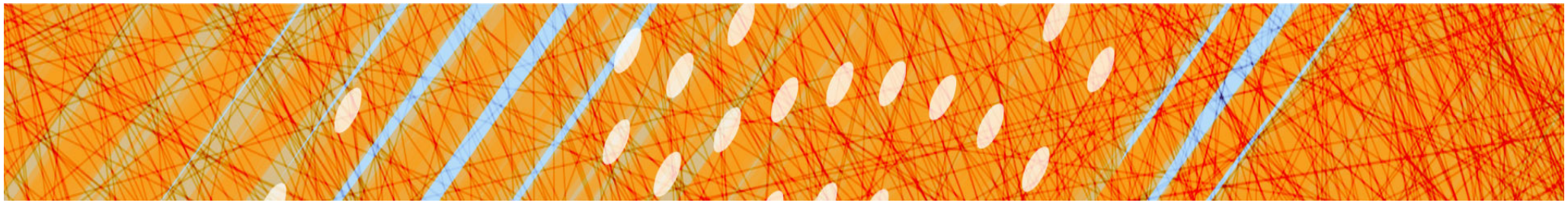
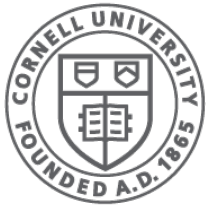


Security

CS 4410
Operating Systems

[E. Birrell, A. Bracy, F. B. Schneider,
E. Sirer, R. van Renesse]



Cornell CIS
COMPUTING AND INFORMATION SCIENCE

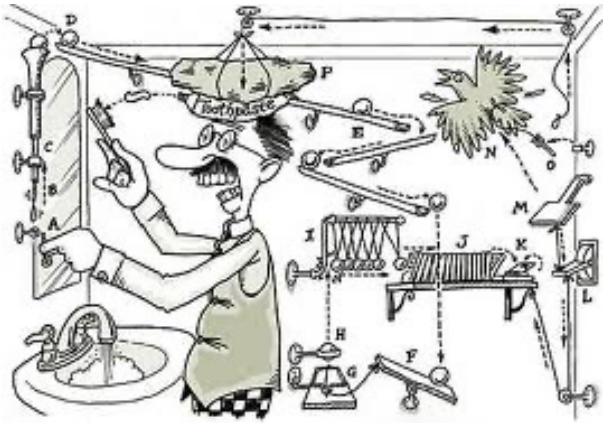
References: Security Introduction and Access Control by Fred B. Schneider

Secure Systems?

A **secure** system will

- do what is expected
- not do the unexpected

despite attacks *and*
offers assurance about this claim



Example:

- Do more: reveal secrets
- Do less: fail to store or retrieve information

Security Properties: CIA

Confidentiality: *keeping secrets*

- who is allowed to learn what information?

Integrity: *permitting changes*

- what changes to the system are allowed?

Availability: *guarantee of service*

- is the service “timely”?

Protect against what?

Some terminology:

vulnerability: Weakness that can be used to cause damage

attack: Method of exploiting a vulnerability

threat: Motivated capable adversary who will mount attacks

All systems have vulnerabilities

Understand the threats and defend against attacks they can mount

All assumptions are vulnerabilities

Cyber threats

- Operator/user blunders.
- Hackers driven by intellectual challenge (or boredom).
- Insiders: employees or customers seeking revenge.
- Criminals seeking financial gain.
- Organized crime seeking gain or hiding criminal activities.
- Terrorist groups or nation states trying to influence national policy.

Cyber threats: Classification

Class I: Execute existing attacks against known vulnerabilities.

Class II: Analyze system, find new vulnerabilities, develop new attacks.

Class III: Create new vulnerabilities (e.g., compromise the supply chain).

Security in the “real world”

Use locks to block attacks: “Prevention”

- Locks must not be annoying, or they won’t be used.
- All locks aren’t the same. They are:
  - Scaled for what they are protecting.
  - Scaled for their environment.
- Police and courts are central---not the locks! “Deterrence through accountability”
  - Tracking down the “bad guys” is what’s central.
  - Locks reduce temptation and reduce workload on police and courts.

“Real world” (cont’d)

- People only pay for security that they think they need
- People avoid annoying locks by buying insurance
- Security is only as strong as the weakest link

Security in Computer Systems

Gold (Au) Standard for Security [Lampson]

Authorization: mechanisms that govern whether actions are permitted

Authentication: mechanisms that bind *principals* to actions

Audit: mechanisms that record and review actions

Security Mechanism Design

What should the mechanism do?

Best to distinguish **policy** from **mechanism**.

Desire mechanisms that implement many policies.

Authorization: Access Control

Operations: learn and/or update information

Principals: instigate operations

- users, processes, threads, AI agents

Objects of operations: memory, files, services

Access Control Policy:

- which principals may perform which operations on which objects

… Enforces confidentiality & integrity

Access Control Mechanisms

Reference Monitor:

- Consulted on each operation invocation
- Allows operation if invoker has required privileges
  - … Can enforce **confidentiality** and/or **integrity**

Assumptions:

- Predefined operations are the sole means by which principals learn or update information
- All predefined operations can be monitored (“complete mediation”)

Trusted Computing Base (TCB)

A secure system should have a well defined TCB in order to discharge assurance requirements.

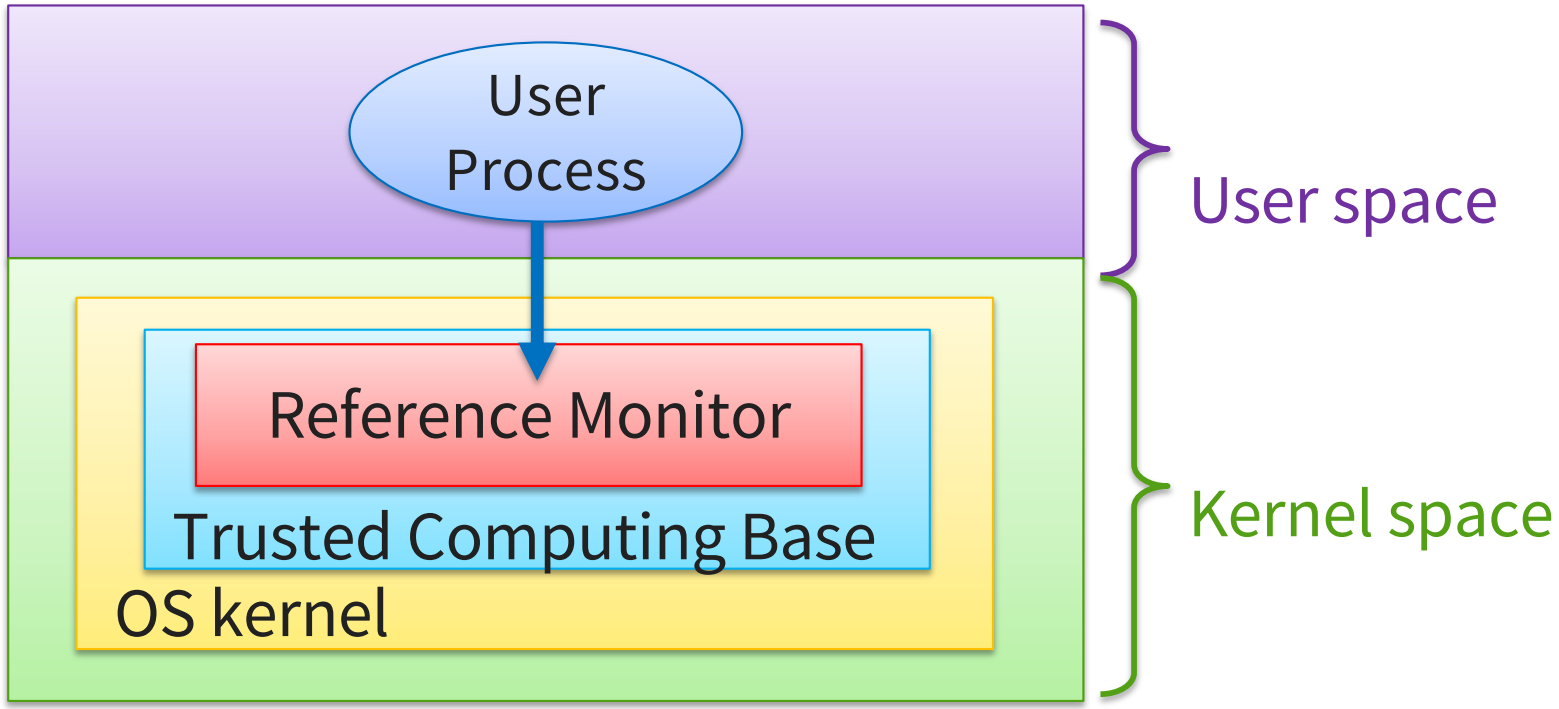
TCB typically contains:

- HW & SW necessary for enforcing security rules
  - most hardware, firmware
  - portion of OS kernel
  - most or all programs with superuser power
- Desirable features of TCB to facilitate analysis:
  - Small size
  - Isolated, so separable

Reference Monitor is in TCB

- All operations intercepted by reference monitor
- Monitor decides: should operation proceed?

… Reference monitor is not a separable module in most OSes…



Who defines authorizations?

Discretionary Access Control (DAC):

- Object **owner** defines authorizations for operations on that object
- Supported in all OS (Linux/MacOSX/Windows File Systems)
- Vulnerable to "Trojan Horse" attacks

Mandatory Access Control (MAC):

- System imposes access control policy that object owners cannot change
- Centralized authority defines authorizations
- Commonly required in institutional settings

Principle of Least Privilege

"Every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job."

- Jerome Saltzer

Implies that we should want to minimize:

- code running inside kernel
- code running as sysadmin (root)

Access Control Matrix

- Abstract model of authorization
- Rows: principals = users
- Columns: objects = files, I/O, *etc.*

Principals	**OBJECTS**		
	`prelim.pdf`	`jan-hw.tex`	`scores.xls`
rvr (prof)	r, w	r	r, w
jan (student)		r, w	

Implementing the “matrix”

- By column:
  - For each object, keep a list of which principals can access the object and how
  - “Access Control List” per object

- By row:
  - For each principal, keep track of what operations that principal can do on what objects
  - “Capability List” per principal

Access Control Lists (ACL)

ACL for an object is a list:

e.g., ⟨ebirrell, {r,w}⟩ ⟨clarkson, {r}⟩ ⟨student, {r}⟩

To check whether a principal P is authorized to perform an operation Op on an object Obj:

- Look up principal P in ACL for Obj.
  - If P not in ACL, reject operation
  - Otherwise, obtain set S of authorized operations.
- Check whether Op is in S.
  - If not, reject in S, reject operation.
  - Otherwise, allow Op to proceed.

Access Control Lists: Roundup

Advantages:

- Efficient to review of permissions for an object
- Revocation is straightforward

Disadvantages:

- Inefficient review of permissions for a principal
- Large ACLs take up space in object
- Vulnerable to confused deputy attack

Capability Lists

The *capability list* for a principal P is a list

e.g., ⟨dac.tex, {r,w}⟩ ⟨dac.pptx, {r,w}⟩

- Performing operation Op on object Obj requires a principal to hold a capability for Obj and Op.
- Capabilities must be unforgeable, so they cannot be counterfeited or corrupted.

Note: Capabilities are (typically) transferable.

Access Control in UNIX

UNIX: has user and group identifiers: `uid` and `gid`

Per process: protection domain = `uid=rvr` and `gid=faculty`

Per file: ACL `owner|group|other` → stored in i-node

- Only owner can change these rights (using chmod)
- Each i-node has 3x3 RWX bits for user, group, others
  - eg.: `RWXRW-R--`
- 2 mode bits allow process to change across domains
  - setuid, setgid bits

(Hybrid!) UNIX access control scheme:

- Authorization (check ACL) performed at `open`
- Returns a file handle → essentially a capability
- Subsequent `read` or `write` uses the file handle

Capabilities: Roundup

Advantages:

- Eliminates confused deputy problems
- Natural approach for user-defined objects

Disadvantages:

- Review of permissions?
- Delegation?
- Revocation?

ACLs vs Capabilities

	ACLs: For each Object: <P_1,$privs_1$> <P_2,$privs_2$>…	**Capabilities:** For each Principal: <O_1,$privs_1$> <O_2,$privs_2$>…
Review rights for object O	Easy! Print the list.	Hard. Need to scan all principals' lists.
Review rights for principal P across all objects	Hard. Need to scan all objects' lists.	Easy! Print the c-list.
Revocation	Easy! Delete P from O's list.	If kernel tracks capabilities, invalidates on revocation. Harder if object tracks revocation list.

Authentication

Establish the identity of user/machine by

- Something you are:

  retinal scan, fingerprint

- Something you have:

  physical key, ticket, credit card, smart card

- Something you know:

  password, secret, answers to security questions, PIN

In the case of an OS this is done during login

- OS wants to know who the user is

Passwords

Secret known only to the subject

Top 10 passwords in 2017: [SsplashData]

1. 123456
2. password
3. 12345678
4. qwerty
5. 12345
6. 123456789
7. letmein
8. 1234567
9. football
10. iloveyou

16: starwars, 18: dragon, 27: jordan23

Top 20 passwords suffice to compromise 10% of accounts [Skyhigh Networks]

Verifying Passwords

How does OS know that the password is correct?

Simplest implementation:

- OS keeps a file with ⟨login, password⟩ pairs
- User types password
- OS looks for a login→ password match

Goal: make this scheme as secure as possible

- display the password when being typed?

Storing Passwords

1. Store username/password in a file
   - Attacker only needs to read the password file
   - Security of system now depends on protection of this file!
     Need: perfect authorization & trusted system administrators

Storing Passwords

1. Store username/password in a file
   - Attacker only needs to read the password file
   - Security of system now depends on protection of this file!
     Need: perfect authorization & trusted system administrators

2. Store login/encrypted password in file
   - Access to password file ≠ access to passwords

Storing Passwords

1. Store username/password in a file
   - Attacker only needs to read the password file
   - Security of system now depends on protection of this file!
     Need: perfect authorization & trusted system administrators

2. Store login/encrypted password in file
   - Access to password file ≠ access to passwords

3. Hashed Passwords

Hashing

Want a function f such that:

1. Easy to compute and store h(p)
2. Hard to compute p given h(p)
3. Hard to find q such that q ≠ p, h(q) = h(p)

Cryptographic hash functions to the rescue!

h(password) = encrypted-password *e.g.*, SHA

(but don't use SHA…)

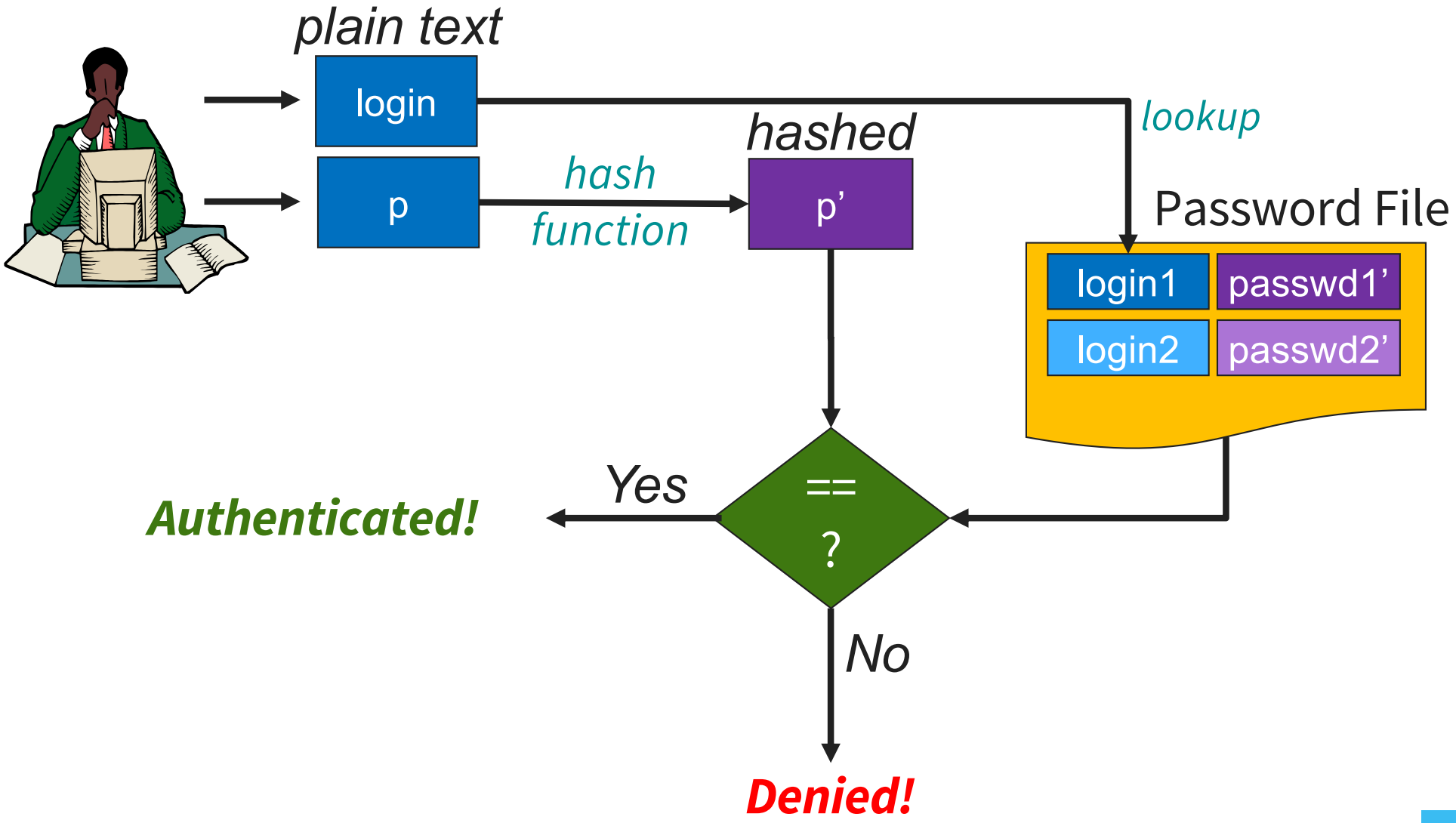
- *one-way property* gives (1) and (2)
- *collision resistance* gives (3)

Remember: *h(encrypted-password) ≠ password*

h^{-1}*(encrypted-password) = password*

h^{-1} hard to compute (hard ≈ impossible)

Storing and Checking Passwords

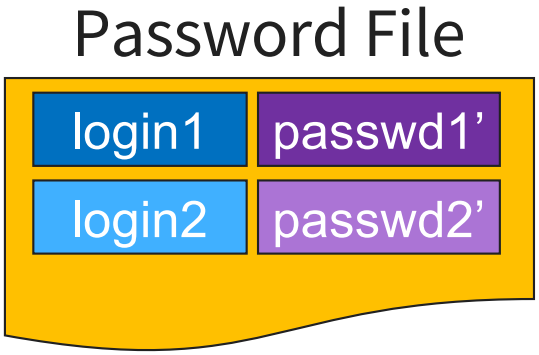


Hashed passwords still vulnerable

Suppose attacker obtains password file:

/etc/passwd public, known hash fn known

+ hard to invert → hard to obtain **all** the passwords



How else can I crack this file?

- Brute Force Attack:
  - Enumerate all possible passwords p, calculate h(p) and see if it matches an entry in the file
- Dictionary Attack
  - List all the likely passwords p, calculate h(p) and check for a match. (recall: top 20 passwords can compromise 10% of accounts)

Salting

Vulnerabilities:

- single dictionary compromises all users
- passwords chosen from small space

Countermeasure: include a unique system-chosen nonce as part of each user's password

- make every user's stored hashed password different, even if they chose the same password
- now passwords come from a larger space

Each user has: **login**, unique salt **s**, passwd **p**
System stores: login, s, H(s, p)

Salting Example

login	salt	h(p||s)
abc123	4238	h(423812345)
abc124	2918	h(2918password)
abc125	6902	h(6902LordByron)
abc126	1694	h(1694qwerty)
abc127	1092	h(109212345)
abc128	9763	h(97636%%TaeFF)
abc129	2020	h(2020letmein)

- If the hacker guesses qwerty, has to try: h(0001qwerty), h(0002qwerty), h(0003qwerty) …
- UNIX adds 12-bit of salt
- Also, passwords should be secure:
  - Length, case, digits, not from dictionary
  - Can be imposed by the OS! This has its own tradeoffs