Operating Systems

22. Authentication

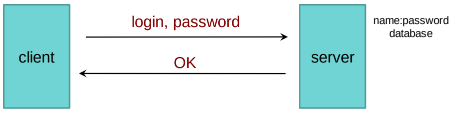
Paul Krzyzanowski

Rutgers University

Spring 2015

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

PAP: Reusable passwords

Problem: 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

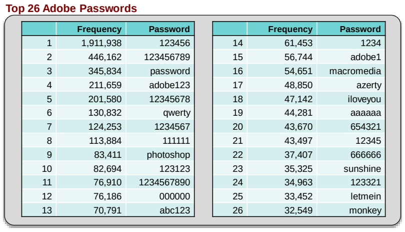
- Given a file, you don't get the passwords
- Have to resort to a dictionary or brute-force attack
- Example, passwords hashed with SHA-512 hashes (SHA-2)

What is a dictionary attack?

November 2013 – Adobe security breach

- 152 million Adobe customer records … with encrypted passwords
- Adobe encrypted passwords with a symmetric key algorithm
- … and used the same key for every password!

Top 26 Adobe Passwords



	Frequency	Password
1	1,911,938	123456
2	446,162	123456789
3	345,834	password
4	211,659	adobe123
5	201,580	12345678
6	130,832	qwerty
7	124,253	1234567
8	113,884	111111
9	83,411	photoshop
10	82,694	123123
11	76,910	1234567890
12	76,186	000000
13	70,791	abc123

	Frequency	Password
14	61,453	1234
15	56,744	adobe1
16	54,651	macromedia
17	48,850	azerty
18	47,142	iloveyou
19	44,281	aaaaaa
20	43,670	654321
21	43,497	12345
22	37,407	666666
23	35,325	sunshine
24	34,963	123321
25	33,452	letmein
26	32,549	monkey

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

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+ur7zPO2yn88xcIiUPQj7xg==

    - You will not have a precomputed hash of *"am$7b22QLpassword"*!

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

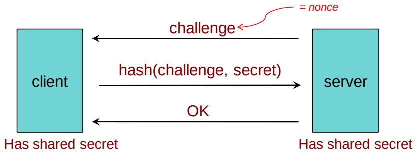
Solutions:

(1) Use one-time passwords

(2) Use an encrypted communication channel

Authentication: CHAP

Challenge-Handshake Authentication Protocol

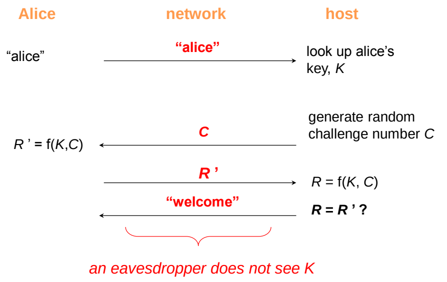


The challenge is a *nonce* (random bits).

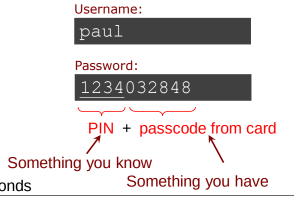
We create a hash of the nonce and the secret.

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

CHAP authentication



One-Time Passwords: SecurID card



Passcode changes every 60 seconds

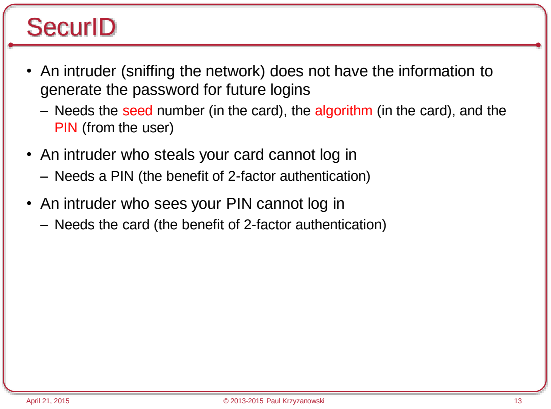
1. Enter PIN
2. Press ◊
3. Card computes password
4. Read password & enter

One-Time Passwords: SecurID card

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

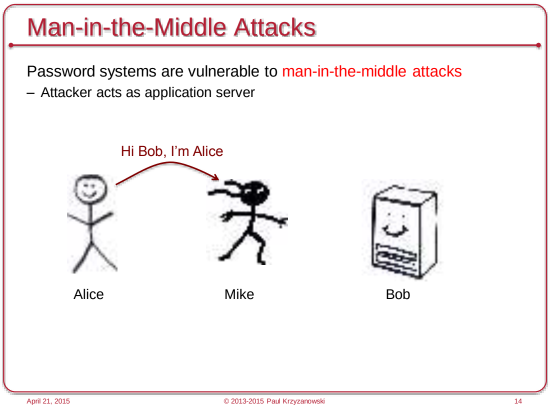
SecurID (SASL) authentication: server side

- 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



SecurID

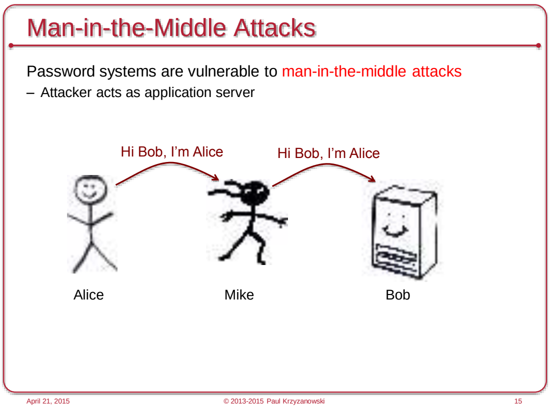
- An intruder (sniffing the network) does not have the information to generate the password for future logins
  – Needs the seed number (in the card), the algorithm (in the card), and the PIN (from the user)
- An intruder who steals your card cannot log in
  – Needs a PIN (the benefit of 2-factor authentication)
- An intruder who sees your PIN cannot log in
  – Needs the card (the benefit of 2-factor authentication)



Man-in-the-Middle Attacks

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

Hi Bob, I'm Alice

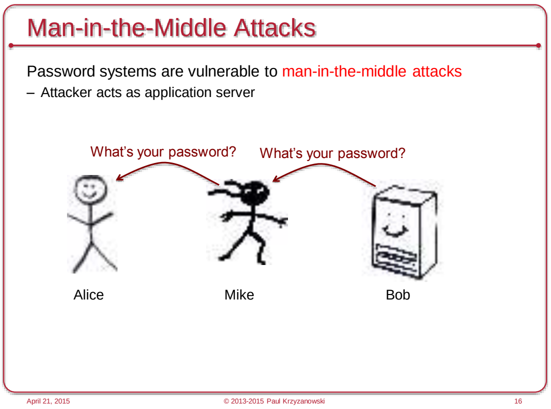


Man-in-the-Middle Attacks

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

Hi Bob, I'm Alice

Hi Bob, I'm Alice

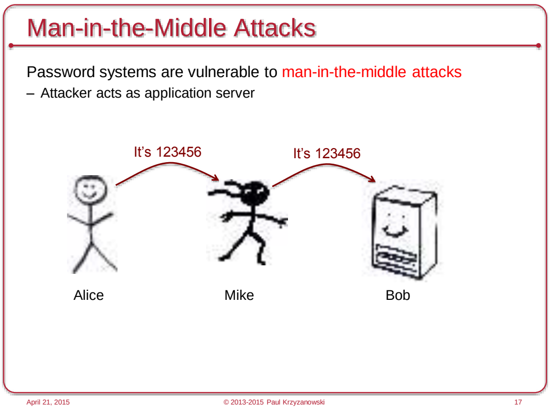


Man-in-the-Middle Attacks

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

What's your password?

What's your password?

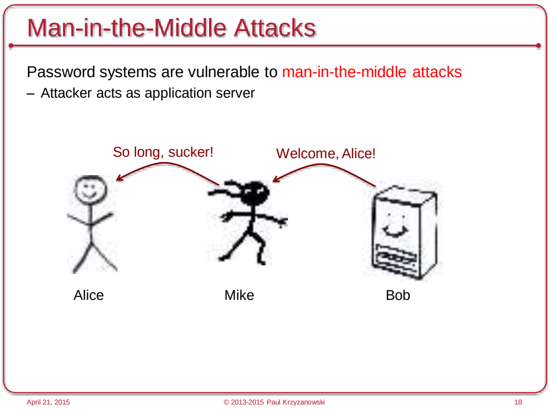


Man-in-the-Middle Attacks

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

It's 123456

It's 123456



Man-in-the-Middle Attacks

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

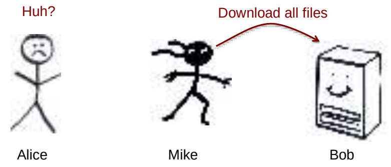
So long, sucker!

Welcome, Alice!

Man-in-the-Middle Attacks

Password systems are vulnerable to man-in-the-middle attacks

– Attacker acts as application server



Guarding against man-in-the-middle

- 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
  - Both parties can reject unauthenticated messages
  - The intruder cannot modify the messages
    - Signatures will fail (need to encrypt the hash)

Public key authentication

Demonstrate we can encrypt or decrypt a *nonce*

A random bunch of bits

- Alice wants to authenticate herself to Bob:
- Bob: generates nonce, *S*
  - Sends it to Alice
- Alice: encrypts *S* with her private key (signs it)
  - Sends result to Bob

Public key authentication

Bob:

1. Look up Alice's public key
2. Decrypt the message from Alice using Alice's public key
3. If the result is *S*, then Bob is convinced he's talking with Alice

For mutual authentication, Alice has to present Bob with a nonce that Bob will encrypt with his private key and return

Public key authentication

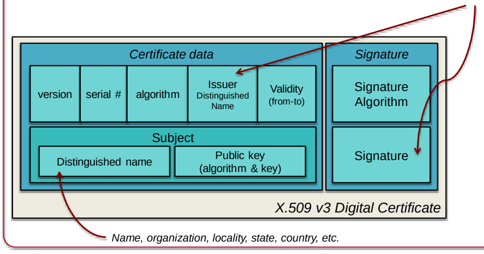
- Public key authentication relies on binding identity to a public key
  - *How do you know it really is Alice's public key?*
- One option:
  get keys from a trusted source
- Problem: requires always going to the source
  - cannot pass keys around
- Another option: *sign the public key*
  - Contents cannot be modified without detection
  - **digital certificate**

X.509 Certificates

ISO introduced a set of authentication protocols

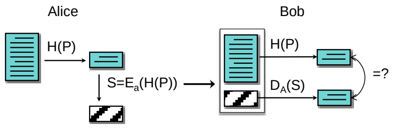
X.509: Structure for public key certificates**:**

Issuer = Certification Authority (CA)



Reminder: What's a digital signature?

Hash of a message encrypted with the signer's private key



X.509 certificates

When you get a certificate

- Verify its signature:
  - hash contents of certificate data
  - Decrypt CA's signature with CA's public key

Obtain CA's public key (certificate) from trusted source

- Certification authorities are organized in a hierarchy
- A CA certificate may be signed by a CA above it
  - Certificate chaining

Certificates prevent someone from using a phony public key to masquerade as another person

…if you trust the CA

Built-in trusted root certificates in iOS 8

- AAA Certificate Services
- AC Raíz Certicámara S.A.
- Actalis Authentication Root CA
- AddTrust Class 1 CA Root
- AddTrust External CA Root
- AddTrust Public CA Root
- AddTrust Qualified CA Root
- Admin-Root-CA
- AdminCA-CD-T01
- AffirmTrust Commercial
- AffirmTrust Networking
- AffirmTrust Premium
- AffirmTrust Premium ECC
- America Online Root Certification Authority 1
- America Online Root Certification Authority 2
- Apple Root CA
- Apple Root Certificate Authority
- ApplicationCA2 Root
- Autoridad de Certificacion Firmaprofesional
- Autoridad de Certificacion Raiz del Estado Venezolano
- Baltimore CyberTrust Root
- Belgium Root CA
- Buypass Class 2 Root CA
- Buypass Class 3 CA 1
- Buypass Class 3 Root CA
- CA Disig
- CA Disig Root R1
- CA Disig Root R2
- CNNIC ROOT
- COMODO Certification Authority
- CRL1
- CertiNomis
- Certigna
- Certinomis - Autorité Racine
- Certinomis - Root CA
- Certum CA
- Certum Trusted Network CA
- Certum Trusted Network CA 2
- Chambers of Commerce Root
- Chambers of Commerce Root - 2008
- China Internet Network Information Center EV Certificates Root
- Cisco Root CA 2048
- Class 2 Primary CA
- ComSign CA
- Common Policy
- D-TRUST Root Class 3 CA 2 2009
- DST ACES CA X6
- DST Root CA X3
- DST Root CA X4
- Deutsche Telekom Root CA 2
- Developer ID Certification Authority
- DigiCert Assured ID Root CA
- DigiCert Assured ID Root G2
- DigiCert Assured ID Root G3
- DigiCert Global Root CA
- DigiCert Global Root G2
- DigiCert Global Root G3
- DigiCert High Assurance EV Root CA
- DigiCert Trusted Root G4
- DigiNotar Cyber CA
- DigiNotar Extended Validation CA
- DigiNotar PKIoverheid CA Organisatie - G2
- DigiNotar PKIoverheid CA Overheid en Bedrijven
- DigiNotar Public CA 2025
- DigiNotar Qualified CA
- DigiNotar Root CA
- DigiNotar Root CA G2
- DigiNotar Services 1024 CA
- DigiNotar Services CA
- Digisign Server ID (Enrich)
- DoD CLASS 3 Root CA
- DoD Root CA 2
- E-Tugra Certification Authority
- EASEE-gas CA
- EBG Elektronik Sertifika Hizmet Sağlayıcısı
- ECA Root CA
- EE Certification Centre Root CA
- Echoworx Root CA2
- Entrust Certification Authority - L1C

Partial list from 475 CAs in
http://support.apple.com/kb/HT5012

The End