CNT 5410 – Computer & Network Security:

Passwords

Prof. Vincent Bindschaedler

Fall 2023

Administrivia

- **Homework 1** is out!
  - Topic: passwords
  - Due: **10/2/2023** on Canvas
  - Advice: start early!
  - ***This is an individual assignment!***
    - Reminder of the Honor Pledge: *On all work submitted for credit by Students at the University of Florida, the following pledge is either required or implied:* ***"On my honor, I have neither given nor received unauthorized aid in doing this assignment."***

Reminder: Something You Know...

- Passwords and passphrases
  - Main issue: People aren't good at remember passwords
  - Can be guessed
    - By a human (someone who knows you?)
    - By a computer: most passwords have low entropy
    - If you can remember it, it's low entropy
- Entropy of 8 alphanumeric characters
  - Each character is in [a-zA-Z0-9] => 62 possibilities
    - $62^8 \approx 2^{48}$ => Entropy is 48 bits (DES uses 56 bits keys)
  - **Bad news: most passwords aren't uniformly distributed**
    - Brute force attacks are feasible
    - But: dictionary attacks are even easier

2017
123456
password
12345678
qwerty
12345
123456789
letmein
1234567
football

Source: SplashData

Reminder: Salting

- Web server stores a database of password hashes
  - In the form of pairs $(u, H(p))$ for each user u
  - Problem: if the database is stolen all passwords can be brute forced together!
  - For common hash functions (e.g., MD5, SHA1) you can find online pre-computed tables of pairs $(u, H(p))$ for passwords p up to a certain length

- Salted password hash
  - Instead of storing $(u, H(p))$ for each user u, the server stores $(u, s, H(s \,||\, p))$ where s is a salt (random or unique per user)
  - The salt ensures that each password must be attacked separately
    - Also prevents pre-computation of hashes for common passwords

Case Study: LAN Manager Hash

- Primary hash algorithm used before Windows NT

- Steps
  - User's password is at most 14 characters
  - The password is first converted to uppercase
  - Split in two halves of 7 characters each
  - Each half is used as a DES key
  - The keys are used to encrypt the string 'KGS!@#$%' with DES
  - The two 64 bits ciphertexts are concatenated to form the hash value (16 bytes)

- How difficult is a brute-force attack?
  - Simply attack each half of 7 characters (uppercase) separately
  - Complexity (assuming alphanumeric): $36^7 \approx 2^{36}$

How do Attackers Get Password Hashes

- Eavesdropping

- Data breaches
  - Someone exploits a vulnerability, obtains a copy of the server database, and publishes it online
    - Server database typically contains email and password hash for each user
  - Some companies get breached multiple times

- Example: Yahoo!
  - Aug 2013: 1 billion accounts
  - Late 2014: 500 million accounts
  - Discovered only in 2016

Password Cracking

- Many of tools available
  - Aircrack-ng, John the Ripper, Hashcat

- Highly customizable
  - Password rules
  - Dictionaries

- Pre-computed tables
  - You can download tables of password to hash pairs to speed up the process
  - Terabytes of tables available for LanMan, MD5, SHA-1

- Q: Is password cracking ethical?

Password Reuse

- Studies
  - at least 75% of users use the same (or similar) passwords across different websites
- Problem:
  - Users cannot remember unique randomly generated passwords for each website
  - Password managers can help!

- *The Emperor's New Password Manager: Security Analysis of Web-based Password Managers.* Li et al. USENIX Security 2014.
  - Finding: An attacker can learn the user's passwords for an arbitrary website in 4 out of 5 passwords managers tested.
  - Password manager is a single point of failure

Password Hash Tables

- What if I pre-compute the hash for each password?
  - To crack a new password: table lookup
  - Can use the same table to crack multiple passwords!

Password	Hash
123456	0xE4BC1EDF
password	0xCAAB89D2
letmein	0x308C992A
...	
soccer	0xF104BF12

- How big is the table?
  - $O(|p + h|\ n)$ for n passwords, if length of password and hash is $|p + h|$
  - LanMan Hash: 2^{36} passwords $\Rightarrow \approx$ 1 TB
  - Alphanumeric passwords of length 8 with SHA1
    - [a-zA-Z0-9] $\Rightarrow$ 62 possibilities $\Rightarrow$ 2^{48} possible passwords
    - Each table entry is 8 bytes (password) + 20 bytes for SHA1 hash
    - 2^{48} entries × 28 bytes per entry $\Rightarrow 2^{53} \approx$ 8 PB
    - Too large to be practical to store

Password Hash Tables

- We can't store the full hash table
  - What if we don't store all the hashes?
  - Can we still make use of the table?

- Time-memory trade-off
  - General cryptographic technique
  - We can do more work (than a lookup) while storing less (the full table)
  - How?
    - Rainbow tables

Hash - Reduce Chains

- Idea
  - **Hash** function H: maps passwords to hashes
  - **Reduce** function R: maps hashes back to passwords
  - Build a chain from password p as follows:
    - $p \longrightarrow H(p) = h$
    - $h \longrightarrow R(h) = p'$
    - $p' \longrightarrow H(p') = h'$
    - ...
  - Example:
    - crypto $\longrightarrow^{H}$ 0xf533e2 $\longrightarrow^{R}$ arudo $\longrightarrow^{H}$ 0x02ab59 $\longrightarrow^{R}$ sfhyun $\longrightarrow^{H}$ ...

Hash - Reduce Chains

- How to use Hash - Reduce chains?
  - Compute chains of fixed length k
  - Store only the first password (starting point) and last password (endpoint) of the chain

- How to do a lookup?
  - Given a hash h
  - Steps
    1. Compute the hash - reduce chain for h until you find an endpoint
    2. Use the corresponding startpoint to recreate the chain
    3. If the chain contains h then we have the password!

  - Is this guaranteed to work? **No!**
    - Why might the chain not contain h?

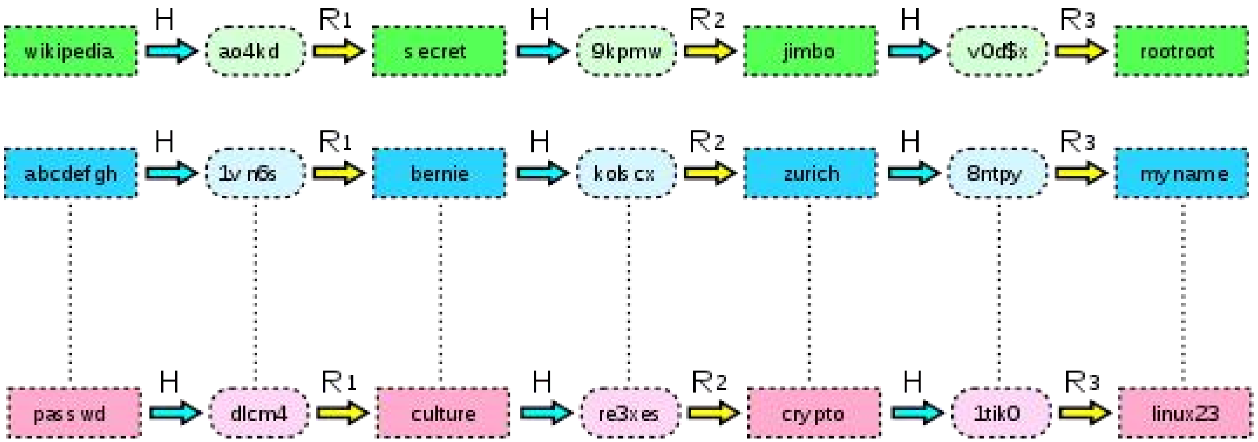
Hash - Reduce Chains

- Is this guaranteed to work? **No!**
  - Why might the chain not contain h?
    - A different hash h' could be reduced to the same value as $R(h)$
    - Some hash - reduce chains could merge
  - In such cases we have a *false alarm*. How do we fix it?
  - Problem: the reduce function **cannot be** collision resistant

- Time-memory trade-off
  - Chains of length 1: fast lookup, huge table
  - Very long chains: small table, slow lookup
  - Trade-off: chains of length k
    - Lookup: $O(k)$
    - Size of table: $O(|p|\ n\ k^{-1})$

Rainbow Tables

- Use multiple different reduce functions R_1, R_2, ..., R_k
- Solves the collison problem of hash - reduce chain
  - Collision only if chains hit the same value at the same iteration
- Example: Rainbow table with 3 reduce functions



Source: Wikipedia

Salt and Pepper

- Salting, peppering, and key stretching defeat rainbow tables

- Salting:
  - Pick a salt value s (public)
  - Compute password hash as $H(s||H(p))$
- Peppering:
  - Pick a short pepper value r (e.g., one character)
  - Compute password hash as $H(p||r)$
  - Server does not store r; it tries all possible pepper values
- Key stretching:
  - Use a fixed number of iterations for hashing $H^n(p)$
  - Password hash functions: e.g., PBKDF, bcrypt

Honeywords

- *Honeywords: Making Password-Cracking Detectable.* Ari Juels and Ron Rivest. ACM CCS 2013.

- Idea:
  - When you signup for an account, the webserver generates k-1 fake passwords (*honeywords*)
  - These honeywords (i.e., their password hashes) are stored in the database

- Honeyword: false password

- Login with a honeyword
  - Sets off an alarm
  - Login attempt with a honeyword suggests a data breach

Password Policies

- Examples
  - Passwords must contain at least one digit or special character
  - Periodic password reset (e.g., every 6 months)
  - Forbid common passwords (blacklist)
- Are such password policies actually effective?
- Weir et al. 2010 study
  - Users capitalize the first letter or add '1' or '!' to the end
- 70% of Rockyou users added a digit before or after their password
  - Calculation
    - 8 alphanumeric character uniformly at random: 48 bits entropy
    - 7 alpha characters ([a-zA-Z]) and one digit after: about 43 bits of entropy
- Periodic password resets
  - users simply add a character to their previous password or replace a character (e.g., ‘l’ $\rightarrow$ ‘1’)

References

- *The Password Thicket: Technical and Market Failures in Human Authentication on the Web.* Bonneau and Preibusch. WEIS 2010.
- *The Emperor's New Password Manager: Security Analysis of Web-based Password Managers.* Li et al. USENIX Security 2014.
- *Honeywords: Making Password-Cracking Detectable.* Ari Juels and Ron Rivest. ACM CCS 2013.

Next Time

- Lecture on 9/20/2023
  - Topic: Research Methods 2
- Reading assignment(s):
  - *Tips and advice when you review a scientific paper.* Bestoun S. Ahmed.
- Upcoming
  - Project Proposals are due 9/27 on Canvas (by 11:59pm)