Authentication

Michalis Polychronakis

Stony Brook University
Authentication

The process of verifying someone’s identity or role
   User, device, service, request, …

What is identity?
   Which characteristics uniquely identify an entity?

Authentication is a critical service
   Enables communicating parties to verify the identity of their peers
   Many other security mechanisms rely on it

Two main types
   Human to computer
   Computer to computer
Credentials

Evidence used to prove an identity

*User Authentication*: credentials supplied by a person

  *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 (IP, subnet, switch port, …)

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, …

Good passwords are easy to remember and hard to guess

- Easy to remember $\Rightarrow$ easy to guess
- Hard to guess $\Rightarrow$ hard to remember
- Bad ideas: date of birth, SSN, zip code, favorite team name, …

Password space (bits) depends on:

- Password length
- Character set

Better way to think about strong passwords

**Long passphrases**

- Can be 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 (often miss the point)

“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”

Make passwords hard to remember! → encourage password reuse
Better: encourage long passphrases, evaluate strength on-the-fly

Periodic password changing

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

Probably too late anyway if password has already been stolen
Makes remembering passwords harder → more password resets
Hinders the use of password managers (!)
What users do: password1 → password2 → password3 → …
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 the hash of 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 happen to 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>Mitsos6902</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>

Password databases are still getting leaked…
Password Cracking

Exhaustive search ➔ infeasible for large password spaces

Dictionary attacks (words, real user passwords from previous leaks, …)

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, …
# 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-Type</th>
<th>Hash-Mode</th>
<th>Hash-Name</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>MD5</td>
<td>MD5(salt)</td>
<td>8743b52063cd84079a6561e3f5c74f5</td>
</tr>
<tr>
<td>10</td>
<td>MD5$</td>
<td>MD5($salt)</td>
<td>01d9a5e65d4d9d826225959afbe:7050461</td>
</tr>
<tr>
<td>20</td>
<td>MD5$</td>
<td>MD5($pass)</td>
<td>f0fda5863031a6dd9a17d8f0a4c0ada2:4225637426</td>
</tr>
<tr>
<td>30</td>
<td>MD5(sha1)</td>
<td>MD5(sha1)</td>
<td>b13dd32ef0af47a3999990a71e43c5d2a1:144816</td>
</tr>
<tr>
<td>40</td>
<td>MD5(sha1)</td>
<td>MD5(sha1)</td>
<td>d63dd0e1f40f0f5618d5f506c54af82:13288442151473</td>
</tr>
<tr>
<td>50</td>
<td>HMAC-MD5</td>
<td>HMAC-MD5</td>
<td>fc471db0d02996e399dc23a5cc7b03370:1234</td>
</tr>
<tr>
<td>60</td>
<td>HMAC-MD5</td>
<td>HMAC-MD5</td>
<td>bdf280436f45a38eaaac3b6b00518f29:1234</td>
</tr>
<tr>
<td>110</td>
<td>SHA1</td>
<td>SHA1</td>
<td>b989eac0e16714731b074b727768294d0e6a277b</td>
</tr>
<tr>
<td>120</td>
<td>SHA1</td>
<td>SHA1</td>
<td>2fc5a64077e1bf9b3239df4234a0e00d07357:2014</td>
</tr>
<tr>
<td>130</td>
<td>SHA1</td>
<td>SHA1</td>
<td>cac35c206d86b77c6cdbe55f91d9425b075062b:5363620024</td>
</tr>
<tr>
<td>140</td>
<td>SHA1</td>
<td>SHA1</td>
<td>c57f6aceb71f45a07d9d91599a47c3a8bcd87c2:631225</td>
</tr>
<tr>
<td>150</td>
<td>SHA1</td>
<td>SHA1</td>
<td>5db01e4cd8777c799ec5f02456ad391ac67663a:8760434884872</td>
</tr>
<tr>
<td>160</td>
<td>SHA1</td>
<td>SHA1</td>
<td>e98689e9f77a1bebc3f39be22aaf6e86e5ea717:1234</td>
</tr>
<tr>
<td>200</td>
<td>MySQL223</td>
<td>MySQL223</td>
<td>7196759210defdc0</td>
</tr>
<tr>
<td>300</td>
<td>MySQL4.1/MySQL5</td>
<td>MySQL4.1/MySQL5</td>
<td>97f7c1c18749de9d8e5f34271d63e178f5d130</td>
</tr>
<tr>
<td>Password</td>
<td>Password</td>
<td>Password</td>
<td>Password</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------</td>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td>123456</td>
<td>1234567</td>
<td>123</td>
<td>ashley</td>
</tr>
<tr>
<td>123456789</td>
<td>qwerty</td>
<td>omgpop</td>
<td>987654321</td>
</tr>
<tr>
<td>picture1</td>
<td>abc123</td>
<td>123321</td>
<td>unknown</td>
</tr>
<tr>
<td>password</td>
<td>Million2</td>
<td>654321</td>
<td>zxcvbnm</td>
</tr>
<tr>
<td>12345678</td>
<td>0000000</td>
<td>qwertyuiop</td>
<td>112233</td>
</tr>
<tr>
<td>111111</td>
<td>1234</td>
<td>qwer123456</td>
<td>chatbooks</td>
</tr>
<tr>
<td>123123</td>
<td>iloveyou</td>
<td>123456a</td>
<td>20100728</td>
</tr>
<tr>
<td>12345</td>
<td>aaron431</td>
<td>a123456</td>
<td>123123123</td>
</tr>
<tr>
<td>1234567890</td>
<td>password1</td>
<td>666666</td>
<td>princess</td>
</tr>
<tr>
<td>senha</td>
<td>qqww1122</td>
<td>asdfghjkl</td>
<td>jacket025</td>
</tr>
</tbody>
</table>

https://en.wikipedia.org/wiki/List_of_the_most_common_passwords
Distribution of 4-digit sequences within RockYou passwords
Wordlists

c#ebc.dk
goddess5
20071002
271075711
zs3cu7za
scoopn
frygas1411
SL12345651
12345687ee123
xuexi2010
daigoro
12345614
DICK4080
567891234
tilg0
6208c861
:zark:
ravishnseha
150571611369
661189
passme
trolovasveta
abdulkhaledque
007816
xLD5X
Florida2011
037037
WestCountry
hitsugaiya
955998126
3n3rmmax
4637324
bugger825
marmaris
jinjin111
170383gp
3484427
f133321
zwqrfg
67078657
43210699
6856
704870704870
pv041886
20060814
512881535
milanimitani
472619
dbyxw888
85717221
cce841215
ariana19321
bbbnln
ang34hehiu
w1112358
Brenda85
786525b
shi461988
pingu
yeybozip
71477nak
stokurew
gea8mw4yz
kukumbike
260888
jordi10
lexusis
kjo11a039
c84bwlb
priyanka05
loveneverdies
u8Aqebj576
FGYfgy77
659397
327296
74748585
19720919
050769585
nicopa
2232566
bearss
notpublic
istesrea100
ashraf19760
48144
22471015
antyzhou115
0167095246
ec1kag
226226226226
6767537/33
mimilebrock
gueis8850
fujinshan
counter
N8mr6n
520057
adc123
bmaster
cqbjh642z
eulada79
EMANUELLI
yanjing
assyst
62157173
0704224950753
6983929
axaxa
hilal1
30091983
2510618981
sookoukan
tosecondlife
p4os8m6q
015614117
acz7190
lysylm2
2xgialdl
gaybar9
88203009
MKtyh87
quiggle
2063775206
fr3ih3it
masich
pengaiwei
coalesce
5402768
thesis
aabbcc894
marlonmaxime
614850
yd220105
584521584521
txudecp
84410545
pietro.chiara
jman1514
timemaps
timeother
521422
willrock
YHrftgDK
xys96exq
mercadotecnia
8s5s8ex7
0125040344
margikta
omaopap
dfTi6n
1314528021
pixma760
pearpear
gothpunksk8er
rftaeo48
8d7R0K
5172032
aics07
34mariah
donggi11we1
samarica
cap1014
0167387943
AE86Trueno
19700913
mcsuap
bu56mbu
danbee
passw<
money521
conan83
nxfqpl
ragte143
kojyihen
058336257
sarah4444
7363437
freindship
JymivW0848
sb inbau
30907891
0515043111
1973@ti
wlgxjf
20081014
leelou4
8UfjgBo
200358808
dellede
liang123.
captainettekt
kwiki-mart
mdvoydas
tigmys2001
denial
678ad5251
woaiwuai
1591591951212
hNbDGN
cardcap
1398503939
001104
desare11
412724198
nibirikab
asferg
hq555
xgames7
muckerlee
choqui67
12130911
lierwe120
skytvdn
milena1995
kambala11
<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 b3nn 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 (MDS)</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 (MDS(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>
</tbody>
</table>
518 pwned websites
10,624,652,379 pwned accounts
113,998 pastes
199,730,234 paste accounts

Largest breaches
772,904,991 Collection #1 accounts
763,117,241 Verifications.io accounts
711,477,622 Onliner Spambot accounts
622,161,052 Data Enrichment Exposure From PDL Customer accounts
593,427,119 Exploit.In accounts
457,962,538 Anti Public Combo List accounts
393,430,309 River City Media Spam List accounts

Recently added breaches
11,788 WeLeakInfo accounts
465,141 Liker accounts
637,279 Travel Oklahoma accounts
66,521 Gab accounts
1,834,006 Oxfam accounts
1,921,722 Ticketcounter accounts
20,339,937 SuperVPN & GeckoVPN accounts
645,786 Film.In accounts
10,585 NurseryCam accounts
358,822 People's Energy accounts
Password Hashing Functions

Hash functions are very fast to evaluate \( \Rightarrow \) facilitate fast password cracking

**Solution:** slow down the guessing process (password “stretching”)

- Benefit: cracking becomes very inefficient (e.g., 10-100ms per check)
- Drawback: increased cost for the server if it must authenticate many users

Make heavy use of available resources

- Fast enough computation 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

**Credential stuffing**: try username + password combinations from previous breaches

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 the 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 after many failed attempts ➔ attackers may now be able to lock you out
Better: disable password authentication and use a key pair ➔ cumbersome if having to log in from many/others’ computers
iPhone is disabled
try again in 1 minute
Emergency
(a) Successful login
(b) Login rejected after name is entered
(c) Login rejected after name and password are typed → less information makes guessing harder
Welcome to the internet's largest and most updated default router passwords database.

Select Router Manufacturer:

<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>netrangr</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 (e.g., ATM skimmers)
- Lift fingerprints (e.g., Apple Touch ID)
- Post-it notes, notebooks, …

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 or counter-based

**Various 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

Must provide several separate credentials of different types

Most common: *two-factor authentication (2FA)*

- Example: Password + hardware token/SMS message/authenticator app, …
- Example: ATM card + PIN

Motivation: a captured/cracked password is not enough to compromise a victim’s account → **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

**More importantly**: the most commonly used 2nd factor (SMS) is the least secure
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
A Hacker Got All My Texts for $16

A gaping flaw in SMS lets hackers take over phone numbers in minutes by simply paying a company to reroute text messages.

By Joseph Cox

March 15, 2021, 1:10pm

I hadn't been SIM swapped, where hackers trick or bribe telecom employees to port a target's phone number to their own SIM card. Instead, the hacker used a service by a company called Sakari, which helps businesses do SMS marketing and mass messaging, to reroute my messages to him. This overlooked attack vector shows not only how unregulated commercial SMS tools are but also how there are gaping holes in our telecommunications infrastructure, with a hacker sometimes just having to pinky swear they...
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” client–server synchronization

Code constantly changes in 30-second intervals

*Phishing is still possible!*

The attacker just needs to proxy the captured credentials in real time (rather than collecting them for later use)
Evilginx2  https://github.com/kgretzky/evilginx2

Man-in-the-middle attack framework for phishing login credentials along with session cookies

- Bypasses 2-factor authentication

No need for HTML templates: just a web proxy

- Victim’s traffic is forwarded to the real website
- TLS termination at the proxy (e.g., using a LetsEncrypt certificate)
I love digging through certificate transparency logs. Today, I saw a fake Google Drive landing page freshly registered with Let’s Encrypt. It had a hardcoded picture/email of presumably the target. These can be a wealth of info that I recommend folks checking out.
Evilginx2’s Tokenized phishing URLs

Scanners look into public certificate transparency logs for newly registered domains

“For some phishing pages, it took usually one hour for the hostname to become banned and blacklisted by popular anti-spam filters”

Solution: create unique phishing URLs

Response to scanner: benign page
https://totally.not.fake.linkedin.foo.com/auth/sigin

Response to victim: malicious page
https://totally.not.fake.linkedin.foo.com/auth/signin?tk=secret_token

Additional countermeasure: temporarily hide the phishing page

While submitting it to bit.ly, sending it through email, appearing on CT log, …
Modlishka  https://github.com/drk1wi/Modlishka

Phishing reverse proxy

- Support for the majority of 2FA authentication schemes
- No website templates
- User credential harvesting (with context based on URL parameter passed identifiers)
- Web panel with a summary of collected credentials and user session impersonation

```bash
>>> "Modlishka" Piotr Duszynski @drk1wi - Reverse Proxy started <<<

Modlishka

Listening on: [127.0.0.1:443]

[Sat Dec 22 14:02:41 2018] INFO Username collected ID:[42bc12cf-eea6-4cc1-acc9-86fe10b81f4c] username: phishingng
[Sat Dec 22 14:02:47 2018] INFO Credentials collected ID:[42bc12cf-eea6-4cc1-acc9-86fe10b81f4c] username: phishingng password: supersecretpass
[Sat Dec 22 14:03:23 2018] INFO [P] Tracking victim via initial parameter 9a8d22a9-19be-ec33-bc6f-ff160227f770
[Sat Dec 22 14:03:46 2018] INFO Credentials collected ID:[9a8d22a9-19be-ec33-bc6f-ff160227f770] username: testuser password: yetanothersecretpass
```
**CredSniper**  [https://github.com/ustayready/CredSniper](https://github.com/ustayready/CredSniper)

Exact login form clones for realistic phishing

Supports TLS via Let's Encrypt, and phishing 2FA tokens

![Real vs Fake Login Forms](image-url)
Even Better Alternative: U2F Tokens

Universal Second Factor (U2F)

FIDO (Fast IDentity Online) alliance: Google, Yubico, …
Supported by all popular browsers and many online services

A different key pair is generated for each origin during registration

Origin = <protocol, hostname, port>
Private key stored re-generated on device
Public key sent to server

Additions to the authentication flow:

Origin (URI): *prevents phishing*
TLS Channel ID (optional): *prevents MitM*
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 (second U2F token, Authenticator App, …)
- Cumbersome: have to pull keychain out and plug token in (or have an always pugged-in token, in which case though it can be stolen along with the device)
- Cost ($10–$70)
2FA Recap – *What threats does it prevent?*

**SMS:** useful against two main threats

- Credential stuffing (people tend to reuse passwords across different services)
- Leaked passwords (post-it, hardware keyloggers, cameras, shoulder surfing, …)

*Introduces new security/privacy issues: SIM swapping, SMS forwarding, SMS spam…*

**Authenticator Apps/Push Auth:** much better alternative than SMS

- Protects against the same threats without relying on phone numbers

**U2F:** *additional protection against phishing*

- Modern phishing toolkits bypass SMS/Authenticator/Push 2FA through MitM
- Humans fall for typosquatting, but U2F’s origin check doesn’t

*None of the above protect against session hijacking and Man-in-the-Browser*

- Game over anyway if the host is compromised after the user has successfully logged in
Password Managers

Have become indispensable

- Encourage the use of complex/non-memorable passwords
- Obviate the need for password reuse: unique passwords per site/service

Protection against phishing: auto-fill won’t work for incorrect domains

- As long as users don’t copy/paste passwords out of the password manager (!)

Various options: third-party applications, OS-level, in-Browser

Password synchronization across devices

- Can the service provider access all my passwords or not?
- Preferable option: passwords should be encrypted with master password never visible to the cloud service
WebAuthn

W3C Web Authentication standard (FIDO2): Successor of FIDO U2F

Use cases

- Low friction and phishing-resistant 2FA (in conjunction with a password)
- Passwordless, biometrics-based re-authorization
- 2FA *without* a password (passwordless logins)

Authenticators: devices that can generate private/public key pairs and gather consent (simple tap, fingerprint read, …)

- Built-in: fingerprint readers, cameras, …
- External: USB, BLE, NFC, …

Single Sign-on/Social Login

Pros

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

Cons

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

Fingerprint reader

Face recognition
   Depth sensing, infrared cameras, …
   “liveness” detection (pulse, thermal) to foil simple picture attack

Retina/iris scanner

Voice recognition
   …

Related concept: 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 (e.g., U2F) 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: PKI based on blockchain distributed ledger
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

Accepting a key change 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.
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 that contains 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”
- Requiring a signature by a third party prevents straightforward tampering
Public Key Infrastructures (PKI)

Facilitate the authentication and distribution of public keys with the respective identities of entities

- People, organizations, devices, applications, …
- Set of roles, policies, hardware, software, and procedures to create, manage, distribute, use, store, and revoke digital certificates and manage public key encryption

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 we 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 tens/hundreds of trusted root certificates

   A public key for any website in the world will be accepted without warning if it has been certified by any of these CAs (more on that in the TLS lecture)
DIGINOTAR FILES FOR BANKRUPTCY IN WAKE OF DEVASTATING HACK

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  (mainly used in PGP – more in the email lecture)

Entirely decentralized authentication
- No single point of failure
- No need to buy certs from CAs

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 public identities**

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

An attacker has to compromise all connected identities

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

Use long passphrases instead of passwords
   Never reuse the same password on different services

Use two-factor authentication when available
   Avoid SMS if possible! Use an authenticator app or U2F instead
   Remove phone number from account after authenticator/U2F setup
   Store your backup codes in a safe location

Use a password manager
   Pick non-memorable passwords and avoid copy/pasting them
   Password auto-fill helps against phishing (auto-fill won’t work if the domain is wrong)
   Not only for passwords! Also for “security” questions

Use SSH keys instead of passwords