ECE/CS230
Computer Systems Security

Charalambos (Harrys) Konstantinou

https://sites.google.com/view/ececs230kaust/

Basic tools in computer security (authentication, access control)

Authentication Basics

- Authentication binds identity to a subject
- Two step process
  - Identification - establish identity to system
  - Verification - process verifies and binds entity and identity

Password Authentication Basics

- User keeps a secret string (password)
- Something the user *knows*
- Advantages?
- Disadvantages?

Attacks

- Steal from the user
  - Install a keylogger (hardware or software)
  - Find it written down
  - Social engineering/Phishing
  - Intercept the password over network
  - Use a side channel
- Steal from the service
  - Install malware on the web server
  - Dump the password database with SQL injection
- Steal from a third party (password reuse)

Secure Passwords

- Uneven distribution makes guessing easier
- Passwords should be uniformly distributed
  - All characters in password chosen with equal probability
- Passwords should be long
  - Longer password = larger brute force search space
- Passwords should never be reused
- Passwords chosen randomly are difficult to remember
  - Tradeoff of security vs. convenience

Storing Passwords

- Password database is highly sensitive
- We should ***never*** store *plaintext* passwords
- Store something that lets user prove they know the password

Passwords

Hash functions

- Input – data of an arbitrary size
- Output – fixed length
- Same input always produces the same output
- One way function – cannot deduce input from output
- A “fingerprint” for the input
- Examples: ~~MD5, SHA-1~~, SHA-256, SHA-512, SHA-3
    - md5("welcome")= 40be4e59b9a2a2b5dffb918c0e86b3d7
- ***None of these should be used directly used for password hashing***

Passwords

Noncryptographic hash functions (and more)

- Cyclic redundancy checks (CRC)
  – CRC-16, CRC-32, etc.
  – Based on polynomials, many variants
- Checksums
  – parity word, sum-16, Adler-32, Luhn alg., etc.
- Noncryptographic hash functions
  – FNV-1, Berstein hash (djb2), Java’s hashCode()
- ***None of these should be used used for password hashing***

Password Cracking

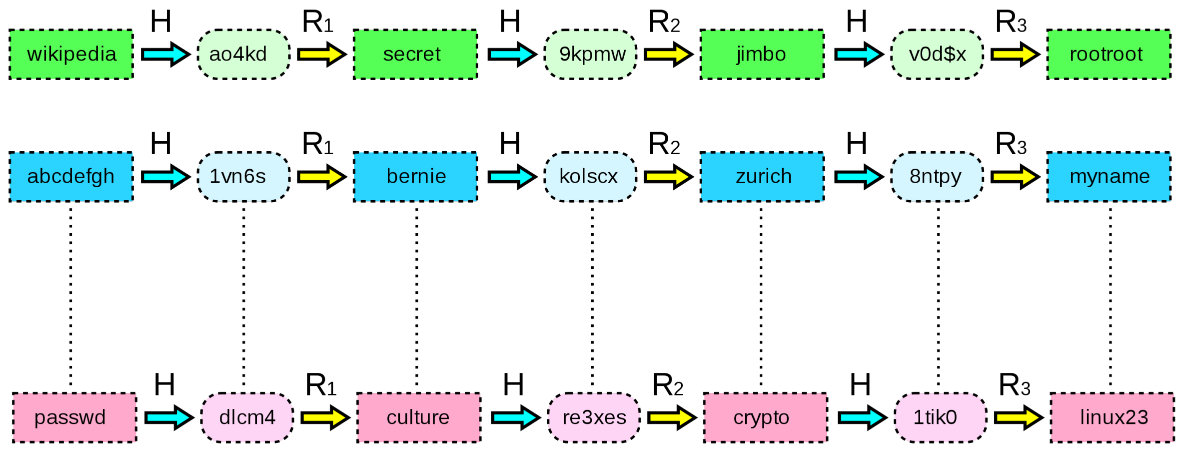
- **Brute force** search through all possible passwords in order
  - Computationally expensive and least efficient (cracked hashes per processor time)
  - Very successful on short and simple passwords

Password Cracking

- Brute force search through all possible passwords in order
- Use a **dictionary**
  - Use a dictionary of common passwords
  - Combine dictionary with common passwords and heuristics (e.g. p@$$w0rd and password123)
  - Use statistical models of user passwords
  - Easy to parallelize: hash password guess, compare to entire hash database
  - Commonly done with arrays of GPUs

Rainbow Tables

- A rainbow table is a precomputed table for reversing cryptographic hash functions, usually for cracking password hashes
  - Many passwords are common
  - Precompute them in a lookup table
  - Time/space tradeoff

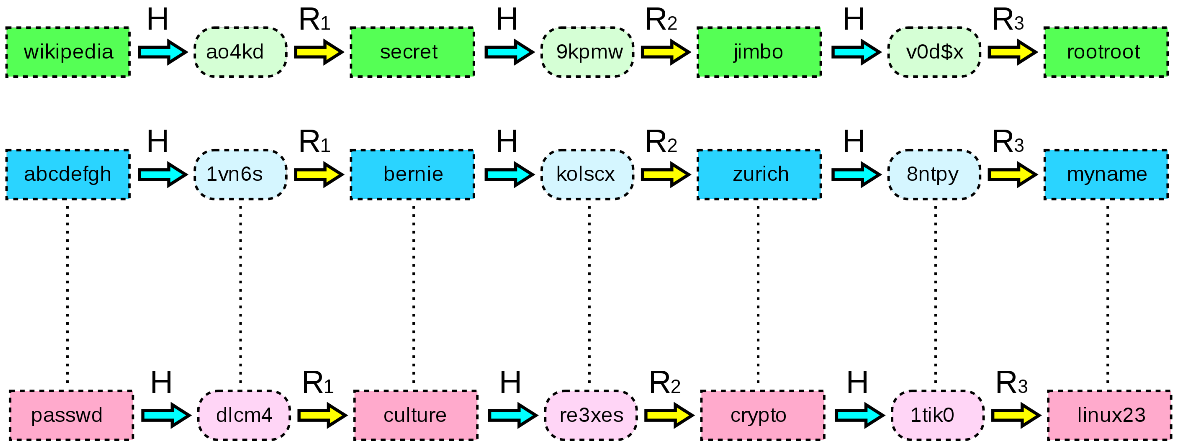


Rainbow Tables

- Suppose we have a password hash function H and a finite set of passwords P. The goal is to precompute a data structure that, given any output h of the hash function, can either locate an element p in P such that H(p) = h, or determine that there is no such p in P.
- The simplest way to do this is compute H(p) for all p in P, but then storing the table requires $\Theta(|P|_n)$ bits of space, where n is the size of an output of H, which is prohibitive for large $|P|$.

Rainbow Tables

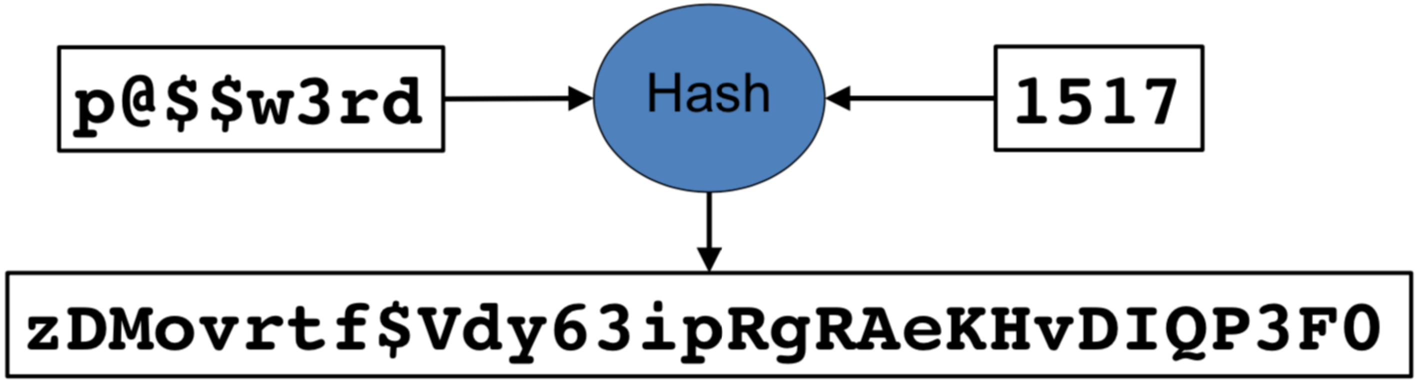
- Hash chains are a technique for decreasing this space requirement. The idea is to define a reduction function R that maps hash values back into values in P.
- Note, however, that the reduction function is not actually an inverse of the hash function, but rather a different function with a swapped domain and codomain of the hash function.
- By alternating the hash function with the reduction function, chains of alternating passwords and hash values are formed.



Defense against rainbow tables

Salting Password Database

- Generate and store a random number, the salt for each password
- Concatenate password and salt to compute hash
- Effectively a unique hash function for each password



Password Security Policies

- Educate users about password security
  - Specifically train them to use good passwords – But they might or might not follow through
- Generate passwords randomly
  - Perfect uniform distribution
  - But not very psychologically acceptable
- Reactive password checking
  - Crack your own user's passwords
  - But expensive and passwords vulnerable until cracked
- Complex password policy/proactive checking

Complex Password Policy/Proactive Checking

- Let the user select their own password
- Force them to follow a policy
- Reject passwords that don’t follow policy
- But...
  - Technically *reduces* number of possible passwords
  - Policy might not be psychologically acceptable
  - We don’t know if users are reusing their passwords

Password Managers

- Application that generates and maintains passwords
- Examples: LastPass, KeePass, DashLane, 1Password
- Advantages:
  - Can handle random passwords
  - **Can create unique passwords for every website and service**
- Disadvantages
  - One point of failure
  - Requires a strong password (could be snooped)
  - Could be hacked (only as secure as the password manager)
  - Inconvenient (doesn’t work for some sites, set up time, etc.)

Single Sign-On (SSO)

- Login to trusted 3rd party (identity provider), who vouches for user identity
- Examples: Facebook Connect, OAuth, OpenID
- Pros and cons similar to Password Managers
- Third party can track users...

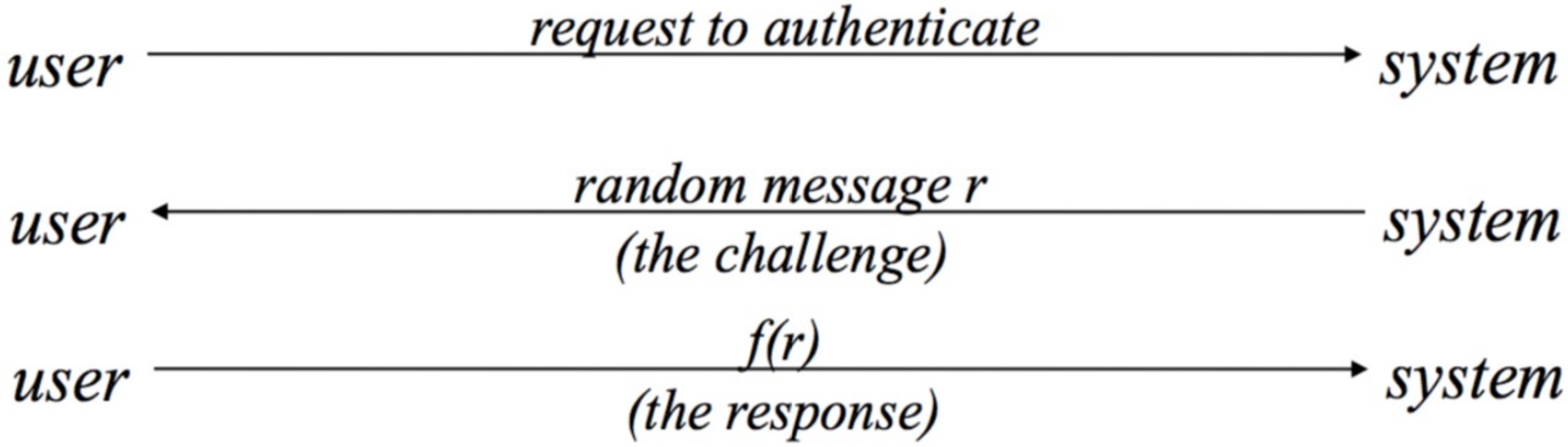
Token-based Authentication

Basics

- Something the user *has*
- Static memory cards
  - Read only
  - e.g. Credit Card
  - Vulnerable to replay attack
- Smart card
  - Storage and computation
  - Enables challenge-response or one-time password
  - Protects against replay attack

Token-based Authentication

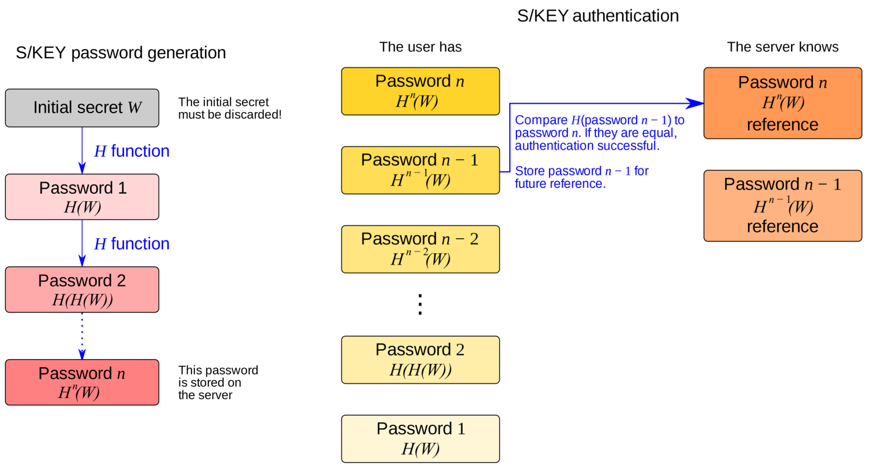
Challenge-Response



Token-based Authentication

One-time password (OTP)

- Smart card can also implement one-time password scheme
- S/Key is one such scheme:
  - Start with a random seed
  - Hash the current seed to produce the next
  - Use the hash outputs in reverse order
- Time-based one-time password (TOTP)
- Vulnerable to man-in-the-middle (MitM)

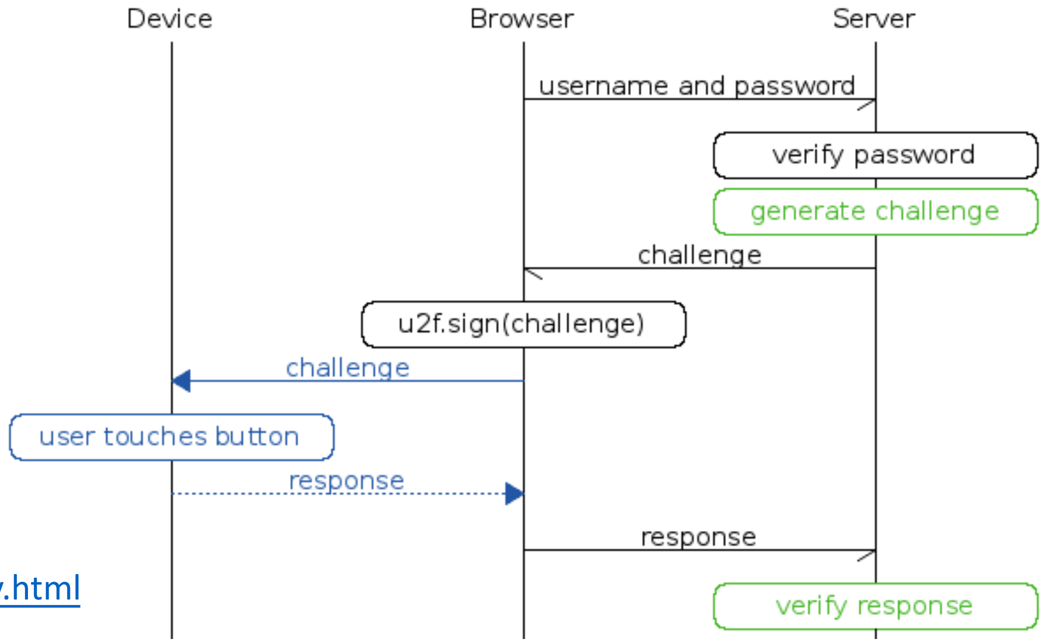


https://en.wikipedia.org/wiki/S/KEY

Token-based Authentication

Universal second factor (U2F)

- Addresses OTP's weakness to MitM
- Website's *origin* is cryptographically bound to the response (not displayed in the diagram)



https://developers.yubico.com/U2F/Libraries/Using_a_library.html

Token-based Authentication

Universal second factor (U2F)

- Disadvantages
  - Token can be lost, stolen, or counterfeited
  - Requires an individual physical token
  - Requires an extra step (mildly inconvenient)
  - Hardware can be expensive..
    - ..but usually isn't
    - $18 for U2F key from Yubico
    - Google, Facebook, and Yubico were all giving these away at a recent conference I attended

Biometric Authentication

- Something the user *is* or *does*
- Derive a signature from biological features of user
  - Voice, fingerprint, face, retina, handwriting, gait
- Advantages?
- Disadvantages?

Biometric Authentication

Disadvantages

- Imprecise measurements require *approximate* matching
  - Essentially a machine learning task
  - False negatives *and* false positives have a cost
- Measurements change over time
- Poor accessibility
- Cannot be replaced or concealed
- Replay attacks/spoofing possible
- Can be legally compelled to provide biometrics

Biometric Authentication

- **Office of Personnel Management (OPM) data breach (2015)**
- Among others: Theft of fingerprints
  - The stolen data included 5.6 million sets of fingerprints
  - Biometrics expert R. Kesanupalli said that because of this, secret agents were no longer safe, as they could be identified by their fingerprints, even if their names had been changed

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

Other Schemes: 2FA

2 Factor Authentication (2FA)

- **Something you have AND something you know**
- Either factor is useless without the other
- Chip and PIN
- Commonly implemented in mobile phones via SMS
- Disadvantages:
  - ONE device (if hacked)
  - SMS is easy to redirect
  - ONE point of failure
- Google authenticator, Duo Mobile, Authy, Yubico Authenticator
- OTP tokens (e.g., TOTP), U2F keys

Other Schemes: Behavior Profiling

- Track access behavior of users
  - Systems used
  - Times and locations when active
  - Typical usage
- Look for anomalous or fraudulent behavior
- “Why is this guy who was in Thuwal 2 minutes ago logging in from Siberia?”
- Used in fraud prevention

Authentication vs Authorization

- Authentication – Who goes there?
  - Restrictions on who (or what) can access system
- **Authorization –** Are you allowed to do that?
  - Restrictions on actions of authenticated users
- Authorization is a form of **access control**
- Authorization enforced by
  - Access Control Lists
  - Capabilities

Access Control

- Access control is a collection of methods and components that supports
  - Confidentiality
  - Integrity
- Goal: allow only authorized **subjects** to access permitted **objects**
- E.g., Least privilege philosophy
  - A subject is granted permissions needed to accomplish required tasks and nothing more

Access Control Designs

- Access control designs define rules for users accessing files or devices
- Three common access control designs
  - Mandatory access control
  - Discretionary access control
  - Role-based access control

Mandatory Access Control (MAC)

- It is a restrictive scheme that does not allow users to define permissions on files, regardless of ownership.
- Instead, **security decisions are made by a central policy administrator.**
- A common implementation is rule-based access control
  - Subject demonstrates need-to-know in addition to proper security clearance
  - Need-to-know indicates that a subject requires access to object to complete a particular task
- Security-Enhanced Linux (SELinux) incorporates MAC

Discretionary Access Control

- Discretionary access control, or DAC, refers to a scheme where **users are given the ability to determine the permissions governing access to their own files.**
  - DAC typically features the concept of both users and groups
  - In addition, DAC schemes allow users to grant privileges on resources to other users on the same system.
- Most common design in commercial operating systems
  - Generally less secure than mandatory control
  - Generally easier to implement and more flexible

Role-Based Access Control

- The role-based access control (RBAC) model can be viewed as an evolution of the notion of group-based permissions in file systems.
- An RBAC system is defined with respect to an organization, such as company, a set of resources, such as documents, print services, and network services, and a set of users, such as employees, suppliers, and customers
- **Uses a subject’s role or task to grant or deny object access**

Access Control Entries and Lists

- An Access Control List (ACL) for a resource (e.g., a file or folder) is a list of zero or more Access Control Entries (ACEs)
- An ACE refers specifies that a certain set of accesses (e.g., read, execute and write) to the resources is allowed or denied for a user or group
- Examples of ACEs for folder “CS230 Grades”
  - Professor; Read; Allow
  - Students; Read; Allow
  - Professor; Write; Allow
  - Students; Write; Deny

Access Control Entries and Lists

Unix Permissions

- Standard for all *nix systems
- Every file is owned by a user and has an associated group
- Permissions often displayed in compact 10-character notation
- To see permissions, use **ls –l** in terminal

Access Control Entries and Lists

Unix Permissions - Permissions Examples (Regular Files)

-rw-r--r--	read/write for owner, read-only for everyone else
-rw-r-----	read/write for owner, read-only for group, forbidden to others
-rwx------	read/write/execute for owner, forbidden to everyone else
-r--r--r--	read-only to everyone, including owner
-rwxrwxrwx	read/write/execute to everyone

Access Control Entries and Lists

Unix Permissions - Permissions Examples (/directory)

drwxr-xr-x	all can enter and list the directory, only owner can add/delete files
drwxrwx---	full access to owner and group, forbidden to others
drwx--x---	full access to owner, group can access known filenames in directory, forbidden to others
-rwxrwxrwx	full access to everyone

Access Control Entries and Lists: Unix

Special Permission Bits

- Three other permission bits exist
  - Set-user-ID (“suid” or “setuid”) bit
  - Set-group-ID (“sgid” or “setgid”) bit
  - Sticky bit

Access Control Entries and Lists: Unix

Special Permission Bits

Set-user-ID (“suid” or “setuid”) bit

- On executable files, causes the program to run as file owner regardless of who runs it
- Ignored for everything else
- In 10-character display, replaces the 4th character (x or -) with s (or S if not also executable)

-rwsr-xr-x: setuid, executable by all

-rwxr-xr-x: executable by all, but not setuid

-rwSr--r--: setuid, but not executable - not useful

Access Control Entries and Lists: Unix

Special Permission Bits

Root

- "root" account is a super user account, like Administrator on Windows
- Multiple roots possible
- File permissions do not restrict root
- This is *dangerous*, but necessary, and OK with good practices

Access Control Entries and Lists: Unix

Changing Permissions

- Permissions are changed with chmod or through a GUI
- Only the file owner or root can change permissions
- If a user owns a file, the user can use chgrp to set its group to any group of which the user is a member
- root can change file ownership with chown (and can optionally change group in the same command)
- chown, chmod, and chgrp can take the -R option to recurse through subdirectories

Access Control Entries and Lists: Unix

Changing Permissions Examples

chown -R root dir1	Changes ownership of dir1 and everything within it to root
chmod g+w,o-rwx file1 file2	Adds group write permission to file1 and file2, denying all access to others
chmod -R g=rwX dir1	Adds group read/write permission to dir1 and everything within it, and group execute permission on files or directories where someone has execute permission
chgrp testgrp file1	Sets file1's group to testgrp, if the user is a member of that group
chmod u+s file1	Sets the setuid bit on file1. (Doesn't change execute bit.)