Authentication and Passwords

Christopher Kruegel

Spring 2023

UC Santa Barbara

Passwords

User Authentication

- Authentication is the process of proving identity within an access control framework
  - Trusted system checks credentials presented by users
  - "Something you *have*, *know*, or *are*"
  - Unguessable, unforgeable, revocable
- Passwords are the *de facto* single-factor credential

Passwords fail in *numerous* ways

Password Attacks

- Passwords can be guessed
  - Passwords should have high entropy
  - People are bad at choosing high-entropy passwords
  - Machines can very quickly test password guesses
- Passwords must be protected at rest and in transit
  - Developers are bad at ensuring these properties

Online Password Attacks

Attacker simply guesses passwords until a correct guess is made

- Authentication systems should limit rate and total number of guesses
- Prevent, or make more difficult, automated interactions
- Apply same principles to *any* secrets (complete mediation) e.g., password recovery mechanisms

Password Strength

$$H = \log_2 N^L = L \log_2 N = L \frac{\log_i N}{\log_i 2}$$

- Entropy (H) is the usual password quality metric
  - N = number of possible symbols, L = lenght of password
  - Measure of *unpredictability* or *average information content*
- How to increase password strength in terms of N, L?
- What assumptions underlie entropy as a strength metric?

Password Selection

Table 1: Humans are notoriously bad at generating memorable random strings [1]

Rank	Password	Change
1	123456	–
2	password	–
3	12345678	+1
4	qwerty	+2
5	12345	-2
6	123456789	NEW
7	letmein	NEW
8	1234567	–
9	football	-2
10	iloveyou	NEW

';--have i been pwned?

Check if you have an account that has been compromised in a data breach

email address or username pwned?

264	4,859,717,682	61,409	59,821,668
pwned websites	pwned accounts	pastes	paste accounts

Top 10 breaches

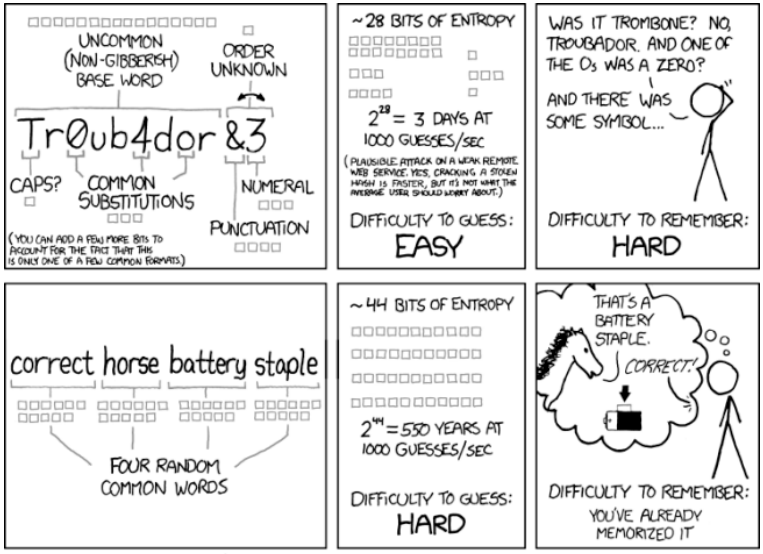
711,477,622 Onliner Spambot accounts

593,427,119 Exploit.In accounts

457,962,538 Anti Public Combo List

There's a lot of advice on how to select good passwords.

Most of it is bad.



THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Password Strength Meters

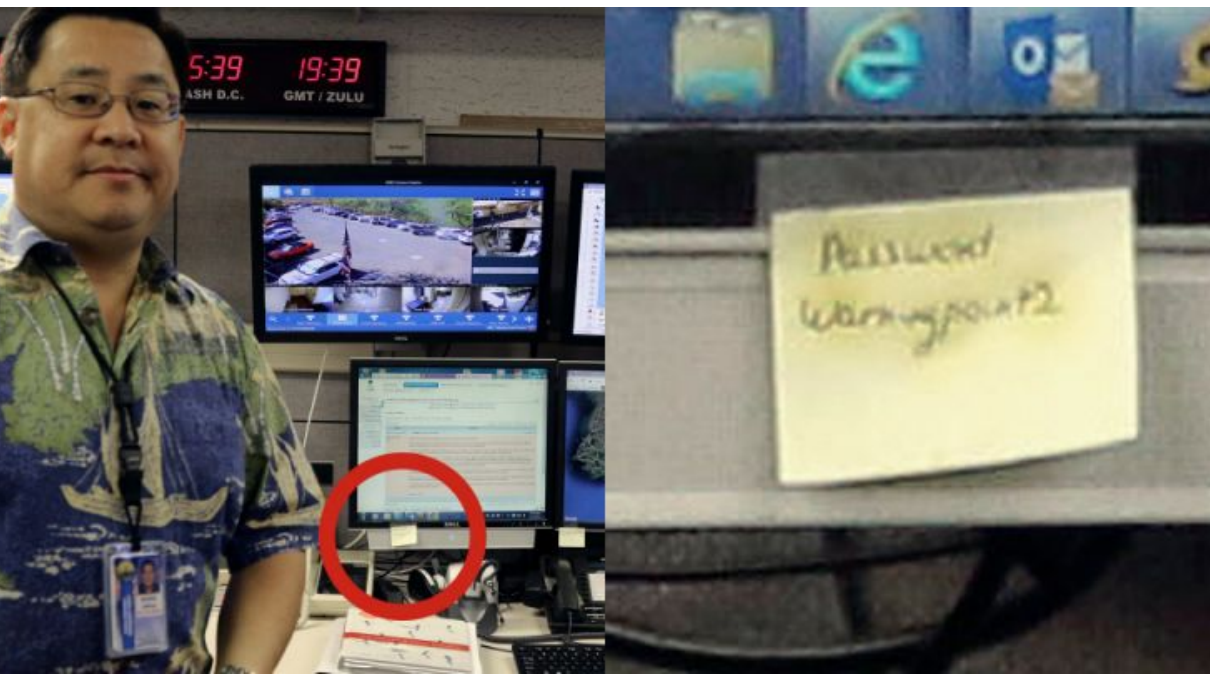
If people can't select good passwords, let's help them

- Meter gives immediate feedback on how strong a password is
- Ideally, should give *suggestions* on how to improve candidate passwords
- Requires a realistic model for what makes a password strong

	qwER43@!	Tr0ub4dour&3	correcthorsebatterystaple
zxcvbn	Weak	So-so	Great!
Dropbox (old)	Great!	Great!	So-so
Citibank	Medium	Strong	1 number required
Bank of America	(not allowed)	(not allowed)	(not allowed)
Twitter	✓ Password is perfect!	✓ Password is perfect!	✓ Password is perfect!

Password Selection Guidelines

- Avoid common passwords
- Avoid personal information
- Use a large symbol alphabet and long strings
- Don’t reuse passwords
- *Use a password manager*


5:39
19:39
GMT / ZULU
Password
Warningpoint2

Offline Password Attacks

Attacker captures password database and directly attacks it

- Obviously if passwords are in cleartext, the game is over
- Passwords are cryptographically hashed (**not encrypted**)
- Passwords checked by comparing hashes

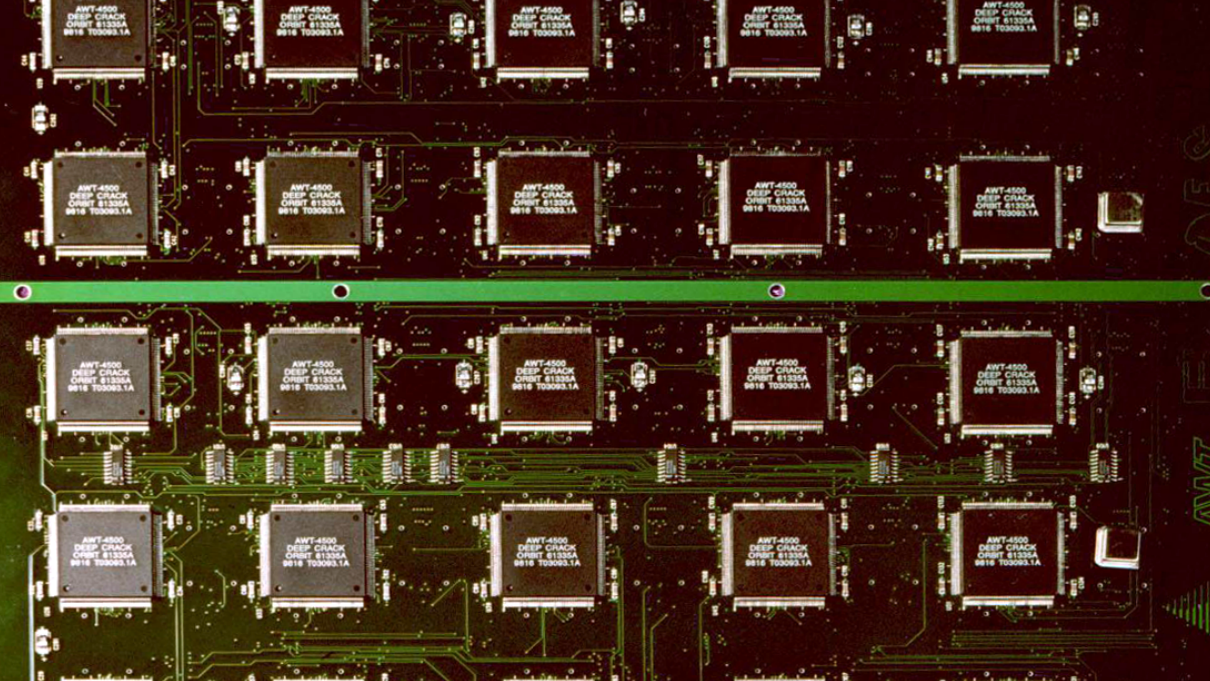
$$h_{\text{stored}} \stackrel{?}{=} H(p_{\text{provided}})$$

Historical Password Hashing

```
$ man 3 crypt
```

- 25 iterations of DES on a zeroed vector
- First eight bytes of the password used as the key
- 12-bit salt to hinder dictionary attacks

What is wrong with this method?


AWT-4500
DEEP CRACK
ORBIT 61335A
9816 T03093.1A

Key Derivation

- Key derivation function (KDF) produces a secret key from a secret input using a pseudorandom function (PRF)
- *Salt* is a nonce intended to prevent precomputation attacks
- *Key stretching* adds salt and iterations to slow each KDF application
- *Key strengthening* is similar but deletes the salt

Modern Password Hashing

```
$ man 3 crypt
```

- Modular crypt format: `$scheme$rounds$salt$hash`
- 10^3 – 10^8 iterations of SHA-2
- Full password is used
- Up to 16 bytes of salt
- See PBKDF2 [2]

Goals: Enlarge the search space, slow the guess rate

Modern Passsword Crackers are Fast

Hashcat Benchmark, 8x Nvidia GTX 1080, MD5

```
Speed.Dev.#1.: 24943.1 MH/s (97.53ms)
Speed.Dev.#2.: 24788.6 MH/s (96.69ms)
Speed.Dev.#3.: 25022.2 MH/s (97.76ms)
Speed.Dev.#4.: 25106.6 MH/s (97.42ms)
Speed.Dev.#5.: 25114.1 MH/s (97.42ms)
Speed.Dev.#6.: 24924.1 MH/s (97.30ms)
Speed.Dev.#7.: 25197.9 MH/s (97.30ms)
Speed.Dev.#8.: 25246.4 MH/s (97.00ms)
Speed.Dev.#*.:   200.3 GH/s
```

Memory-Hard Password Hashing

```
let block_size_factor = 8;
let block_size = 128 * block_size_factor;
let blocks = pbkdf2_hmac_sha256(passphrase, salt, 1, block_size * pf);
for i in 0..p { blocks[i] = ro_mix(blocks[i], 2^cost_factor); }
let expensive_salt = blocks.into_iter().join();
return pbkdf2_hmac_sha256(passphrase, expensive_salt, 1, key_length);
```

- scrypt [3] password-based key derivation function (PBKDF)
- Renders hardware-based attacks difficult by requiring large amounts of memory
- Also see Argon2 [4]

Password Search Strategies

- Precomputation
- Brute-force search
- Dictionary attacks
- Mutation rules
- Generative models
- Combinations of the above

Precomputation Attacks

Given a password space P, hash digest space D, and hash function $H : P \mapsto D$, precompute an inverse mapping $H' : D \mapsto P$

- Naïve precomputation requires $\Theta(|P|n)$ bits
- Hash chains can be used to balance the time-space tradeoff between run-time guessing and computing H'

Hash Chains

Precompute a list of password – hash digest mappings, but only store the start and end values

- Hash chains define a reduction $R : D \mapsto P$
- Reductions are not inverse mappings!
- Instead, R cover the space of likely passwords

Computing Hash Chains

$$p_{i,0} \xrightarrow{H} h_{i,0} \xrightarrow{R} p_{i,1} \rightsquigarrow h_{i,k-m} \xrightarrow{R} p_{i,k} \xrightarrow{H} h_{i,k}$$

- Chains are computed by selecting an initial password p_i and alternating applications of H, R up to length k
- Chain i becomes $(p_{i,0}, h_{i,k})$

Using Hash Chains

$$h_{i,j} \xrightarrow{R} p_{i,j} \rightsquigarrow h_{i,k-1} \xrightarrow{R} p_{i,k} \xrightarrow{H} h_{i,k}$$

$$\Downarrow$$

$$p_{i,0} \xrightarrow{H} h_{i,0} \xrightarrow{R} p_{i,1} \rightsquigarrow h_{i,j-1} \xrightarrow{R} p_{i,j} \xrightarrow{H} h_{i,j}$$

- To use given a hash h_j, apply R, H until a chain end value $h_{i,k}$ is found
- Then take $p_{i,0}$ and recompute the chain to find $H(p_{i,k}) = h_j$

Hash Chain Collisions

$$R(\text{"123456"}) = h_i = R(\text{"iloveyou"})$$

- Hash chains are prone to collisions $\Rightarrow$ false positives
- Very difficult to make R collision resistant since it must map into space of likely passwords
- Collisions cause chain merges that reduce coverage of P
- Merges $\Rightarrow$ chains might not contain a password even if an end value matches (Why?)

Rainbow Tables

- Rainbow tables reduce collision likelihood by using a reduction *family* $\mathbf{R} = \{R_1, R_2, \ldots, R_k\}$
- Instead of repeated applications of H, R, rainbow tables use $H, R_1, H, R_2, \ldots, H, R_k$ (Why?)

Rainbow Tables

- Rainbow tables reduce collision likelihood by using a reduction *family* $\mathbf{R} = \{R_1, R_2, \ldots, R_k\}$
- Instead of repeated applications of H, R, rainbow tables use $H, R_1, H, R_2, \ldots, H, R_k$ (Why?)

Collisions only occur between two chains if reduction functions are aligned!

Hash Chain Caveats

- Tables must be built for each hash function and symbol alphabet
- Salting and key stretching defeats efficiency gains
- Expensive to build

Brute-Force Search

```
// Try "aaaaaaaa", "aaaaaaab", "aaaaaaac", ...
let initial_guess = "aaaaaaaa";
for guess in password_space_iterator(initial_guess) {
    if hash(guess) == target_hash {
        println!("H({guess}) = {target_hash}");
        break;
    }
}
```

Dictionary Attacks

```
// Just try every entry in some provided dictionary
for guess in read_lines(dict_path) {
    if hash(guess) == target_hash {
        println!("H({guess}) = {target_hash}");
        break;
    }
}
```

Mutation Rules

```
// Example rule: Change all instances of 'e' to '3'
for guess in read_lines(dict_path) {
    for rule in rules {
        let mutated_guess = rule(guess);
        if hash(guess) == target_hash {
            println!("H({guess}) = {target_hash}");
            break;
        }
    }
}
```

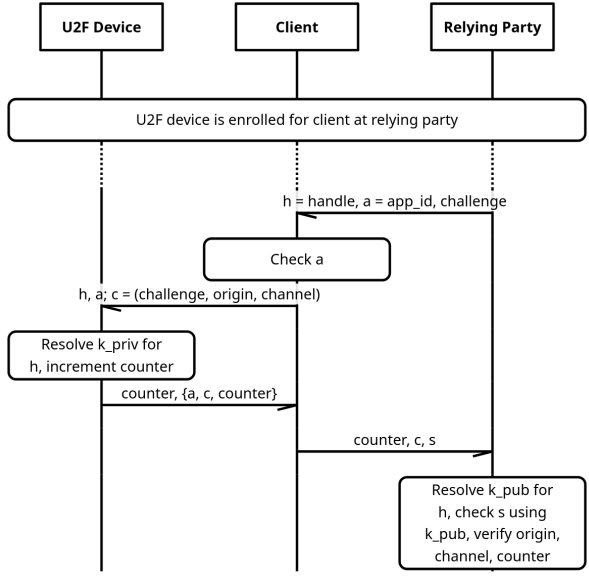
One-Time Passwords

```
let counter = floor((now() - epoch()) / interval);
let hotp = select_bytes(hmac_sha1(secret, counter));
let totp = hotp(secret, time_counter) % 10^d
```

- A one-time password (OTP) is only valid for one authentication attempt and cannot be replayed
- SMS codes
- Time-based One-Time Password algorithm (TOTP) [5]
  - Mostly used as a second factor

Universal Second Factor (U2F) [6]

- Adds a second factor *bound to a counterparty*
- Requires use of a hardware module with trusted element
- Requires user interaction, but prevents phishing/MitM attacks


U2F Device
Client
Relying Party
U2F device is enrolled for client at relying party
h = handle, a = app_id, challenge
Check a
h, a; c = (challenge, origin, channel)
Resolve k_priv for
h, increment counter
counter, {a, c, counter}
counter, c, s
Resolve k_pub for
h, check s using
k_pub, verify origin,
channel, counter

References

[1] “100 Worst Passwords of 2017.” [Online]. Available: https://www.teamsid.com/worst-passwords-2017-full-list/. [Accessed: 25-Jan-2018].

[2] “PKCS #5: Password-Based Cryptography Specification Version 2.0,” Sep-2000. [Online]. Available: https://tools.ietf.org/rfc/rfc2898.txt. [Accessed: 25-Jan-2018].

[3] “The scrypt Password-Based Key Derivation Function,” Aug-2016. [Online]. Available: https://tools.ietf.org/rfc/rfc7914.txt. [Accessed: 25-Jan-2018].

[4], and, “The password hash Argon2, winner of PHC.” [Online]. Available: https://github.com/P-H-C/phc-winner-argon2. [Accessed: 28-Jan-2020].

[5] “TOTP: Time-Based One-Time Password Algorithm,” May-2011. [Online]. Available: https://tools.ietf.org/rfc/rfc6238.txt. [Accessed: 25-Jan-2018].

References (cont.)

[6] “U2F v1.2 Specifications,” 11-Jul-2017. [Online]. Available: https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411.zip. [Accessed: 25-Jan-2018].