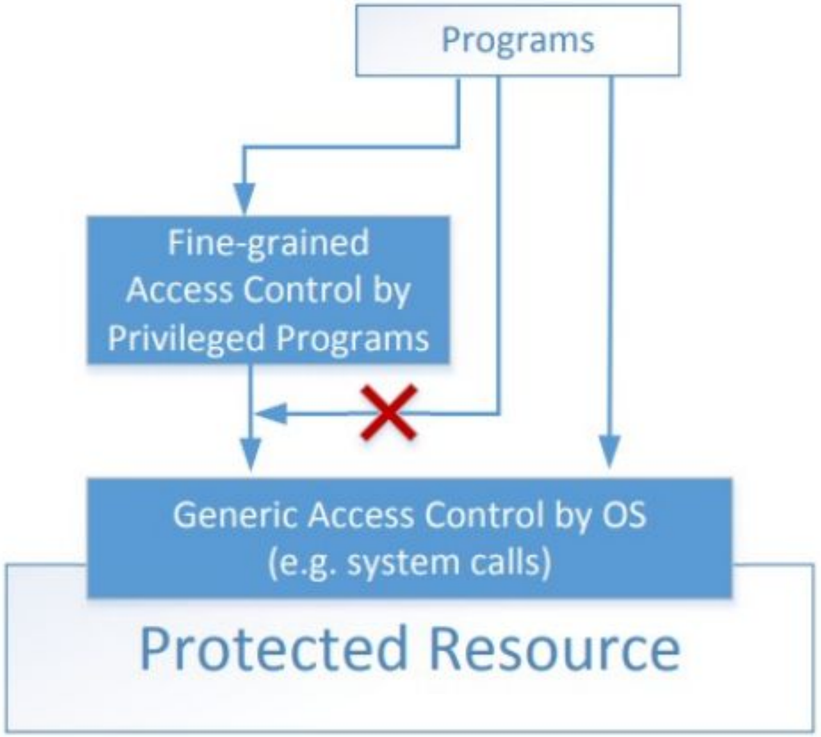
- Run with privileges of file’s owner or group
- Each process has real and effective UID/GID
- real is user who launched setuid program
- effective is owner/group of file, used in access checks

Two-Tier Approach

Implementing fine-grained access control in operating systems make OS over complicated.

OS relies on extension to enforce finegrained access control

Privileged programs are such extensions


Programs
Fine-grained Access Control by Privileged Programs
Generic Access Control by OS (e.g. system calls)
Protected Resource

Types of Privileged Programs

- Daemons
  - Computer program that runs in the background
  - Needs to run as root or other privileged users
- Set-UID Programs
  - Widely used in UNIX systems
  - Program marked with a special bit

Shown as “s” in file listings

- -rw**s**--x--x 1 root root 52528 Oct 29 08:54 /bin/passwd

- Obviously need to own file to set the setuid bit

- Need to own file and be in group to set setgid bit

SetUID concept

Allow user to run a program with the program owner's privilege.

Allow users to run programs with temporary elevated privileges

Example: the passwd program

```
$ ls -l /usr/bin/passwd

-rwsr-xr-x 1 root root 41284 Sep 12 2012 /usr/bin/passwd
```

Set-UID mechanism: A Power Suit mechanism implemented in Unix

SetUID concept

Every process has two User IDs.

- **Real UID (RUID):** Identifies real owner of process
- **Effective UID (EUID):** Identifies privilege of a process
    - Access control is based on EUID

When a normal program is executed,

RUID = EUID,

- they both equal to the ID of the user who runs the program

When a Set-UID is executed,

RUID ≠ EUID.

- RUID still equal to the user’s ID, but EUID equals to the program owner’s ID

Linux Capabilities

Wireshark needs network access, not ability to delete all files

Linux subdivides root's privileges into ~ 40 capabilities(7) - Linux manual page ,

e.g.:

- cap_net_admin – configure network interfaces (IP address, etc.)

- cap_net_raw – use raw sockets (bypassing UDP/TCP)

- cap_sys_boot – reboot; cap_sys_time – adjust system clock

Usually root gets all, but behavior can be modified by

"securebits" (see prctl(2) - Linux manual page)

Capabilities don't survive **execve** unless bits are set in both

thread & inode (exception: ambient capabilities)

"Effective" bit in inode acts like **setuid** for capability

$ ls -al /usr/bin/dumpcap

-rwxr-xr-- 1 root wireshark 116808 Jan 30 06:23 /usr/bin/dumpcap

$ getcap /usr/bin/dumpcap

/usr/bin/dumpcap cap_dac_override,cap_net_admin,cap_net_raw=eip

[Oops, cap_dac_override ≈ root! neeeded for USB capture]

• See also: getcap(8) - Linux manual page , setcap(8) - Linux manual page , capsh(1) - Linux manual page

Other permissions

When can process A send a signal to process B with kill?

- Allow if sender and receiver have same effective UID

- But need ability to kill processes you launch even if suid

- So allow if real UIDs match, as well

- Can also send SIGCONT w/o UID match if in same session

Debugger system call ptrace

- Lets one process modify another's memory

- Setuid gives a program more privilege than invoking user

- So don't let a process ptrace a more privileged process

- E.g., Require sender to match real & effective UID of target

- Also disable/ignore setuid if ptraced target calls exec

- Exception: root can ptrace anyone

Security holes in Unix

Even without root or setuid, attackers can trick root owned processes into doing things. . .

```
$ cp /bin/cat ./mycat
$ sudo chown root maycat

#before we nable set-UID
 $ ./mycat /etc/shadow
./mycat: /etc/shadow: Permission denied
```

```
$ ls -l mycat
-rwxr-xr-x 1 root adaskin 44016 Dec 30 09:47 mycat
```

```
# Enable set-UID
$ sudo chmod 4755 mycat
$ ls -l mycat
-rwsr-xr-x 1 root adaskin 44016 Dec 30 09:47 mycat


$ ./mycat /etc/shadow
…
```

A Set-UID program is just like any other program, except that it has a special marking, which a single bit called Set-UID bit

```
$ cp /bin/id ./myid
$ sudo chown root myid
$ ./myid
uid=1000(seed) gid=1000(seed) groups=1000(seed),
```

```
$ sudo chmod 4755 myid
$ ./myid
uid=1000(seed) gid=1000(seed) euid=0(root) ...
```

```
$ cp /bin/cat ./mycat
$ sudo chown root mycat
$ ls -l mycat
-rwxr-xr-x 1 root seed 46764 Feb 22 10:04 mycat
$ ./mycat /etc/shadow
./mycat: /etc/shadow: Permission denied
```

↰ Not a privileged program

```
$ sudo chmod 4755 mycat
$ ./mycat /etc/shadow
root:$6$012BPz.K$fbPkT6H6Db4/B8c...
daemon:*:15749:0:99999:7:::
...
```

↰ Become a privileged program

```
$ sudo chown seed mycat
$ chmod 4755 mycat
$ ./mycat /etc/shadow
./mycat: /etc/shadow: Permission denied
```

↰ It is still a privileged program, but not the root privilege

How is Set-UID Secure?

Allows normal users to escalate privileges

• This is different from directly giving the privilege (sudo command)

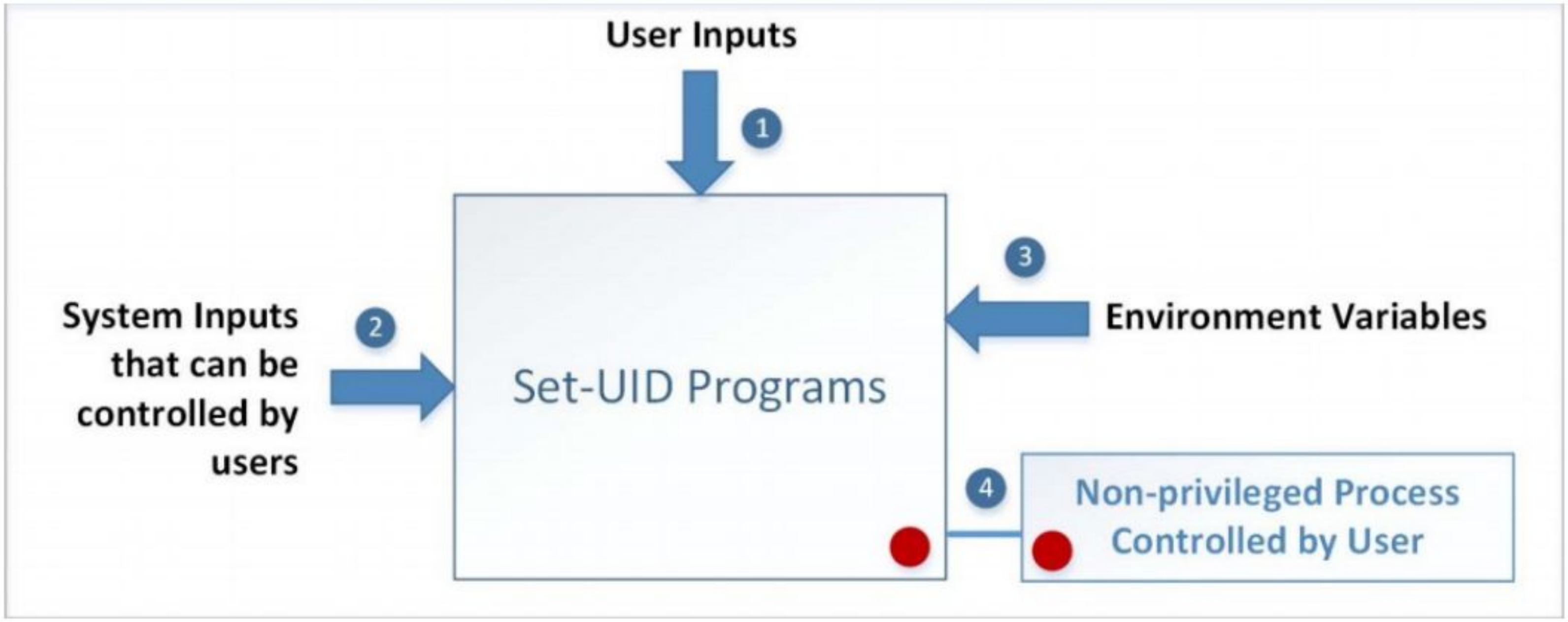
• Restricted behavior – similar to superman designed computer chips

Unsafe to turn all programs into Set-UID

• Example: /bin/sh

• Example: vi

Attack Surfaces of Set-UID Programs


User Inputs
1
System Inputs that can be controlled by users
2
Set-UID Programs
3
Environment Variables
4
Non-privileged Process Controlled by User

Attacks via User Inputs

User Inputs: Explicit Inputs

• Buffer Overflow

Overflowing a buffer to run malicious code

• Format String Vulnerability

Changing program behavior using user inputs as format strings

CHSH – Change Shell

• Set-UID program with ability to change default shell programs

• Shell programs are stored in /etc/passwd file

Issues

• Failing to sanitize user inputs

• Attackers could create a new root account

Attack input

```
bob:$6$jUODEFsfwfi3:1000:1000:Bob Smith,,,:/home/bob:/bin/bash
```

Attacks via System Inputs

System Inputs

Race Condition

- **the time-of-check-to-time-of use (TOCTTOU) flaws**
- Symbolic link to privileged file from a unprivileged file
- Influence programs
- Writing inside world writable folder

Even without root or setuid, attackers can trick root owned processes into doing things. . .

Attacks via System Inputs

Example: Want to clear unused files in /tmp

• Every night, automatically run this command as root:

find /tmp -atime +3 -exec rm -f -- {} \;

find /tmp -atime +3 -exec rm -f -- {} \;

• find identifies files not accessed in 3 days

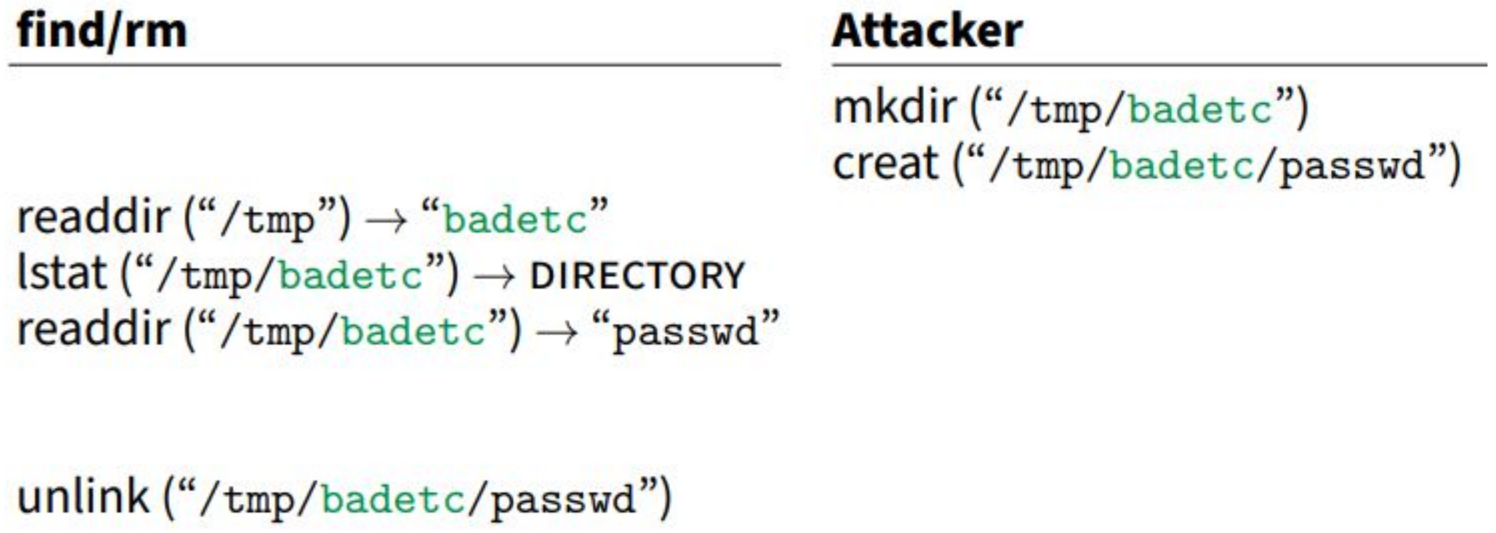
- executes rm, replacing {} with file name

• rm -f -- path deletes file path

- Note “--” prevents path from being parsed as option

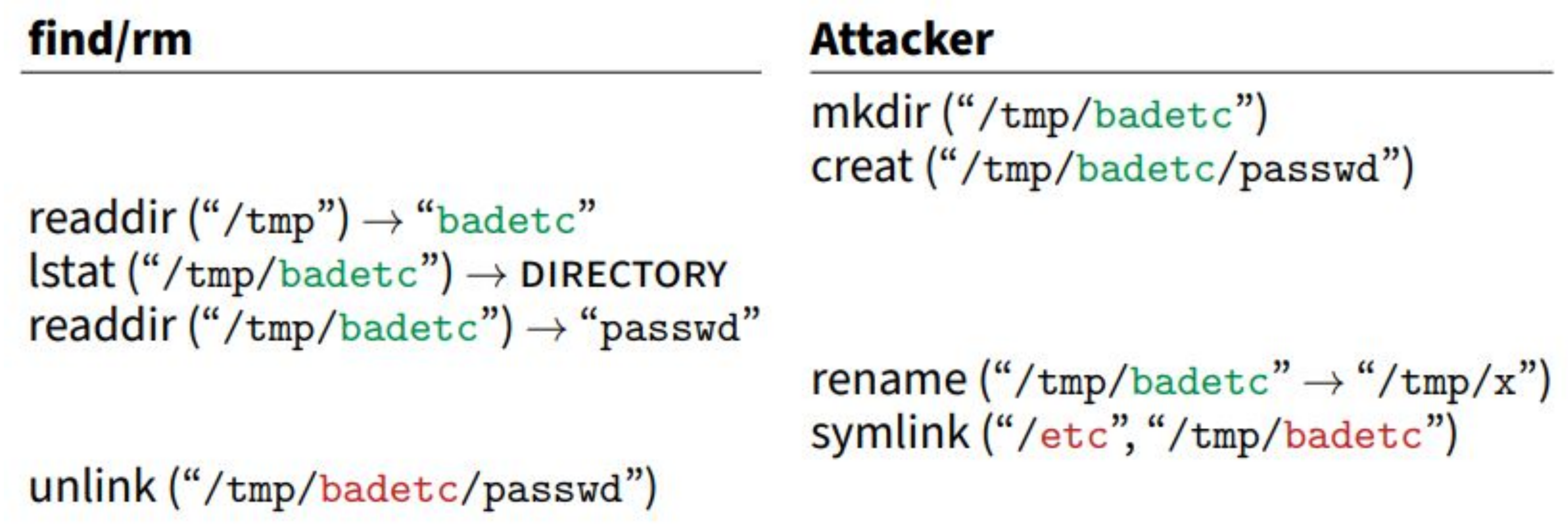
• What’s wrong here?

An attack



find/rm	Attacker
	mkdir (“/tmp/badetc”)
	creat (“/tmp/badetc/passwd”)
readdir (“/tmp”) → “badetc”	
lstat (“/tmp/badetc”) → DIRECTORY	
readdir (“/tmp/badetc”) → “passwd”	
unlink (“/tmp/badetc/passwd”)	

Time-of-check-to-time-of-use [TOCTTOU] bug



find/rm	Attacker
	mkdir ("/tmp/badetc")
	creat ("/tmp/badetc/passwd")
readdir ("/tmp") → "badetc"	
lstat ("/tmp/badetc") → DIRECTORY	
readdir ("/tmp/badetc") → "passwd"	
	rename ("/tmp/badetc" → "/tmp/x")
	symlink ("/etc", "/tmp/badetc")
unlink ("/tmp/badetc/passwd")	

- find checks that /tmp/badetc is not symlink
- But meaning of file name changes before it is used

xterm command

Provides a terminal window in X-windows

Used to run with setuid root privileges

- - Requires kernel pseudo-terminal (pty) device
- - Required root privs to change ownership of pty to user
- - Also writes protected utmp/wtmp files to record users

Had feature to log terminal session to file

Programs may not clean up privileged capabilities before downgrading

```
fd = open (logfile, O_CREAT|O_WRONLY|O_TRUNC, 0666); /* ... */
```

xterm command

• Had feature to log terminal session to file

```
if (access (logfile, W_OK) < 0)
    return ERROR;
fd = open (logfile, O_CREAT|O_WRONLY|O_TRUNC, 0666);
/* ... */
```

• xterm is root, but shouldn’t log to file user can’t write

• access call avoids dangerous security hole

- Does permission check with real, not effective UID

Wrong: Another TOCTTOU bug

An attack

xterm	Attacker
	creat (“/tmp/log”)
access (“/tmp/log”) → OK	
	unlink (“/tmp/log”)
	symlink (“/tmp/log” → “/etc/passwd”)
open (“/tmp/log”)	

- **Attacker changes** `/tmp/log` **between check and use**
  - xterm unwittingly overwrites `/etc/passwd`
  - Another TOCTTOU bug
- **OpenBSD man page: “CAVEATS: access() is a potential security hole and should never be used.”**

Preventing TOCCTOU

Use new APIs that are relative to an opened directory fd

- openat, renameat, unlinkat, symlinkat, faccessat
- fchown, fchownat, fchmod, fchmodat, fstat, fstatat
- O_NOFOLLOW flag to open avoids symbolic links in last component
- But can still have TOCTTOU problems with hardlinks

Lock resources, though most systems only lock files (and locks are typically advisory)

Wrap groups of operations in OS transactions

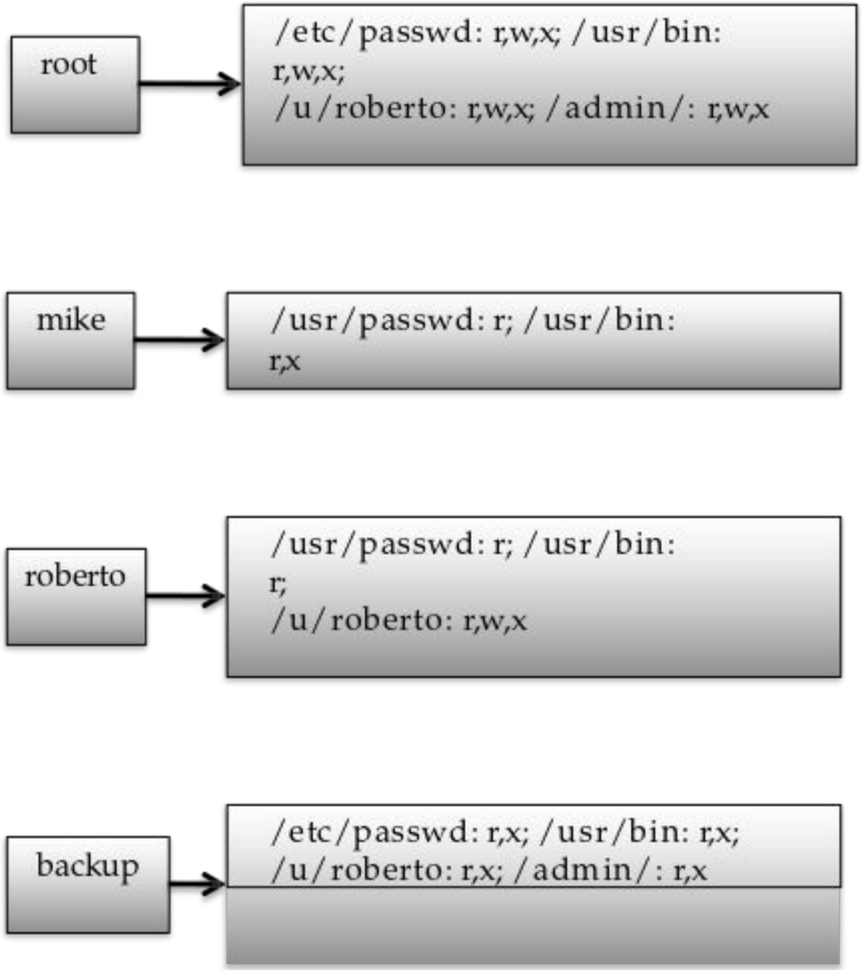
- - Microsoft supports for transactions on Windows Vista and newer
- CreateTransaction Kernel Transaction Manager - Win32 apps | Microsoft Learn , CommitTransaction, RollbackTransaction
- - A few research projects for POSIX [Valor] [TxOS]

Remember: Capabilities

Objects

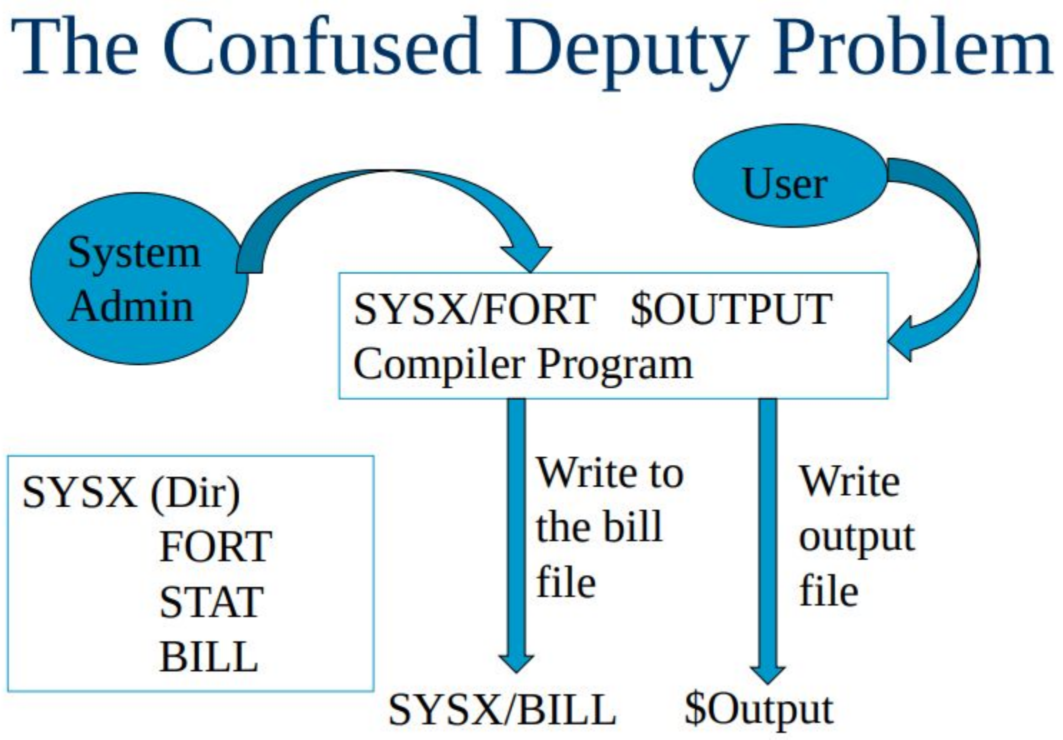
Subjects

	File 1	File 2	File 3	…	File n
User 1	read	write	-	-	read
User 2	write	write	write	-	-
User 3	-	-	-	read	read
…					
User m	read	write	read	write	read


root
/etc/passwd: r,w,x; /usr/bin: r,w,x;
/u/roberto: r,w,x; /admin/: r,w,x
mike
/usr/passwd: r; /usr/bin: r,x
roberto
/usr/passwd: r; /usr/bin: r;
/u/roberto: r,w,x
backup
/etc/passwd: r,x; /usr/bin: r,x;
/u/roberto: r,x; /admin/: r,x

A confused deputy

```
fort -o /sysx/bill file.f
```



https://www.cs.purdue.edu/homes/ninghui/courses/426_Fall10/handouts/426_Fall10_lect19.pdf

- The compiler runs with authority from two sources
  - the invoker (i.e., the programmer)
  - the system admin (who installed the compiler and controls billing and other info)

It is the deputy of two masters

When access a resource, must select a capability, which also selects a master

- this solves the problem

Capabilities

- Can help avoid confused deputy problem

Three general approaches to capabilities:

- Hardware enforced (Tagged architectures like M-machine)

- Kernel-enforced (Hydra, KeyKOS)

- Self-authenticating capabilities (like Amoeba)

- Good history in [Levy]

Limitations of capabilities

IPC performance a losing battle with CPU makers

- CPUs optimized for “common” code, not context switches
- Capability systems usually involve many IPCs

Capability model never fully took off as kernel API

- - Requires changes throughout application software
- - Call capabilities “file descriptors” or “Java pointers” and people
- will use them
- - But discipline of pure capability system challenging so far
- - People sometimes quip that capabilities are an OS concept of the
- future and always will be

But real systems do use capabilities

- - Firefox security based on language-level object capabilities
- - FreeBSD now ships with Capsicum, making capabilities available

Capability Leaking

• In some cases, Privileged programs downgrade themselves during execution

• Example: The su program

• This is a privileged Set-UID program

• Allows one user to switch to another user (say user1 to user2)

• Program starts with EUID as root and RUID as user1

• After password verification, both EUID and RUID become user2’s (via privilege downgrading)

Such programs may lead to capability leaking

Programs may not clean up privileged capabilities before downgrading

Attacks via Capability Leaking

The /etc/zzz file is only writable by root

File descriptor is created (the program is a root-owned Set-UID program)

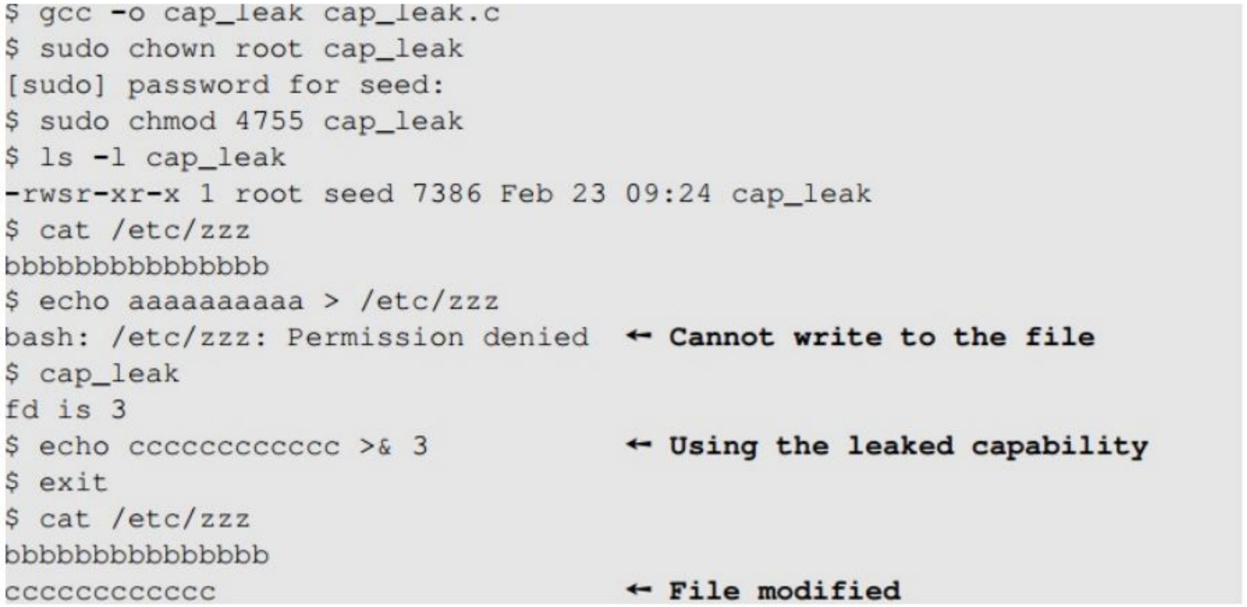
The privilege is downgraded

Invoke a shell program, so the behavior restriction on the program is lifted

```
fd = open("/etc/zzz", O_RDWR | O_APPEND);
if (fd == -1) {
   printf("Cannot open /etc/zzz\n");
   exit(0);
}

// Print out the file descriptor value
printf("fd is %d\n", fd);

// Permanently disable the privilege by making the
// effective uid the same as the real uid
setuid(getuid());

// Execute /bin/sh
v[0] = "/bin/sh"; v[1] = 0;
execve(v[0], v, 0);
```

The program forgets to close the file, so the file descriptor is still valid.

Capability Leak



```
$ gcc -o cap_leak cap_leak.c
$ sudo chown root cap_leak
[sudo] password for seed:
$ sudo chmod 4755 cap_leak
$ ls -l cap_leak
-rwsr-xr-x 1 root seed 7386 Feb 23 09:24 cap_leak
$ cat /etc/zzz
bbbbbbbbbbbbbbb
$ echo aaaaaaaaaa > /etc/zzz
bash: /etc/zzz: Permission denied   ← Cannot write to the file
$ cap_leak
fd is 3
$ echo cccccccccccc >& 3            ← Using the leaked capability
$ exit
$ cat /etc/zzz
bbbbbbbbbbbbbbb
cccccccccccc                        ← File modified
```

How to fix the program?

Destroy the file descriptor before downgrading the privilege (close the file)

Capability Leaking in OS X – Case Study

- OS X Yosemite found vulnerable to privilege escalation attack related to capability leaking in July 2015 (OS X 10.10)
- Added features to dynamic linker `dyld`
  - DYLD_PRINT_TO_FILE environment variable
- The dynamic linker can open any file, so for root-owned Set-UID programs, it runs with root privileges. The dynamic linker `dyld`, does not close the file. There is a capability leaking.
- Scenario 1 (safe): Set-UID finished its job and the process dies. Everything is cleaned up and it is safe.
- **Scenario 2 (unsafe):** Similar to the "`su`" program, the privileged program downgrade its privilege, and lift the restriction.

Invoking Programs

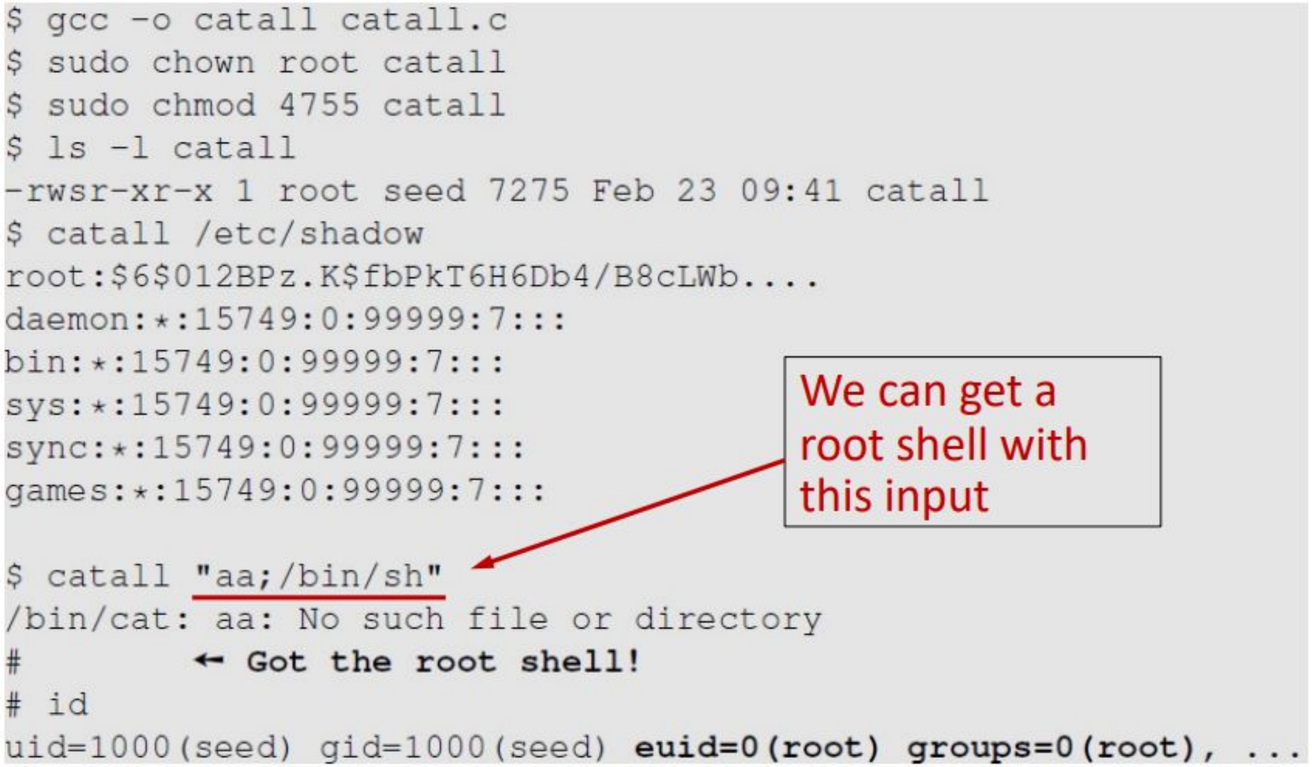
- Invoking external commands from inside a program
- External command is chosen by the Set-UID program
  - Users are not supposed to provide the command (or it is not secure)
- Attack:
  - Users are often asked to provide input data to the command.
  - If the command is not invoked properly, user's input data may be turned into command name. This is dangerous.

Invoking Programs : Unsafe Approach

```
int main(int argc, char *argv[])
{
  char *cat="/bin/cat";

  if(argc < 2) {
    printf("Please type a file name.\n");
    return 1;
  }

  char *command = malloc(strlen(cat) + strlen(argv[1]) + 2);
  sprintf(command, "%s %s", cat, argv[1]);
  system(command);
  return 0 ;
}
```

- The easiest way to invoke an external command is the system() function.
- This program is supposed to run the `/bin/cat` program.
- It is a root-owned Set-UID program, so the program can view all files, but it can't write to any file.

Question: Can you use this program to run other command, with the root privilege?



```
$ gcc -o catall catall.c
$ sudo chown root catall
$ sudo chmod 4755 catall
$ ls -l catall
-rwsr-xr-x 1 root seed 7275 Feb 23 09:41 catall
$ catall /etc/shadow
root:$6$012BPz.K$fbPkT6H6Db4/B8cLWb....
daemon:*:15749:0:99999:7:::
bin:*:15749:0:99999:7:::
sys:*:15749:0:99999:7:::
sync:*:15749:0:99999:7:::
games:*:15749:0:99999:7:::

$ catall "aa;/bin/sh"
/bin/cat: aa: No such file or directory
#        ← Got the root shell!
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=0(root), ...
```

Problem: Some part of the data becomes code (command name)

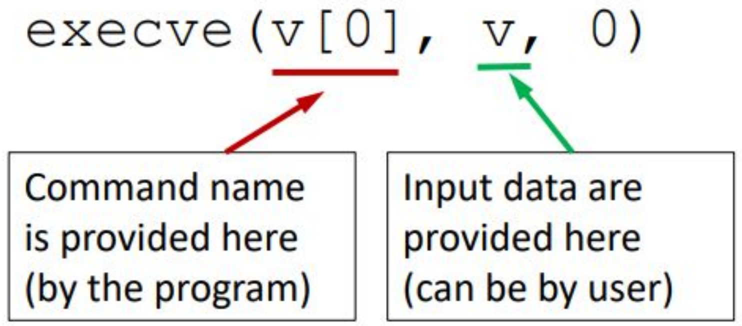
A Note

- In Ubuntu 16.04, /bin/sh points to /bin/dash, which has a countermeasure
  - It drops privilege when it is executed inside a set-uid process
- Therefore, we will only get a normal shell in the attack on the previous slide
- Do the following to remove the countermeasure

```
Before experiment: link /bin/sh to /bin/zsh
$ sudo ln -sf /bin/zsh /bin/sh

After experiment: remember to change it back
$ sudo ln -sf /bin/dash /bin/sh
```

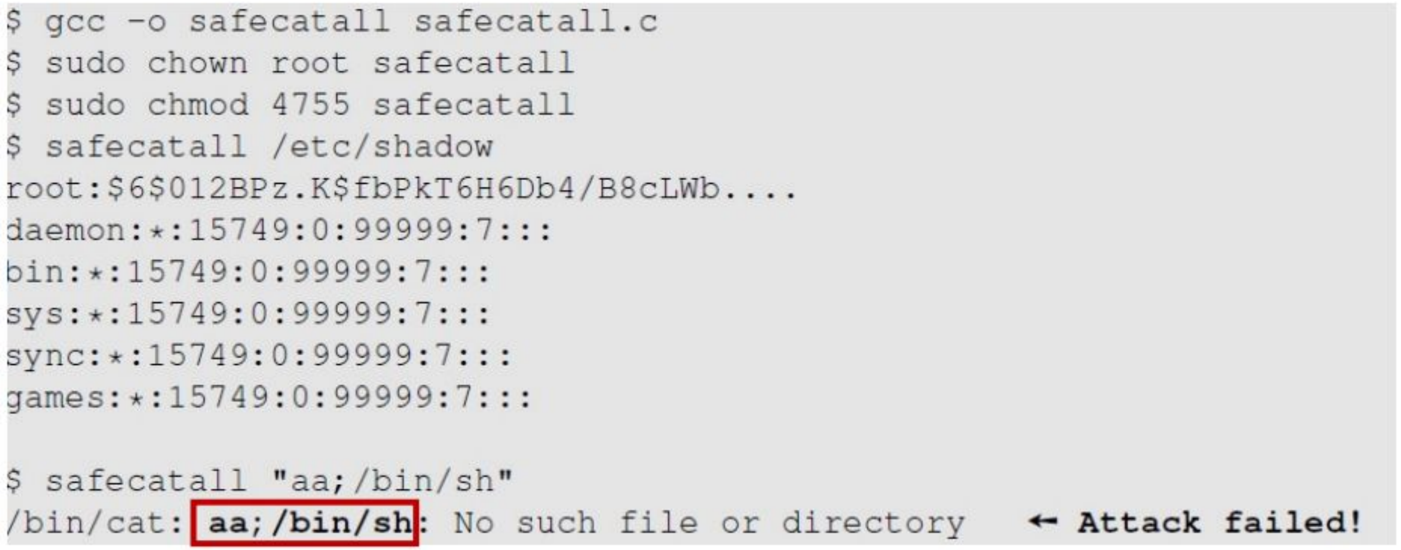
Invoking Programs Safely: using **execve()**

```
int main(int argc, char *argv[])
{
  char *v[3];

  if(argc < 2) {
    printf("Please type a file name.\n");
    return 1;
  }

  v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = 0;
  execve(v[0], v, 0);

  return 0 ;
}
```


execve(v[0], v, 0)
Command name is provided here (by the program)
Input data are provided here (can be by user)

Why is it safe?
Code (command name) and data are clearly separated; there is no way for the user data to become code

Invoking Programs Safely (Continued)



```
$ gcc -o safecatall safecatall.c
$ sudo chown root safecatall
$ sudo chmod 4755 safecatall
$ safecatall /etc/shadow
root:$6$012BPz.K$fbPkT6H6Db4/B8cLWb....
daemon:*:15749:0:99999:7:::
bin:*:15749:0:99999:7:::
sys:*:15749:0:99999:7:::
sync:*:15749:0:99999:7:::
games:*:15749:0:99999:7:::

$ safecatall "aa;/bin/sh"
/bin/cat: aa;/bin/sh: No such file or directory   ← Attack failed!
```

The data are still treated as data, not code

Note: execlp(), execvp() and execvpe() duplicate the actions of the shell. These functions can be attacked using the PATH Environment Variable

Invoking External Commands in Other Languages

- Risk of invoking external commands is not limited to C programs
- We should avoid problems similar to those caused by the system() functions
- Examples:
  - Perl: open() function can run commands, but it does so through a shell
  - PHP: system() function

```
<?php
  print("Please specify the path of the directory");
  print("<p>");
  $dir=$_GET['dir'];
  print("Directory path: " . $dir . "<p>");
  system("/bin/ls $dir");
?>
```

  - Attack:
    - `http://localhost/list.php?dir=.;date`
    - Command executed on server : "`/bin/ls .;date`"

Attacks via Environment Variables

Behavior can be influenced by inputs that are not visible inside a program.

Environment Variables: These can be set by a user before running a program.

PATH Environment Variable

• Used by shell programs to locate a command if the user does not provide the full path for the command

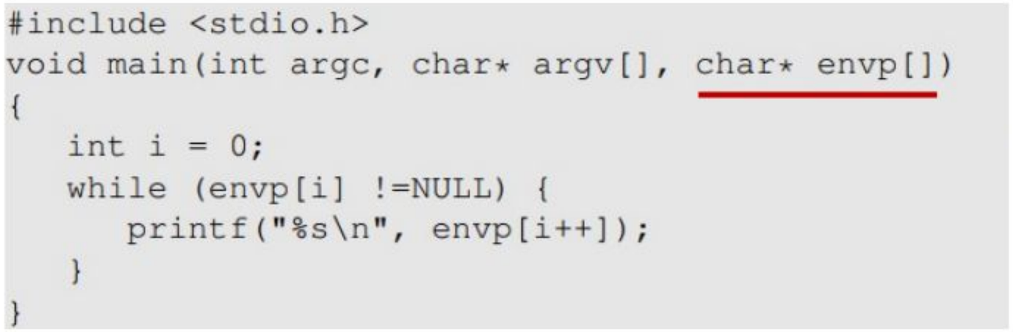
• system(): call /bin/sh first

• system("ls")

• /bin/sh uses the PATH environment variable to locate "ls"

• Attacker can manipulate the PATH variable and control how the "ls" command is found

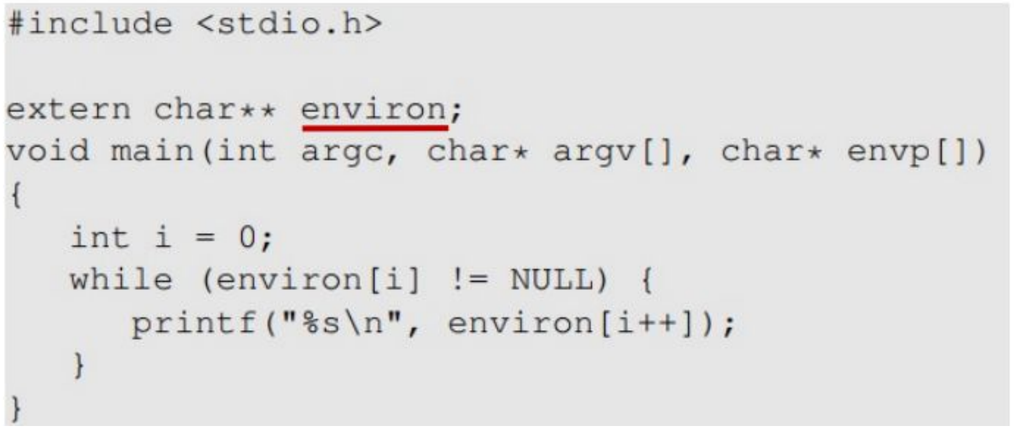
How to Access Environment Variables



```c
#include <stdio.h>
void main(int argc, char* argv[], char* envp[])
{
   int i = 0;
   while (envp[i] !=NULL) {
      printf("%s\n", envp[i++]);
   }
}
```

From the main function

More reliable way:
Using the global variable

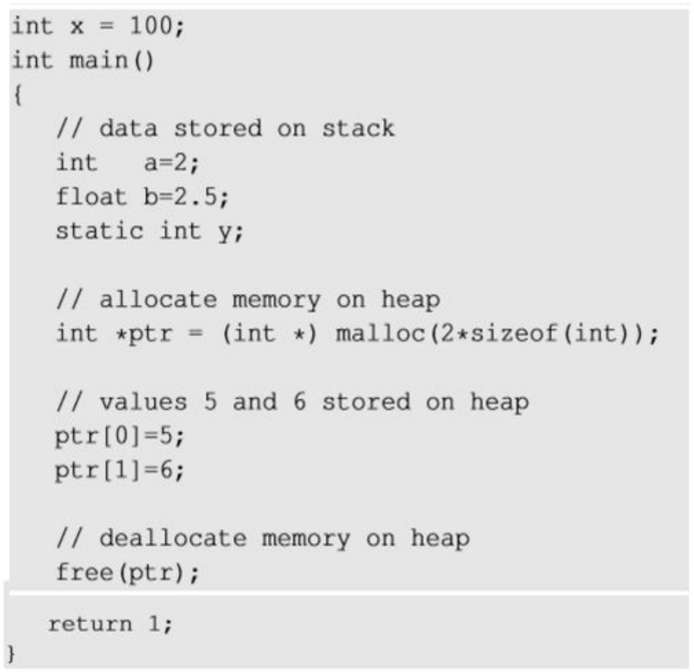


```c
#include <stdio.h>

extern char** environ;
void main(int argc, char* argv[], char* envp[])
{
   int i = 0;
   while (environ[i] != NULL) {
      printf("%s\n", environ[i++]);
   }
}
```

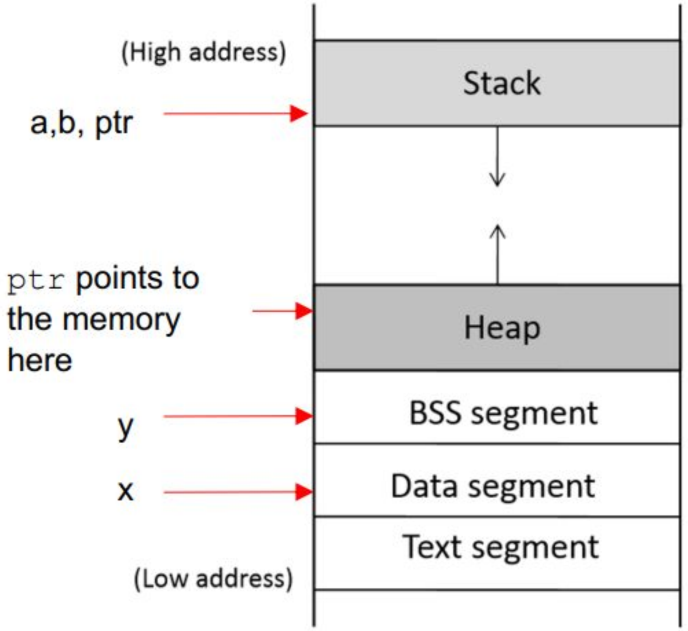
for attack examples see:
https://ics.uci.edu/~goodrich/teach/cs201P/notes/02_Environment_Variables.pdf

Buffer overflow attack

Program Memory Stack



```
int x = 100;
int main()
{
    // data stored on stack
    int   a=2;
    float b=2.5;
    static int y;

    // allocate memory on heap
    int *ptr = (int *) malloc(2*sizeof(int));

    // values 5 and 6 stored on heap
    ptr[0]=5;
    ptr[1]=6;

    // deallocate memory on heap
    free(ptr);

   return 1;
}
```



for examples and prevention see
https://ics.uci.edu/~goodrich/teach/cs201P/notes/04_Buffer_Overflow.pdf

Countermeasures for buffer overflow attacks

Developer approaches:

- • Use of safer functions like strncpy(), strncat() etc,
- safer dynamic link libraries that check the length of the data before copying.

OS approaches:

ASLR (Address Space Layout Randomization)

- To randomize the start location of the stack that is every time the code is loaded in the memory, the stack address changes.
- not totally solve

Compiler approaches:

- Stack-Guard
- Canary check done by compiler.

Hardware approaches:

- Non-Executable Stack

Principle of Isolation

Principle: Don’t mix code and data.

Attacks due to violation of this principle :

• system() code execution

• Cross Site Scripting –

• SQL injection -

• Buffer Overflow attacks -

Principle of Least Privilege

A privileged program should be given the power which is required to perform it’s tasks.

• Disable the privileges (temporarily or permanently) when a privileged program doesn’t need those.

In Linux, seteuid() and setuid() can be used to disable/discard privileges.

Different OSes have different ways to do that

Example: Windows 10

- Security is based on **user accounts**
    - Each user has unique security ID
    - Login to ID creates **security access token**
        - Includes security ID for user, for user’s groups, and special privileges
        - Every process gets copy of token
        - System checks token to determine if access allowed or denied
- Uses a **subject** model to ensure access security
    - A subject tracks and manages permissions for each program that a user runs
- Each object in Windows has a security attribute defined by a security descriptor
    - For example, a file has a **security descriptor** that indicates the access permissions for all users

Example: Windows 7 (Cont.)

- Win added mandatory integrity controls – assigns **integrity label** to each securable object and subject
    - Subject must have access requested in discretionary access-control list to gain access to object
- Security attributes described by security descriptor
    - Owner ID, group security ID, discretionary access-control list, system access-control list
- Objects are either **container objects** (containing other objects, for example a file system directory) or **noncontainer objects**
    - By default an object created in a container inherits permissions from the parent object
- Some Win 10 security challenges result from security settings being weak by default, the number of services included in a Win 10 system, and the number of applications typically installed on a Win 10 system

Microarchitectural attacks

Cache timing attacks

```
const char *table;

int victim (int secret_byte){

        return table[secret_byte*64];

}
```

Accessing memory based on secret data can leak the data

Approach 1: Flush/Evict + Reload

- **Share table with victim process (shared lib or deduplication)**
- **Flush table from cache (clflush instruction, or overflow cache)**
- **After victim, time reads of table, fast line tells you secret_byte**

Approach 2: Prime + Probe

- **No shared memory, but attacker primes cache with its own buffer**
- **Victim's table access evicts one of attacker's cache lines**
- **Slow cache line (+ cache mapping) reveals secret data**

Speculative execution key to performance

```
unsigned char *array1, *array2;

int array1_size, array2_size;

int lookup (int input){

        if (input < array1_size)

        return array2[array1[input] * 4096];

        return -1;

}
```

CPU predicts branches to mask memory latency

e.g. predict input < array_size even if array1_size not cached

Wait to get array1_size from memory before retiring instructions

Squash incorrectly predicted instructions by reverting registers

But can’t revert cache state, only registers

Example: intel Haswell

Spectre attack

```
unsigned char *array1, *array2;

int array1_size, array2_size;

int lookup (int input){

        if (input < array1_size)

        return array2[array1[input] * 4096];

        return -1;

}
```

Say attacker supplies input, wants to read array1[input]

- input can exceed bounds, reference any byte in address space

Ensure array1 cached, but array1_size and array2 uncached

Flush+reload attack on array2 now reveals array1[input]

see https://spectreattack.com/spectre.pdf for more

Many more variants of Spectre

Attack on JavaScript JIT

- Malicious JavaScript reads secrets outside of JavaScript sandbox

eBPF compiles packet filters in kernel (e.g., for tcpdump)

- Can generate code to reveal arbitrary kernel memory

Can even use victim code that's not supposed to be executed

- - Mistrain branch predictor on indirect branch

Use other speculation channels

Mitigation

Replace array bounds checks with index masking (used by Chrome)

return array2[array1[input&0xffff] * 4096]

- Limits distance of bounds violation

• Place JavaScript sandbox in separate address space

XOR pointers with type-dependent poison values (in JITs)

- xor pointers with random poison value

• Make CPUs a bit better about leaking state through side channels

• Insert “gratuitous” memory barriers to prevent speculation on sensitive data

• Unfortunately general solution still an open problem

OS Security

https://www.scs.stanford.edu/24wi-cs212/notes/security.pdf