Lecture 17: Authentication and Access Control

Stephen Huang

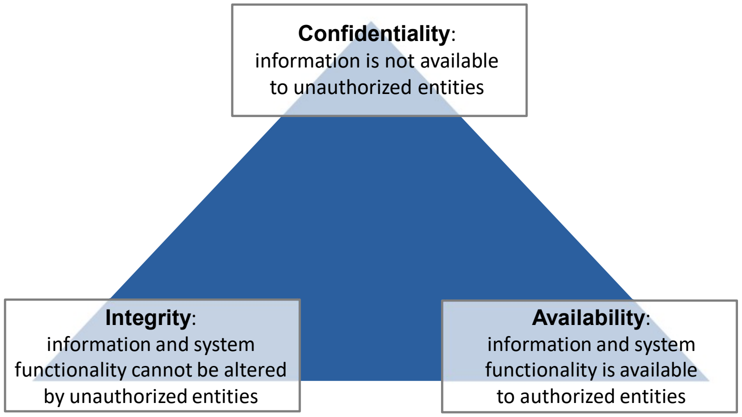
Content

1. Authentication
2. Access Control (Authorization)
3. Unix Access Control

Vulnerabilities

Vulnerabilities

Reminder: Security Objectives



Vulnerabilities

- Authentication: confirming the identity of a user or a host
- Authorization: specifying access rights/privileges to resources

access control in a narrow sense

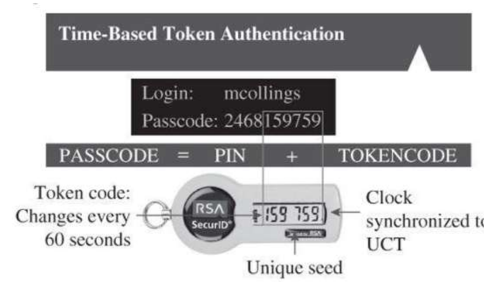
1. Authentication

Authentication: reliably verifying the identity of someone or something
- computer authenticates another computer
- computer authenticates a user

- Typical methods of computer authentication
  - cryptography-based (*example:* using the Kerberos protocol or using public-key certificates)
  - address-based (*example:* identifying a computer based on its IP address)
- Types of authentication
  - one-way authentication or mutual authentication
  - one-time or establishing a session (*e.g.*, combined with key exchange)

Authentication Mechanisms

Authentication is based on something you know, are, or have.
- Something you know: password, PIN, phrase, facts
- Something you are: Biometrics such as fingerprint, voice, faces
- Something you have:
  - Static tokens such as ID badges, physical keys, chips (on credit card),
  - Dynamic tokens such as SecureID Token



User Authentication

- Types of user authentication factors
  - knowledge: some secret known only by the user (*e.g.*, password)
  - ownership: some physical object possessed by the user (*e.g.*, bank card)
  - inherence: some physical characteristic of the user (*e.g.*, fingerprint)
- Password-based user authentication
  - typical form of knowledge-based authentication
  - verifier stores the password in a database or file
  - often combined with cryptography-based approaches to protect the password from eavesdropping
  - password must be easy to remember but hard to guess

Password-Based Authentication

- *Problem:* easy-to-remember passwords are weak
  - Miller's law: the number of objects an average human can hold in working memory is 7 ± 2.
  - Published in 1956 by cognitive psychologist George A. Miller ("The Magical Number Seven, Plus or Minus Tw The o…")
  - The length of passwords that users can easily remember (i.e., not write down somewhere) is very limited
    - *example:* 8 alphanumeric characters → 36^8 possibilities ~ 2^{48} possibilities
      → brute-force guessing may be computationally feasible
    - most popular passwords of 2019 (according to *SplashData*):
      1. `123456` 2. `123456789` 3. `qwerty` 4. `password` 5. `1234567`

Brute-force Attack

- Brute-force attack: password guessing
  - online: attacker must rely on the verifier to test the correctness of a password
    → verifier can limit the number of attempts (e.g., number of unsuccessful login)
  - offline: attacker can test the correctness of a password on its own

Password Storage

- Cleartext passwords are insecure
  - system administrators (and other local users) may easily read passwords
  - attackers who have compromised a system may be able to read passwords
- Example incident: Yahoo data breach
  - in September 2016, Yahoo announced that hackers breached its system sometime in late 2014
  - hackers accessed personal information (e.g., names, e-mail addresses, dates of birth, …) associated with 500 million Yahoo! user accounts
- Users tend to reuse passwords
  → breach may affect other systems as well

Storing Hashed Passwords

- Store the cryptographic hash of the password
  - during authentication, the user enters the plaintext password, and the verifier computes its hash and compares it with the stored hash
  - The attacker can perform offline guessing to recover the plaintext password
- Example: Unix systems
  - on modern systems, hashed passwords are stored in `/etc/shadow`, which can be read only by the root user
  - non-sensitive information is stored in the file `/etc/passwd`, which is readable by all local users
- Brute-forcing multiple hashed passwords
  - first, precompute a table of [password, hash] values for possible passwords
  - second, for each hashed password, look up the precomputed hash value

Precomputed Hash Chains

- Lookup Table of all possible passwords.

```
4a7d1ed414474e4033ac29ccb8653d9b:0000
25bbdcd06c32d477f7fa1c3e4a91b032:0001
fcd04e26e900e94b9ed6dd604fed2b64:0002
...
fa246d0262c3925617b0c72bb20eeb1d:9999
```

- Attacker's problem: The list of possible passwords is too long, which means prohibitive space requirement for storing precomputed hashes.
- Precomputed hash chain: trading off <u>space</u> for running <u>time</u>.

Building Hash Chains

- Reduction function R: maps a hash value to a possible password (**not** the actual inverse of the hash function)
- Choose a random set of initial passwords (*e.g.*, `aaaaaa`, `bbbbbb`, …)
- For each password,
  - compute a chain of passwords and hash values by alternating between using the hash and reduction functions, *e.g.*:

$$\texttt{aaaaaa} \xrightarrow[H]{} \texttt{281DAF40} \xrightarrow[R]{} \texttt{sgfnyd} \xrightarrow[H]{} \texttt{920ECF10} \xrightarrow[R]{} \texttt{kiebgt}$$

  - store the initial and final passwords (e.g., (`aaaaaa`, `kiebgt`)) in a table.
- The reduce function is essentially a deterministic pseudo-random password generator.

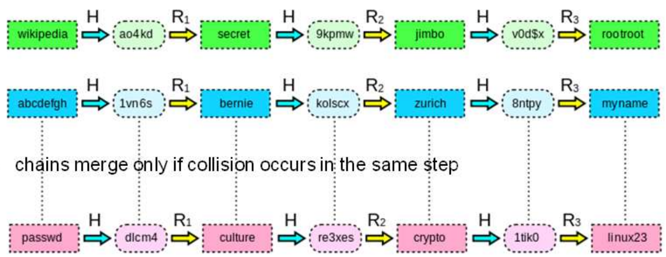
Recovering a Password

- To recover a hashed password, start building a chain from the hash value, and test if any of the resulting passwords is among the stored final passwords
  - *e.g.*, if hash is `920ECF10`, then build chain to `kiebgt`
- When a match is found, the correct password can be recovered from the chain
  - *e.g.*, from stored initial password `aaaaaa`, we can build chain to `sgfnyd`

$$\texttt{aaaaaa} \xrightarrow[H]{} \texttt{281DAF40} \xrightarrow[R]{} \texttt{sgfnyd} \xrightarrow[H]{} \texttt{920ECF10} \xrightarrow[R]{} \texttt{kiebgt}$$

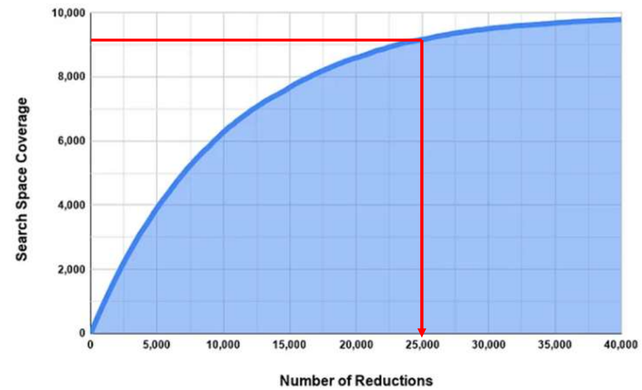
Rainbow Table

- Hash chain limitation: when hash or password values collide, the remainder of the chains (including the final value) are the same
  - number of usable chains is limited
- Rainbow table
  - to prevent merging chains, use a sequence of reduction functions $R_1, \ldots, R_k$



  - chains merge only if collision occurs in the same step

Probability of a collision



Rainbow Table

- When one value collides, all subsequent values collide, too. Merging.
- It gets harder and harder to avoid a collision.
- We cannot efficiently detect the collision at generation time since that would require holding on to the full contents of the previous chains.
- The solution to merging is to use a sequence of reduce functions R_i in Step i.
- A practical implementation of a sequence of reduce function is to add a second parameter of "salt".
- Merges are still possible, but only if the collision "lines up".

Salting

- Before hashing a password, mix it with a salt value
  - both when the password is set and during verification
  - verifier stores: username, salt, H(password + salt)
- Salt value
  - randomly generated for each user account
  - may be stored in plaintext by the verifier
- Salt values do not have to be memorized → strong randomness
  - prevents precomputing hashes since the attacker cannot consider all possible salt values (different salt values require different precomputation)
  - also hides identical passwords, which would result in identical hashes
- However, it does not make guessing a single password harder (assuming that the attacker knows the salt)

Salting Issues

- Example: 2012 LinkedIn hack
  - on June 5, LinkedIn was breached, and around 6.5 million hashed passwords were stolen
  - on June 6, a large number of recovered plaintext passwords were posted online
  - in May 2016, it was discovered that an additional 100 million might have been compromised in the incident
  - Major weaknesses:
    - passwords were not salted before hashing
    - passwords were hashed using SHA-1

- SHA-1 is relatively easy to compute
  → brute-force guessing is relatively fast

Rainbow Tables

- Rainbow tables aren't popular anymore.
- Brute-force attacks aren't that effective.
- Secured hashing is immune since it uses salting.
- The best rainbow tables publicly available only go up to 8 characters for a full character set.
- The average password length is over 9 characters,
- Other methods are better. Wordlist attacks with manipulation rules are far more effective at getting actual user-picked passwords.

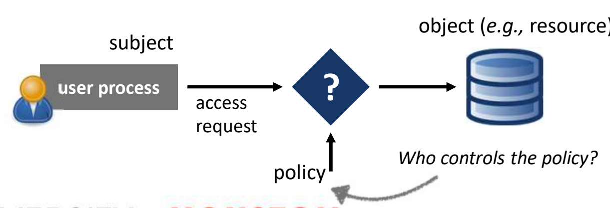
Multi-Factor Authentication

- User is authenticated only after passing multiple independent authentication mechanisms
  - typically, each mechanism is built on a different type of factor (e.g., knowledge + possession), so it is independent of the other mechanisms
  - The attacker must circumvent all authentication mechanisms to succeed
- Possession factors
  - disconnected token: not connected to the client computer, typically, the user manually enters authentication data displayed by the token
  - connected token: physically connected to the client computer (e.g., USB token)
- Inherence factors
  - includes fingerprint, face, voice, or iris recognition

example:
RSA SecureID token

2. Access Control - Authorization

- Access control (i.e., authorization): approving or rejecting access requests.
- Abstractions
  - subjects: entities that can perform actions on the system
  - objects: resources to which access must be controlled
- Control access to objects based on a policy



Discretionary Access Control (DAC)

- Allows access rights to be propagated at the subjects' discretion
- Often implemented using the notion of owner
  - every object has an owner subject, who can set the permissions for that object
- Used by popular operating systems (*e.g.,* Unix and Windows)

Problem: non-malicious users are not necessarily trustworthy

- phishing: subjects may be tricked into propagating their access rights to malicious entities
- malware: malicious code running with a subject's credentials can disclose or modify sensitive information

→ large organizations working with sensitive data may need centralized control

Mandatory Access Control (MAC)

- Restricts the access of subjects to objects based on a system-wide set of rules
  - system-wide rules are set by a central authority (e.g., system administrator)
  - policy is mandatory → users do not have full control over access to the resources that they create
- Traditionally used for implementing multilevel security
  - objects have security classifications (e.g., "Top Secret", "Secret")
  - subjects have security clearances
- Available in some form on many modern operating systems
  - SELinux and AppArmor for Linux, and Mandatory Integrity Control for Windows
- May be combined with DAC: grant access only if both DAC and MAC permit the access

Access Control Models

Access matrix: → Objects →

Subjects ↓

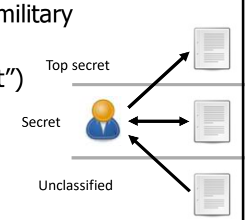
r w		r		r
r	r	r	r	
r		r w		r w
r w	r	r w		r
w	w	r w	w	r w

access rights (*e.g.,* read and write)

- Access control list (ACL): list permissions for each object
  - for each object, list pairs of [subject, access right]
- Role-based access control (RBAC): row oriented
  - create a set of roles (e.g., based on real-world job functions), and assign a role (or roles) to each subject
  - for each role, list pairs of [object, access right]

Bell-LaPadula Model

- Developed by D. Bell and L. LaPadula in the 1970s for enforcing access control in government and military applications
- Multilevel security (*e.g.,* "Top Secret", "Secret")
  - objects have security classifications
  - subjects have security clearances

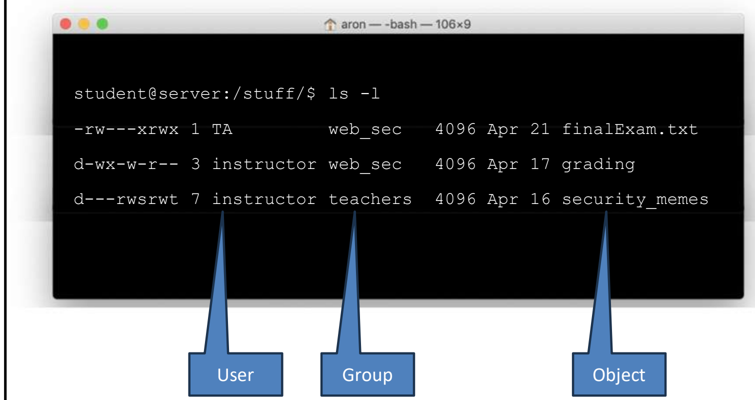


- Focuses on confidentiality
- Rules
  1. **simple security property**: subjects cannot read objects at a higher security level
  2. ***-property**: subjects cannot write to objects at a lower security level
  3. **discretionary security property**: use an access matrix to specify DAC
- Information may be transferred from a higher level to a lower level by trusted subjects

3. Unix Access Control

- Basic Concepts
  - user: has a unique UID (special UID = 0 for root user)
  - group (collection of multiple users): has a unique GID
- Access control abstraction
  - subject = process
    - has an effective UID and GID (as well as real and saved UIDs and GIDs)
  - object = file
    - has an owner (UID) and a group (GID), typically inherited from the process that created the file
    - almost everything is a file on a Unix system (regular files, directories, devices, Unix domain sockets, …)

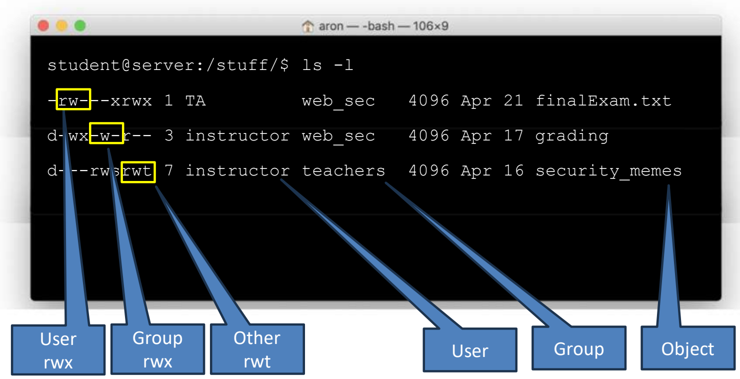
Example



Unix Access Control: Permission

- Each file has 12 permission bits
  - **read**, **write**, and **execute** permission for owner, group, and others
  - set user ID (setuid), set group ID (setgid), sticky bits
- When a process wants to read/write/execute a file,
  1. if effective UID = file owner → use read/write/execute permission for owner
  2. else if effective GID = file group → use read/write/execute permission for group
  3. else → use read/write/execute permission for others
- For directories,
  - read means listing the contents of the directory
  - write means creating, renaming, and deleting files in the directory
  - execute means accessing the files (and directories) within the directory (must also have execute permission on all the parent directories)

Example

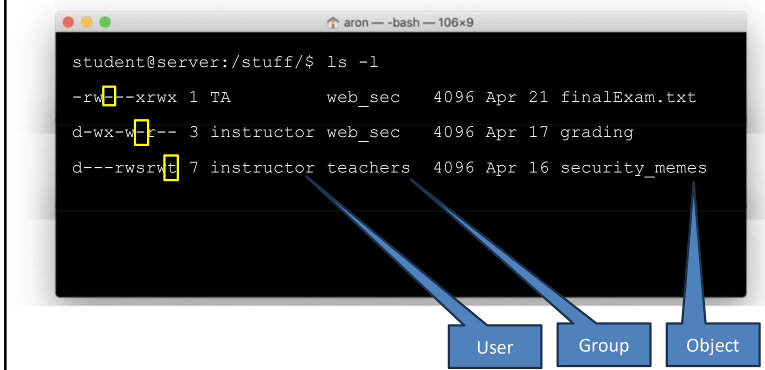


The sticky bit should have been called the "restricted deletion (by owner) bit".

Sticky, Set UID, and Set GID Bits

- Sticky bit
  - when set on a directory, files within that directory can be renamed or deleted only by their owners, the directory owner, or a superuser
  - for example, sticky bit is typically used on the `/tmp` directory
- Set UID bit
  - when set on an executable file, the effective UID of a process executing the file is set to the file owner UID
  - for example, set UID bit is typically used on the `passwd` command
- Set GID bit
  - when set on an executable file, the effective GID of a process executing the file is set to the file group GID
  - when set on a directory, new files created within will inherit the GID of the directory

Example



Unix Access Control Conclusion

- Processes running with `setuid`/`setgid`
  - effective UID/GID is the UID/GID of the executable file, while the real UID/GID is the UID/GID of the parent process
- Changing the owner or group of a file
  - only a superuser can change ownership
  - only a member of a group can change the group of a file to that group
- Traditional Unix access control is DAC / ACL
- Some Unix versions offer support for other policies
  - SELinx (Security Enhanced Linux): support for RBAC or MAC
  - Oracle Solaris: support for RBAC

Example

```
cooluser@LAPTOP-5V55HON5:/topsecretfolder$ ls -l
total 8
drwxrwxr-x 2 root root 4096 Oct 18 19:34 secret1
drwxrwx-   2 root root 4096 Oct 18 19:31 secret2
cooluser@LAPTOP-5V55HON5:/topsecretfolder$
```

```
cooluser@LAPTOP-5V55HON5:/topsecretfolder$ ls secret1
pepperNeggRecipe.txt
cooluser@LAPTOP-5V55HON5:/topsecretfolder2$
```

```
cooluser@LAPTOP-5V55HON5:/topsecretfolder$ ls secret2
ls: cannot open directory 'secret2': Permission denied
cooluser@LAPTOP-5V55HON5:/topsecretfolder3$
```

```
root@LAPTOP-5V55HON5:/topsecretfolder#     ls secret2
GiardinieraRecipe.txt
root@LAPTOP-5V55HON5:/topsecretfolder#
```

Next Topic

- Authentication and Access Control
- Software Security