Intro to IT Security

CS306C—Fall 2022

Prof. Antonio R. Nicolosi

Antonio.Nicolosi@stevens.edu

Authentication & Authorization (Access Control)

Setup

- Let A, B denote two entities
- A will act as the **prover** (trying to prove her identity)
- B will act as the **verifier**

Generalities

- Suppose that A wants to authenticate herself to B
- This must be accomplished through some kernel of trust, based upon prior knowledge, *e.g.,*
  - a trusted public key
  - a shared secret

Authentication

The task of A proving to B that A is the owner of the corresponding private key or the shared secret, etc.

Ideally this proof should satisfy the following properties:

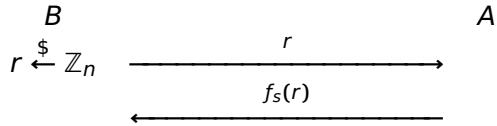
- Easy to create given the secret
- Hard to forge without the secret
- Remain hard to forge, no matter how many other proofs have been seen by an adversary

Authentication Via PRFs

Pseudo-Random Functions (PRFs) are a simple way to construct an authentication scheme.

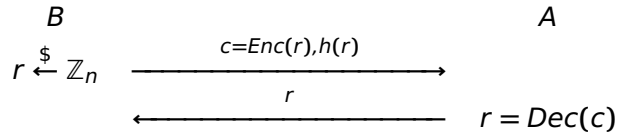
Setup:

- $\mathcal{F} = \{f_i : \mathbb{Z}_n \to \mathbb{Z}_m \mid i \in \{0,1\}^k\}$, a family of pseudorandom functions
- Shared secret: $s \in \{0,1\}^k$
- Algorithm:



Authentication via Public-Key Cryptography

If *Enc* denotes the encryption algorithm for an asymmetric scheme and if *h* denotes a cryptographic hash function, we can provide authentication as follows:



Types of Authentication

- End user → End user (e.g., Alice & Bob)
- End user → Local computer (e.g., login)
- End user → Remote computer (e.g., web site login)
- Computer → Computer (e.g., DRM)
- Local computer → End user (e.g., prevent fake ATM)
- Remote computer → End user (e.g., anti-phishing)

Passwords

- Passwords are a classic way to authenticate
- Advantages:
  - Familiar; common
  - Easy to use
  - Easy to remember (sometimes...)

Problems with Passwords

- If password is sent in the clear, can be intercepted
- If password is encrypted, requires establishment of encryption key
- People choose bad passwords (*e.g.,* not enough entropy)
- Passwords are easily observed (sniffed by spyware, or perhaps other humans)

Spy / Spyware Defenses

Use a randomized keypad; use mouse for input rather than keyboard

Storage of Plaintext Passwords

- If passwords are kept as plaintext, then a compromise of the system could be especially catastrophic
- Worse: Might be able to guess password in linear time!
  - Place first character of password as last byte on VM page
  - Make sure the next page is unmapped
  - Try all first letters, one by one
  - Got a page fault? Must have correct first letter
  - Repeat for second character of password, etc.

Hashed Passwords

One approach to mitigate the risks if the system is compromised is to hash the password first:

- The user computes and sends $h(pwd)$ for some one-way, collision-resistant hash h
- Operating system simply checks the hash

What could go wrong?

Attacks on Hashed Passwords

While better than storing passwords in the clear, this still leaves avenues for attacks.

Off-line Dictionary Attack:

- If h is known, one can compute $h(pwd)$ for a long list of common passwords
- If the file of hashed passwords is recovered, one can immediately check for matches and lookup the password

Example: Unix Hashed Passwords

- Store one-way function of password
  - Only hashes first 8 characters
  - Apply DES 25 times with password as key, starting from an all-zero input block
  - Key generally hard to recover from ciphertext
  - So put hashed password in world-readable `/etc/passwd`
  - To validate password, hash it and compare to stored hash
- Hash function "salted" with 12 extra bits for each user
  - Prevent attacker from building dictionary of hashes of common passwords
  - Prepend seed to hashed password for use in verification

Example: Unix Hashed Passwords

- Store one-way function of password
  - Only hashes first 8 characters
  - Apply DES 25 times with password as key, starting from an all-zero input block
  - Key generally hard to recover from ciphertext
  - So put hashed password in world-readable `/etc/passwd`
  - To validate password, hash it and compare to stored hash
- Hash function “salted” with 12 extra bits for each user
  - Prevent attacker from building dictionary of hashes of common passwords
  - Prepend seed to hashed password for use in verification

Unix Hashed Passwords: Weaknesses

- Off-line password guessing attacks
  - Attacker gets password file, guesses passwords at home
  - Computationally expensive, but undetectable by victims
- Fails to account for hardware & software improvements
  - DES `crypt` on 1977 VAX-11/780: 4 crypts/sec
  - Bitsliced implementation on 600 MHz alpha: 214,000 cr/sec
  - But users don't chose better passwords now than in 1977...
- Server still gets plaintext password at login
  - Attacker or server operator can modify login to record it
  - But users often use same password on different systems

Systems with passwords: Design Hints

- Any algorithm that uses passwords should:
  - Take a salt as input (prevent hash dictionaries)
  - Take a cost parameter as input
- Administrators need to increase hashing cost as hardware gets faster

Systems with passwords: Examples

- Bad examples: PGP & ssh private keys
  - Stored in user's home directory, encrypted with password
  - Given encrypted private key, can guess password off-line
  - Cost of guess even cheaper than password `crypt`!
- Good example: OpenBSD/FreeBSD `bcrypt`
  - Hashing password is exponential in cost parameter
  - Cost goes in hashed password along with salt

Network Passwords

- Many systems grant access through a password
- How to implement? Example:
  - Server stores user's password (or hash)
  - Client connects to server, sends username, password
  - Server compares password to stored version
  - Grants access if they match
- Is this a good approach?

Weaknesses

- How do you know you are talking only to the server?
  - Attacker might be eavesdropping on Ethernet
  - Attacker might mess with DNS (or IP routing) so you talk to wrong machine
- Eavesdropper will just read your password off the network
- Server knows your plaintext password
  - Or at least sees it at login time, even if stores hash
  - Extra-bad if people re-use passwords on multiple machines

s/key Password Authentication

- Goal: Protect against passive eavesdroppers
  - Also: Minimize harm of bad clients (e.g., public terminals)
- Idea: One time passwords, not valid after snooped
- Algorithm takes user's real password, p, a random "salt" s, and server machine name m
  - First one time password is $H^{(100)}(p, m, s)$ (for one way hash H)
  - Next password is $H^{(99)}(p, m, s)$, etc.
  - After 99 logins, must change salt or password
- Benefit: Very convenient
  - Carry list of one-time passwords or calculate on smartphone

Weaknesses

- Vulnerable to network attackers
  - Attacker impersonates server
  - Re-sends one-time password to real server
- Vulnerable to off-line password guessing
  - Attacker sees s and $H^{(n)}(p, m, s)$
  - Can verify guesses of p off-line—check against dictionary of common passwords
  - H not very expensive (should be $H^{(n)}(G(p, m, s))$ for expensive G—or perhaps increase n)
- Bad client can compromise session
  - Before logout, can insert command to create back door:
    `echo ssh-rsa AAAA...Fbg== badguy » .ssh/authorized_keys`

Password-derived public keys

- Derive public key from user's password
  - E.g., use password as seed for pseudo-random generator to generate (public key, private key) pair
  - Client can regenerate private key given password
- Server stores public key for each user
- Use public key authentication, E.g.:
  - $S \rightarrow C : N_S$
  - $C \rightarrow S : K_u, \{C, S, K_S, u, session, N_S\}_{K_u^{-1}}$
  - K_S is server public key, used to authenticate server
  - S looks up public key K_u for credentials and checks signature

Weaknesses

- No salt
  - Users with same password will have same (public key, private key) pair
- No cost parameter
  - Can’t take too long to log in
  - But over time generating key will get faster
- Public key is just like password hash
  - Eavesdropper will see key, can mount off-line attack
- No authentication of server to user
  - W/o accessing server, attacker can pretend server is giving bad answers

Just For Fun

Ever wondered how garage door openers work, or the remote control for the car door locks?

- PRF authentication would be nice, but it requires interaction...
- Building a transmitter into the car and a receiver into the remote seems like too much effort
- How to avoid it?

Non-Interactive Protocol

- Both the transmitter and receiver keep track of an index k
- Transmitter just sends PRF values in sequence: sends $a = f_s(k)$ and then increments k
- Transmitter verifies that $a = f_s(k)$: if so opens door and increments k

Problem: what happens if you hit the remote, but the receiver doesn't hear you?

Non-Interactive Protocol

To make the non-interactive protocol more usable, the designers sacrificed a little security:

- Both the transmitter and receiver keep track of an index k
- Transmitter again sends $a = f_s(k)$ and increments k
- Receiver checks to see if
$$a \in \{f_s(k+i)\}_{i=0}^{255}$$
- If so, then set $k = k + i$ where $a = f_s(k+i)$

Access Control

- Protected Entity: “object” O
- Active object: “subject” S
- An object can be
  - File
  - Directory of files
  - Executing program
  - Database entry
  - Data structure
  - System resource (printer, ...)

General Principles

- Fine-grained access control is desirable
  - *E.g.*, control access to files not just directories
  - More powerful, but of course more complex
- Least privilege
  - Grant minimum abilities necessary to complete task
- Closed *vs.*. open policies (a.k.a. default-on *vs.* default-off)
  - Closed: forbidden unless explicitly allowed
  - Open: allowed unless explicitly forbidden
- Conflict resolution
  - Prevent conflicts, or know how to deal with them

Access Control Policies

- Types of Policies
- Implementations

Access Control Policies

- Discretionary Access Control (DAC)
- Mandatory Access Control (MAC)
- Role-Based Access Control (RBAC)
- Not necessarily mutually exclusive
  - Can use different mechanisms for different resources
  - Can apply two policies, and allow access only if both allow

Discretionary Access Control (DAC)

- Control based on the identity of the user and access rules
- Rights can be delegated at user's *discretion*
- Most common (*e.g.,* user, group, owner permissions on Unix)

Mandatory Access Control (MAC)

- Control over the policy is centralized
- Control based on comparing security labels with security clearances
- Delegation not allowed
- Originated in military applications
- Recently integrated into commodity systems (*e.g.,* the “Mandatory Integrity Control” in Windows version > 6)

Role-Based Access Control (RBAC)

- Control based on user's (or program's) role (rather than their identity alone)
- User's right can change depending on their current role
- More recent proposal; adds flexibility and ease
- One could construct equivalent MAC structures, but RBAC is more naturally organized
- Examples include
  - Numerous database systems
  - FreeBSD
  - Microsoft Active Directory

DAC: Means of Implementation

- Access Control Matrix
- Access Control List
- Capabilities

Access Control Structure: An Analogy

- Think of the setup as a bipartite graph with labeled edges
  - The left vertexes are subjects, the right are objects and the edges (labeled) define the permissions
- Two common ways to represent graphs also apply to representing a control structure, and some of the same considerations apply
  - Matrix representation
  - List representation

Access Control Matrix

- Matrix indexed by all subjects and objects
  - Characterizes rights of each subject with respect to each object
- Formally: set of object O and subject S, set of possibles rights
  - Objects: label columns of matrix
  - Subjects: label rows of matrix
  - Entry *(s, o)* contains the right for *s* to *o*
  - *E.g.*, read/write/execute

Access Control Matrix: Extensions

- Rights can be functions
  - "Actual" right depends on the system state
  - Equivalently, may depend on system history
- Rights can form hierarchies
  - *E.g.*, right X implies right Y
- How fine-grained the access control is depends on how fine-grained the rights are

Access Control Matrix: Drawbacks

- Number of subjects/objects is very large
- Matrix is usually sparse (most entries empty or contain some default value)
- One central matrix modified every time subjects/objects are created/deleted or rights are modified
- "Small" change can result in "many" changes to the access control matrix
  - *E.g.*, making a file publicly readable
- No support for dynamic access control
  - Does not describe if/how permissions are allowed to change

Access Control List (ACL)

- One access control list per object
- Can be viewed as storing the columns of the access control matrix with the appropriate object
- One list per object showing all subjects with access and their rights
- Possible to assign “default” rights to an object
  - Easy to make an object public
- Example: access based on user, group, and compartment
  - Use of wild cards
  - To specify a group of hosts, a 1 is used
  - To specify a full subnet with a wildcard: 172.16.30.0 0.0.0.255
    - This tells the router to match up the first three octets exactly, but the fourth octet can be any value

ACL in practice

- Full granularity may not be supported
  - *E.g.*, user-level, group-level, or only public/private?

Conflicts

- How to handle conflicts if two subjects give different permissions on an object
  - Disallow multiple owners
  - Allow access if any entry gives rights
  - Allow access only if no entry denies rights
  - Apply first applicable entry

Capabilities

- Can be viewed as storing the rows of the access control matrix with the appropriate subject
- Some burden for implementing protection placed on the user rather than just the OS
  - Analogy: user has a “ticket” which grants access to an object
  - A capability is an unforgeable token giving user access to an object and describing the level of allowable access
  - Capabilities can specify new types of rights

Capabilities: Two Approaches

- Ticket is held by OS, which returns to the subject a pointer to the ticket
- Ticket is held by the user, but protected from forgery by cryptographic mechanisms
  - How . . . ?
  - Two possibilities: ticket verified by the object or by the OS itself
  - Who holds the key in each case . . . ?

ACL vs. Capabilities

- Access control list
  - ACL associated with each object
  - Upon request, check user/group against the ACL
  - Relies on authentication of the user
- Capabilities
  - Can be passed from one user/process to another
  - Upon request, check validity of capability
  - No need to know the identity of the user/process making the request

ACL vs. Capabilities

- How would delegation be handled using ACLs vs. using capabilities?
  - ACL: run process using the name of the caller, or OS can explicitly support delegation
  - Capabilities: allow delegation “for free”
- How can a user make a file public when capabilities are used?
- How to revoke capabilities?

Capabilities: Advantages

- Better at enforcing “principle of least privilege”
  - Provide access to minimal resources, to the minimal set of subjects
- Allow much finer-grained control over subjects (process-level instead of user-level)

Capabilities: Disadvantages

- Overhead
- Revocation more difficult
- Controlling delegation more difficult
- Making files world-readable more difficult (impossible?)

MAC: Military Security Policy

- Primarily concerned with secrecy
- Objects given “classification” (rank; compartments)
- Subjects given “clearance” (rank; compartments)
- “Need to know” basis
  - Subject with clearance (r, C) dominates object with classification (r', C') only if $r \geq r'$ and $C' \subseteq C$
  - Classifications/clearance not necessarily hierarchical

Bell-LaPadula model

- Simple security condition: S can read O if and only if $l_o \leq l_s$
- *-property: S can write O if and only if $l_s \leq l_o$
  - Why?
- “Read down; write up”
  - Information flows upward

Dynamic Rights

- Could consider dynamic rights
  - Once a process reads a file at one security level, cannot write to any file at a lower security level

Basic Security Theorem

- If a system begins in a secure state, and always preserves the simple security condition and the *-property, then the system will always remain in a secure state
  - *I.e.*, information never flows down

Communicating Down

- How to communicate from a higher security level to a lower one?
- Dynamic rights
  - Max. security level *vs.* current security level
  - Maximum security level must always dominate the current security level
- Declassification
  - Reduce security level to write down
  - Security theorem no longer holds
  - Must rely on users to be security-conscious

Role-Based Access Control (RBAC)

- Access controls assigned based on *roles*
  - Can use an access matrix, where “subjects” are roles
- Users assigned to different roles
  - Can be static or dynamic
  - A user can have multiple roles assigned
  - Can use “access matrix” with users as rows, and roles as columns
  - Will, in general, be more compact than a full-blown access control matrix