Computer Security

08. Authentication

Paul Krzyzanowski
Rutgers University
Spring 2018

Authentication

- Identification: who are you?
- Authentication: prove it
- Authorization: you can do it

- Protocols such as Kerberos combine all three

Authentication

Three factors:

- something you have *key, card*
  - Can be stolen
- something you know *passwords*
  - Can be guessed, shared, stolen
- something you are *biometrics*
  - Usually needs hardware, can be copied (sometimes)
  - Once copied, you're stuck

Multi-Factor Authentication

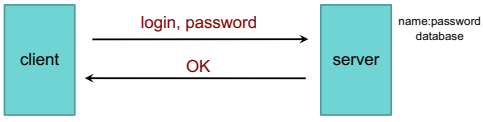
Factors may be combined

- ATM machine: 2-factor authentication
  - ATM card something you have
  - PIN something you know
- Password + code delivered via SMS: 2-factor authentication
  - Password something you know
  - Code validates that you possess your phone

Two passwords ≠ Two-factor authentication

Authentication: PAP

Password Authentication Protocol



- Unencrypted, reusable passwords
- Insecure on an open network
- Also, password file must be protected from open access
  - But administrators can still see everyone's passwords

Passwords are bad

- Human readable & easy to guess
  - People usually pick really bad passwords
- Easy to forget
- Usually short
- Static … reused over & over
- Replayable

Common Passwords

Adobe security breach (November 2013)
- 152 million Adobe customer records … with encrypted passwords
- Adobe encrypted passwords with a symmetric key algorithm
- … and used the same key to encrypt every password!

Top 26 Adobe Passwords

	Frequency	Password		Frequency	Password
1	1,911,938	123456	14	61,453	1234
2	446,162	123456789	15	56,744	adobe1
3	345,834	password	16	54,651	macromedia
4	211,659	adobe123	17	48,850	azerty
5	201,580	12345678	18	47,142	iloveyou
6	130,832	qwerty	19	44,281	aaaaaa
7	124,253	1234567	20	43,670	654321
8	113,884	111111	21	43,497	12345
9	83,411	photoshop	22	37,407	666666
10	82,694	123123	23	35,325	sunshine
11	76,910	1234567890	24	34,963	123321
12	76,186	000000	25	33,452	letmein
13	70,791	abc123	26	32,549	monkey

PAP: Reusable passwords

Problem #1: Open access to the password file

What if the password file isn't sufficiently protected and an intruder gets hold of it? All passwords are now compromised!

Even if a trusted admin sees your password, this might also be your password on other systems.

Solution:

Store a hash of the password in a file
- Given a file, you don't get the passwords
- Have to resort to a dictionary or brute-force attack
- Example, passwords hashed with SHA-512 hashes (SHA-2)

What is a dictionary attack?

- **Suppose you got access to a list of hashed passwords**
- **Brute-force, exhaustive search: try every combination**
  - Letters (A-Z, a-z), numbers (0-9), symbols (!@#$%...)
  - Assume 30 symbols + 52 letters + 10 digits = 92 characters
  - Test all passwords up to length 8
  - Combinations = $92^8 + 92^7 + 92^6 + 92^5 + 92^4 + 92^3 + 92^2 + 92^1 = 5.189 \times 10^{15}$
  - If we test 1 billion passwords per second: ≈ 60 days
- **But some passwords are more likely than others**
  - 1,991,938 Adobe customers used a password = "123456"
  - 345,834 users used a password = "password"
- **Dictionary attack**
  - Test lists of common passwords, dictionary words, names
  - Add common substitutions, prefixes, and suffixes

Easiest to do if you steal a hashed password file – so we read-protect the hashed passwords

Defenses

- Rate-limit guesses
  - Add timeouts after an incorrect password
    - Linux waits about 3 secs – and terminates the *login* program after 5 tries
- Lock out the account after *N* bad guesses
  - But this makes you vulnerable to denial-of-service attacks
- Use a slow algorithm to make guessing slow

What is salt?

- How to speed up a dictionary attack
  - Create a table of precomputed hashes
  - Now we just search a table

    Example: SHA-512 hash of "password" =
    sQnzu7wkTrgkQZF+0G1hi5AI3Qmzvv0bXgc5THBqi7mAsdd4Xll27ASbRt
    9fEyavWi6m0QP9B8lThf+rDKy8hg==

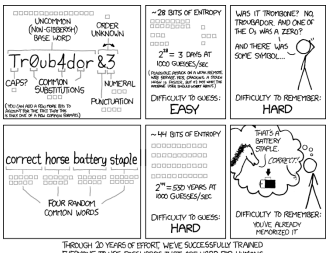
- Salt = random string (typically up to 16 characters)
  - Concatenated with the password
  - Stored with the password file (it's not secret)
  - Even if you know the salt, you cannot use precomputed hashes to search for a password (because the salt is prefixed)

    Example: SHA-512 hash of "am$7b22QLpassword", salt = "am$7b22QL":
    ntIxjDMnueMWig4dtWoMbaguucW6xV6cHJ+7yNrGvdoyFFRVb/LLqS01/pXS
    8xZ+ur7zPO2yn88xcliUPQj7xg==

  - You will ***not*** have a precomputed *hash("am$7b22QLpassword")*

Longer passwords

- English text has an entropy of about 1.2-1.5 bits per character
- Random text has an entropy ≈ log2(1/95) ≈ 6.6 bits/character



Assume 95 printable characters

People forget passwords

- Especially seldom-used ones
- How do we handle that?
- Email them?
  - Common solution
  - Requires that the server be able to get the password (can't store a hash)
  - What if someone reads your email?
- Reset them?
  - How do you authenticate the requester?
  - Usually send reset link to email address created at registration
  - But – what if someone reads your mail? …or you no longer have that address?
- Hints?
- Write them down?
  - OK if the threat model is electronic only

Reusable passwords in multiple places

- People often use the same password in different places
- If one site is compromised, the password can be used elsewhere
  - People often try to use the same email address and/or user name
- This is the root of phishing attacks
- Password managers
  - Software that stores passwords in an encrypted file
  - Do you trust the protection? The synchronization capabilities?
  - Can malware get to the database?
  - In general, these are good
    - Way better than storing passwords in a file
    - Encourages having unique passwords per site
    - Password managers may have the ability to recognize web sites & defend against phishing

PAP: Reusable passwords

Problem #2: Network sniffing

Passwords can be stolen by observing a user's session in person or over a network:
- snoop on telnet, ftp, rlogin, rsh sessions
- Trojan horse
- social engineering
- brute-force or dictionary attacks

Solutions:

(1) Use one-time passwords

(2) Use an encrypted communication channel

One-time passwords

Use a different password each time
- If an intruder captures the transaction, it won't work next time

Three forms

1. Sequence-based: password = *f*(previous password)
2. Time-based: password = *f*(time, secret)
3. Challenge-based: *f*(challenge, secret)

S/key authentication

- One-time password scheme
- Produces a limited number of authentication sessions
- Relies on one-way functions

S/key authentication

Authenticate Alice for 100 logins
- pick random number, R
- using a one-way function, f(x):

$x_1 = f(R)$
$x_2 = f(x_1) = f(f(R))$
$x_3 = f(x_2) = f(f(f(R)))$
… …
$x_{100} = f(x_{99}) = f(\ldots f(f(f(R)))\ldots)$

Give this list to Alice

- then compute:
  $x_{101} = f(x_{100}) = f(\ldots f(f(f(R)))\ldots)$

S/key authentication

Authenticate Alice for 100 logins

store x_{101} in a password file or database record associated with Alice

alice: x_{101}

S/key authentication

Alice presents the *last* number on her list:

Alice to host: { "alice", x_{100} }

Host computes f(x_{100}) and compares it with the value in the database

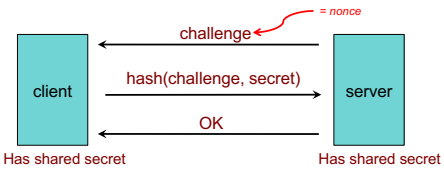
if (x_{100} provided by alice) = passwd("alice")
  replace x_{101} in db with x_{100} provided by alice
  return success
else
  fail

next time: Alice presents x_{99}

if someone sees x_{100} there is no way to generate x_{99}.

Authentication: CHAP

Challenge-Handshake Authentication Protocol

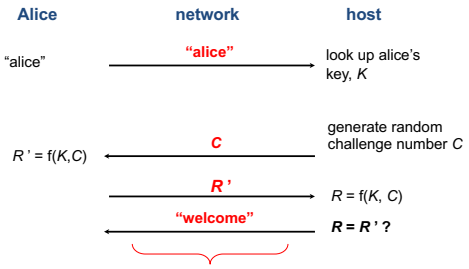


The challenge is a *nonce* (random bits).

We create a hash of the nonce and the secret.

An intruder does not have the secret and cannot do this!

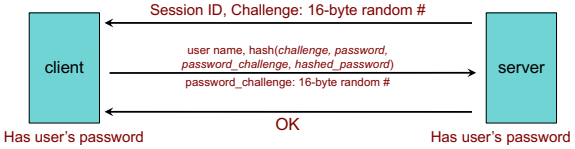
CHAP authentication



an eavesdropper does not see K

Authentication: MS-CHAP

Microsoft's Challenge-Handshake Authentication Protocol



The same as CHAP – we're just hashing more things in the response

SecurID card

Username: paul

Password: 1234032848

PIN + passcode from card

Something you know

Something you have

Passcode changes every 60 seconds

1. Enter PIN
2. Press ◊
3. Card computes password
4. Read password & enter

Password: 354982

SecurID card

- Proprietary device from RSA
  - SASL mechanism: RFC 2808
- Two-factor authentication based on:
  - **Shared secret key** (seed) ⇐ Something you have
    - stored on authentication card
  - **Shared personal ID** – PIN ⇐ Something you know
    - known by user

SecurID (SASL) authentication: server side

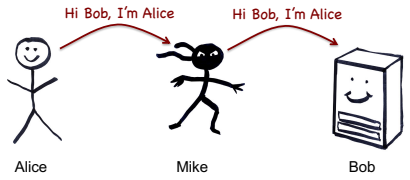
- Look up user's PIN and seed associated with the token
- Get the time of day
  - Server stores relative accuracy of clock in that SecurID card
  - historic pattern of drift
  - adds or subtracts offset to determine what the clock chip on the SecurID card believes is its current time
- Passcode is a cryptographic hash of seed, PIN, and time
  - server computes ***f* (seed, PIN, time)**
- Server compares results with data sent by client

Man-in-the-Middle Attacks

Password systems are vulnerable to man-in-the-middle attacks
- Attacker acts as the server

Man-in-the-Middle Attacks

Password systems are vulnerable to man-in-the-middle attacks
- Attacker acts as the server

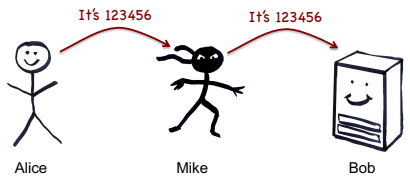


Man-in-the-Middle Attacks

Password systems are vulnerable to man-in-the-middle attacks
- Attacker acts as the server

Man-in-the-Middle Attacks

Password systems are vulnerable to man-in-the-middle attacks
- Attacker acts as the server

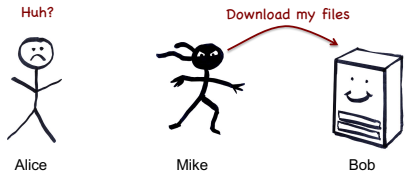


Man-in-the-Middle Attacks

Password systems are vulnerable to man-in-the-middle attacks
- Attacker acts as the server

Man-in-the-Middle Attacks

Password systems are vulnerable to man-in-the-middle attacks
- Attacker acts as the server



Guarding against man-in-the-middle attacks

- Use a covert communication channel
  - The intruder won't have the key
  - Can't see the contents of any messages
  - But you can't send the key over that channel!
- Use signed messages for all communication
  - Signed message = { message, encrypted hash of message }
  - Both parties can reject unauthenticated messages
  - The intruder cannot modify the messages
    - Signatures will fail (they will need to know how to encrypt the hash)
- But watch out for replay attacks!
  - May need to use session numbers or timestamps

Cryptographic toolbox

- Symmetric encryption
- Public key encryption
- Hash functions
- Random number generators

The End