CSE331 Computer Security Fundamentals

10/3/2017 Authentication

Michalis Polychronakis
Stony Brook University
Authentication

The process of reliably verifying the identity or role of someone (or something)

What is identity?
Which characteristics uniquely identify an entity?

Authentication is a critical service, as many other security mechanisms are based on it
Entity authentication is the security service that enables communicating parties to verify the identity of their peers

Two main types
- Human to computer
- Computer to computer
Credentials

Evidence used to prove an identity

*User Authentication*: credentials supplied by the user

- Something you know
- Something you have
- Something you are

*Computer authentication*: crypto, location

- Computers (in contrast to humans) can “remember” large secrets (keys) and perform complex cryptographic operations
- Location: evidence that an entity is at a specific place (e.g., IP address/subnet)

Authentication can be delegated

- The verifying entity accepts that a trusted third party has already established authentication
Something You Know: Password-based Authentication

Passwords, passphrases, pins, key-phrases, access codes, …

Say the magic word

Good passwords are easy to remember and hard to guess

Easy to remember ➔ easy to guess
Hard to guess ➔ hard to remember
Bad ideas: DOB, SSN, zip code, favorite team name, …

Password space (bits) depends on:

Password length
Character set

Better way to think about strong passwords

**Long passphrases**, combined with custom variations, symbols, numbers, capitalization, …
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 Policies (often have the opposite effect)

Password rules

“At least one special character”
“Minimum/Maximum length of 8/12 characters”
“Must contain at least one number”
“Must contain at least one capital letter”

Hard to remember! ➔ encourage password reuse, writing down passwords insecurely, …

Periodic password changing

“You haven’t changed your password in the last 90 days”

Probably too late anyway if password has been stolen
Makes remembering passwords harder ➔ more passwords resets
What users do: password1 ➔ password2 ➔ password1 ➔ …
Attacking Passwords

Offline cracking
Online guessing
Eavesdropping
Capturing

\{ Brute force attacks \}
Password Storage

Storing passwords as plaintext is disastrous

Better way: store a cryptographic hash of the password

Even better: store a “salted” version of the password

Defend against dictionary attacks: prevent precomputation of hash values (wordlists of popular passwords, rainbow tables, …)

Even if two users have the same password, their hash values will be different ➔ need to be cracked separately

Salting does not make brute-force guessing a given password harder!

<table>
<thead>
<tr>
<th>Username</th>
<th>Salt</th>
<th>Password hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bobbie</td>
<td>4238</td>
<td>h(4238, $uperman)</td>
</tr>
<tr>
<td>Tony</td>
<td>2918</td>
<td>h(2918, 63%TaeFF)</td>
</tr>
<tr>
<td>Mitsos</td>
<td>6902</td>
<td>h(6902, zour1da)</td>
</tr>
<tr>
<td>Mark</td>
<td>1694</td>
<td>h(1694, Rockybrook#1)</td>
</tr>
</tbody>
</table>

Still, password databases are getting leaked…
Password Cracking

Exhaustive search → infeasible for large password spaces

Dictionary attacks
   Language words
   Lists of previously leaked real user passwords

Variations, common patterns, structure rules
   Prepend/append symbols/numbers/dates, weird capitalization,
   l33tspeak, visually similar characters, intended misspellings, …

Target-specific information
   DOB, family names, favorite team, pets, hobbies, anniversaries,
   language, slang, …
   Easy to acquire from social networking services and other public sites
   Particularly effective against “security questions”

Advanced techniques
   Probabilistic context-free grammars, Markov models, …

Combination of all the above
Example hashes

If you get a "line length exception" error in hashcat, it is often because the hash mode that you have requested does not match the hash. To verify, you can test your commands against example hashes.

Unless otherwise noted, the password for all example hashes is hashcat.

Generic hash types

<table>
<thead>
<tr>
<th>Hash-Mode</th>
<th>Hash-Name</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>MD5</td>
<td>8743b52063cd84097a65d1633f5c74f5</td>
</tr>
<tr>
<td>10</td>
<td>md5($pass.$salt)</td>
<td>01dfae6e5d4d90d9892622352959afbe:7050461</td>
</tr>
<tr>
<td>20</td>
<td>md5($salt.$pass)</td>
<td>f0fda58630310a6dd91a7d8f0e4c6da2:4226357426</td>
</tr>
<tr>
<td>30</td>
<td>md5(utf16le($pass).$salt)</td>
<td>b31d032cfdcf47a39999a71e43c5d2a:144816</td>
</tr>
<tr>
<td>40</td>
<td>md5($salt.utf16le($pass))</td>
<td>d63d0e21fd0c5f518d55ef306c54af82:13288442151473</td>
</tr>
<tr>
<td>50</td>
<td>HMAC-MD5 (key = $pass)</td>
<td>fc741db0a2968c39d9c2a5cc75b05370:1234</td>
</tr>
<tr>
<td>60</td>
<td>HMAC-MD5 (key = $salt)</td>
<td>bfd280436f45fa38eeacac3b00518f29:1234</td>
</tr>
<tr>
<td>100</td>
<td>SHA1</td>
<td>b89eaa7e61417341b710b727768294d0e6a277b</td>
</tr>
<tr>
<td>110</td>
<td>sha1($pass.$salt)</td>
<td>2fc5a684737ce1bf73b239df432416e0dd07357:2014</td>
</tr>
<tr>
<td>120</td>
<td>sha1($salt.$pass)</td>
<td>cac35ec206d8687d7cb0b55f31d9425b075082b:5363620024</td>
</tr>
<tr>
<td>130</td>
<td>sha1(utf16le($pass).$salt)</td>
<td>c57f6ac1b71f45a070db91a59fa47c23abc687c2:631225</td>
</tr>
<tr>
<td>140</td>
<td>sha1($salt.utf16le($pass))</td>
<td>5d61e4cd8776c7969cfd62456d639a4c876343484872</td>
</tr>
<tr>
<td>150</td>
<td>HMAC-SHA1 (key = $pass)</td>
<td>c69889ff370f6f1bc3fb19fe22aa860e57a71:1234</td>
</tr>
<tr>
<td>160</td>
<td>HMAC-SHA1 (key = $salt)</td>
<td>d89c92b4400b15c39e462a8ca939ab40c3aeea:1234</td>
</tr>
<tr>
<td>200</td>
<td>MYSQL323</td>
<td>7196759210defdc0</td>
</tr>
<tr>
<td>300</td>
<td>MySQL4.1/MySQL5</td>
<td>fc7c1b8749cf99d88e5f34271d636178fb5d130</td>
</tr>
</tbody>
</table>
## 25 Most-used (Worse) Passwords

<table>
<thead>
<tr>
<th>Password</th>
<th>Letmein</th>
<th>Year</th>
<th>Username</th>
</tr>
</thead>
<tbody>
<tr>
<td>password</td>
<td>letmein</td>
<td>2000</td>
<td>jordan</td>
</tr>
<tr>
<td>123456</td>
<td>monkey</td>
<td>jordan</td>
<td>superman</td>
</tr>
<tr>
<td>12345678</td>
<td>696969</td>
<td></td>
<td>harley</td>
</tr>
<tr>
<td>1234</td>
<td>abc123</td>
<td></td>
<td></td>
</tr>
<tr>
<td>qwerty</td>
<td>mustang</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12345</td>
<td>michael</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dragon</td>
<td>shadow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pussy</td>
<td>master</td>
<td></td>
<td></td>
</tr>
<tr>
<td>baseball</td>
<td>jennifer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>football</td>
<td>111111</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Distribution of 4-digit sequences within RockYou passwords
<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Last Update</th>
<th>Num of Hashes</th>
<th>Progress</th>
<th>Left Hashes</th>
<th>Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>6505</td>
<td>H4v3 1 b33n pwn3d (SHA1)</td>
<td>02.10.2017 - 02:03:24</td>
<td>320'294'464</td>
<td>319'837'535 (99.86%)</td>
<td>Get</td>
<td>Get</td>
</tr>
<tr>
<td>5638</td>
<td>P4y4sUGym (MD5)</td>
<td>02.10.2017 - 02:04:19</td>
<td>241'266</td>
<td>221'152 (91.66%)</td>
<td>Get</td>
<td>Get</td>
</tr>
<tr>
<td>4920</td>
<td>L1nk3d1n (SHA1)</td>
<td>02.10.2017 - 03:24:58</td>
<td>61'829'262</td>
<td>60'147'825 (97.28%)</td>
<td>Get</td>
<td>Get</td>
</tr>
<tr>
<td>3282</td>
<td>4mzr3v13w7r4d3r.c0m (MYSQL5)</td>
<td>02.10.2017 - 03:25:32</td>
<td>41'823</td>
<td>39'166 (93.65%)</td>
<td>Get</td>
<td>Get</td>
</tr>
<tr>
<td>3186</td>
<td>X5pl17 (SHA1)</td>
<td>02.10.2017 - 03:32:38</td>
<td>2'227'254</td>
<td>2'162'101 (97.07%)</td>
<td>Get</td>
<td>Get</td>
</tr>
<tr>
<td>2499</td>
<td>Hashkiller 32-hex left total</td>
<td>02.10.2017 - 11:48:14</td>
<td>9'976'651</td>
<td>1'723'709 (17.28%)</td>
<td>Get</td>
<td>Get</td>
</tr>
<tr>
<td>2498</td>
<td>Hashkiller 40-hex left total</td>
<td>02.10.2017 - 13:22:34</td>
<td>1'739'204</td>
<td>350'788 (20.17%)</td>
<td>Get</td>
<td>Get</td>
</tr>
<tr>
<td>1619</td>
<td>4m4t3urc0mmuni7y.c0m</td>
<td>02.10.2017 - 13:33:26</td>
<td>197'302</td>
<td>57'407 (29.1%)</td>
<td>Get</td>
<td>Get</td>
</tr>
<tr>
<td>1535</td>
<td>b73r.c0m (MD5)</td>
<td>02.10.2017 - 13:34:43</td>
<td>63'070</td>
<td>32'543 (51.6%)</td>
<td>Get</td>
<td>Get</td>
</tr>
<tr>
<td>1427</td>
<td>4v17r0n.fr</td>
<td>02.10.2017 - 13:34:43</td>
<td>2'405</td>
<td>2'334 (97.05%)</td>
<td>Get</td>
<td>Get</td>
</tr>
<tr>
<td>1366</td>
<td>v0d4f0n3 (MD5($pass.=&quot;s+(a*)&quot;)</td>
<td>02.10.2017 - 13:34:44</td>
<td>322</td>
<td>307 (95.34%)</td>
<td>Get</td>
<td>Get</td>
</tr>
<tr>
<td>1361</td>
<td>L1n4 r07-5 - 27 (MD5)</td>
<td>02.10.2017 - 13:34:44</td>
<td>176</td>
<td>128 (95.27%)</td>
<td>Get</td>
<td>Get</td>
</tr>
</tbody>
</table>
Password Hashing Functions

Problem: hash functions are very fast to evaluate \(\Rightarrow\) facilitate fast password cracking

Solution: slow down guessing process (password “stretching”)
- Benefit: cracking becomes very inefficient (e.g., 10-100ms per check)
- Drawback: increased cost for the server if it must handle many users

Make heavy use of available resources
- Computation should be fast enough to validate honest users, but render password guessing infeasible
- Adaptable: flexible cost (time/memory complexity) parameters

Bcrypt [Provos and Mazières, 1999]
- Cost-parameterized, modified version of the Blowfish encryption algorithm
- Tunable cost parameter (exponential number of loop iterations)

Alternatives: Scrypt (memory-hard), PBKDF2 (PKCS standard)
Online Guessing

Similar strategy to offline guessing, but rate-limited
   Connect, try a few passwords, get disconnected, repeat…

Prerequisite: know a valid user name

Many failed attempts can lead to a system reaction
   Introduce delay before accepting future attempts (exponential backoff)
   Shut off completely (e.g., ATM capturing/disabling a card after 3 tries)
   Ask user to solve a CAPTCHA

Very common against publicly accessible SSH, VPN, RDP, and other servers
   Main reason people move sshd to a non-default port
   Fail2Ban: block IP address after many failed attempts ➔ may allow an attacker to lock you out of the server (!)
   Better: disable password auth and use a key pair ➔ cumbersome if having to log in from many/others’ computers
iPhone is disabled

try again in 1 minute
(a) A successful login
(b) Login rejected after name is entered
(c) Login rejected after name and password are typed ➔ less information makes guessing harder
Try the Default First

![Router Passwords.com](https://www.routerpasswords.com)

Welcome to the internet's largest and most updated default router passwords database.

Select Router Manufacturer:

- CISCO

**Find Password**

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Model</th>
<th>Protocol</th>
<th>Username</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>CISCO</td>
<td>CACHE ENGINE</td>
<td>CONSOLE</td>
<td>admin</td>
<td>diamond</td>
</tr>
<tr>
<td>CISCO</td>
<td>CONFIGMAKER</td>
<td></td>
<td>cmaker</td>
<td>cmaker</td>
</tr>
<tr>
<td>CISCO</td>
<td>CNR Rev. ALL</td>
<td>CNR GUI</td>
<td>admin</td>
<td>changeme</td>
</tr>
<tr>
<td>CISCO</td>
<td>NETRANGER/SECURE IDS</td>
<td>MULTI</td>
<td>netranger</td>
<td>attack</td>
</tr>
<tr>
<td>CISCO</td>
<td>BBSM Rev. 5.0 AND 5.1</td>
<td>TELNET OR NAMED PIPES</td>
<td>bbsd-client</td>
<td>changeme2</td>
</tr>
<tr>
<td>CISCO</td>
<td>BBSD MSDE CLIENT Rev. 5.0 AND 5.1</td>
<td>TELNET OR NAMED PIPES</td>
<td>bbsd-client</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Eavesdropping and Replay

Physical world

- Watch user type password (shoulder surfing)
- Cameras (ATMs skimmers)
- Lift fingerprints (iPhone)
- Post-it notes

Network makes things easier

- Sniffing (LAN, WiFi, …)
- Man-in-the-Middle attacks

Defenses

- Encryption
- One-time password schemes
Kerberos

Long-lived vs. session keys

- Use long-lived key for authentication and negotiating session keys
- Use “fresh,” ephemeral session keys (prevent replay, cryptanalysis, old compromised keys) for encrypted communication, MACs, …

Kerberos: most widely used (non-web) single sign-on system

- Originally developed at MIT, now used in Unix, Windows, …

Authenticate users to services: using their password as the initial key, without having to retype it for every interaction

- A Key Distribution Center (KDC) acts as a trusted third party for key distribution
- Online authentication: Variant of Needham-Schroeder protocol
- Assumes a non-trusted network: prevents eavesdropping
- Assumes that the Kerberos server and user workstations are secure…

Use cases: workstation login, remote share access, printers, …
Password Capture

Hardware bugs/keyloggers

Software keyloggers/malware

Cameras

Phishing

Social engineering
Press Ctrl-Alt-Delete to begin.

Requiring this key combination at startup helps keep computer secure. For more information, click Help.
(a) Correct login screen
(b) Phony login screen
Something You Have: Authentication Tokens

One-time passcode tokens
  Time-based
  Counter-based

Other authentication tokens: store certificates, encryption keys, challenge-response, ...

Smartcards (contact or contactless)
  Identification, authentication, data storage, limited processing
  Magnetic stripe cards, EMV (chip-n-pin credit cards), SIM cards, RFID tags, ...

USB/NFC tokens, mobile phones, watches, ...
  Can be used as authentication devices
Multi-factor Authentication

Present several separate credentials of different types

Most common: *two-factor authentication* (2FA)
- Example: Password + hardware token, mobile phone, …
- Example: ATM card + PIN

Motivation: a lost/guessed password is not enough anymore for attackers ➔ not always true

*Man-in-the-Middle*: set up fake banking website, relay password to real website, let the user deal with the second factor…

*Man-in-the-Browser*: hijack/manipulate an established session after authentication has completed (banking Trojans)

*Dual infection*: compromise both PC and mobile device

Implementation-dependent usability issues
- Token may be lost, in-flight WiFi but cannot receive SMS, …
- Fallback: backup one-time-use passcodes (where to keep them?)
SMS Is Not a Secure 2nd Factor  
*(but still better than no 2nd factor)*

Social engineering  
Call victim’s mobile operator and hijack the phone number  
SIM swap, message/call forwarding, …

Message interception  
Rogue cell towers: IMSI catchers, StingRays,…  
Some phones even display text messages on the lock screen (!)

SS7 attacks  
The protocol used for inter-provider signaling is severely outdated and vulnerable  
Allows attackers to spoof change requests to users' phone numbers and intercept calls or text messages
‘Sim swap’ gives fraudsters access-all-areas via your mobile phone

There’s a new, little-known scam designed to empty your bank account, as one Vodafone customer found to her cost
Better Alternative: Authenticator App

Six/eight digit code provided after successful password validation

Time-based one-time password (TOTP)

Code computed from a shared secret key and the current time (using HMAC)
The key is negotiated during registration

Requires “rough” synchronization between client and server

Code constantly changes in 30-second intervals

Phishing is still possible!
The attacker needs to proxy the captured credentials in real time (rather than collecting them for later use)

Session hijacking and Man-in-the-Middle are still possible!
After the user has successfully logged in
Even Better Alternative: U2F Tokens

Universal Second Factor (U2F)

FIDO (Fast IDentity Online) alliance: Google, Yubico, …
Supported by many popular online services
Supported by Chrome and Opera
(and soon Mozilla and Microsoft)

A different key pair is generated for each origin during registration

Origin = <protocol, hostname, port>
Private key stored on device
Public key sent to server
U2F tokens

Benefits

- Easy: just tap the button (no typing)
- Works out of the box (no drivers to install)
- USB, NFC, Bluetooth communication
- No shared secret between client and server
- Origin checking → effective against phishing!

Drawbacks

- Can be lost → a fallback is needed (e.g., Authenticator App)
- Still a bit cumbersome: have to pull keychain out of pocket and plug token in (or have an always pugged-in token per device)
- Cost ($7 – $60)

*Man-in-the-Browser is still possible!*
Single Sign-on/Social Login

Pros

Convenience: fewer passwords to remember
Rich experience through social features
Easier development: outsource user registration/management

Cons

Same credentials for multiple sites: single point of failure
Access to user’s profile
User tracking
Biometrics

Fingerprint reader

Face recognition
  Depth sensing, “liveness” detection (pulse, thermal), etc. to foil simple picture attack

Retina/iris scanner

Voice recognition
  ...

Continuous authentication
  Keystroke timing, usage patterns, …
Crypto-based Authentication

Rely on a cryptographic key to prove a user’s identity

User performs a requested cryptographic operation on a value (challenge) that the verifier supplies
  Usually based on knowledge of a key (secret key or private key)
  Can use symmetric (e.g., Kerberos) or public key schemes

How can we trust a key? Why is it authentic?
  Need to establish a level of trust

Different approaches: **TOFU, PKI, web of trust**
  Emerging approach: blockchain/ledger-based PKI
**Trust on First Use** (aka Key Continuity)

Use case: SSH

Performs *mutual authentication*

Server *always* authenticates the client
password, key pair, …

Client almost always authenticates the server – *except the first time!*

First connection: server presents its public key
No other option for the user but to accept it: MitM opportunity
Subsequent connections: client remembers server’s key, and triggers an alert on key mismatch

Pragmatic solution, but shifts the burden to users

Users must determine the validity of the presented key
Assuming a key change is valid without verifying the new key offers no protection against MitM (unfortunately, that’s what most users do)
@ WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED! @

IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
Please contact your system administrator.
Add correct host key in /root/.ssh/known_hosts to get rid of this message.
Offending key in /root/.ssh/known_hosts:1
RSA host key for 192.168.2.5 has changed and you have requested strict checking.
Host key verification failed.
This is a normal message in a normal conversation.

Now Alice is going to reinstall.

As soon as Alice reinstalled, I saw the notice above. Impressive.

Now Alice has uninstalled, and this message is being transmitted before Alice reinstalls.

Alice’s security code changed. Tap for more info.

Alice’s security code changed. Tap for more info.

Scan the code on your contact’s phone, or ask them to scan your code, to verify that your messages and calls to them are end-to-end encrypted. You can also compare the number above to verify. This is optional. Learn more.
Certificates

How can we distribute “trusted” public keys?
   Public directory → risk of forgery and tampering
   More practical solution: “certified” public keys

A certificate is a digitally signed message containing an identity and a public key
   Makes an association between a user/entity and a private key
   Valid until a certain period

Why trust a certificate?
   Because it is signed by an “authority”
   Third party’s signature prevents tampering
Public Key Infrastructures (PKI)

Facilitate the authentication and distribution of public keys based on identities

Set of roles, policies, and procedures to create, manage, distribute, use, store, and revoke certificates

An issuer signs certificates for subjects

Trust anchor

Methods of certification

Certificate authorities (hierarchical structure – root of trust)

Web of trust (decentralized, peer-to-peer structure)
Certificate Authorities

Trusted third-parties responsible for certifying public keys
   Most CAs are tree-structured
   Single point of failure: CAs can be compromised!

Why should we trust an authority?
   How do you know the public key of the Certificate Authority (CA)?

CA’s public key (trust anchor) must somehow be provided out of band
   Trust has to start somewhere

Operating systems and browsers are pre-configured with ~200 trusted root certificates
   A public key for any website in the world will be accepted without warning if certified by any of these CAs (more in the TLS lecture)
A Dutch certificate authority that suffered a major hack attack this summer has been unable to recover from the blow and filed for bankruptcy this week.
Web of Trust

Entirely decentralized authentication
  No single point of failure
  No need to buy certs from CAs
  Used in PGP

Users sign other users’ keys
  Only if they deem them trustworthy
  Certificate signings can form an arbitrarily complex graph
  Users can verify path to as many trust anchors as they wish

Drawbacks
  Hard to use, requires in-person verification – key signing parties!
  Hard to know what trust level to assign transitively
WoT Alternative: Online Social “Tracking”
Keybase.io

In essence, a directory associating public keys with names

Identity established through *public signatures*

**Identity proofs:** “I am Joe on Keybase and MrJoe on Twitter”

**Follower statements:** “I am Joe on Keybase and I just looked at Chris’s identity”

**Key ownership:** “I am Joe on Keybase and here’s my public key”

**Revocations:** “I take back what I said earlier”

Keybase identity = sum of other public identities

Twitter, Facebook, Github, Reddit, domain ownership, …

Example:

An attacker has to compromise all connected identities

The more connected identities, the harder to impersonate a user
Best Practices

Pick long passwords (passphrases)

Never reuse the same password on different services

Never share passwords

Use SSH keys instead of passwords

Use two-factor authentication when available

Use Authenticator App or U2F instead of SMS
  Disassociate phone number from account after initial setup

Use a password manager
  Not only for passwords! Also for “security” questions…