EITA25 Computer Security (Datasäkerhet)
User Authentication

PAUL STANKOVSKI WAGNER, EIT, 2020-01-27

User Authentication

- **Identification**
  - Present an identifier to a security system
  - Example: username
- **Verification**
  - Verify the claimed identity
  - Example: password
- An authenticated identity provides the basis for both *access control* and *accountability*
- Do not confuse **user authentication** and **message authentication**
  - **User authentication:** Establishing the validity of a claim
  - **Message authentication:** Verify integrity and source authenticity of a message

User Authentication
- Identification
- Verification

Means of Authentication

- **Something you know**, e.g., password, passphrase, PIN
- **Something you have**, e.g., smart card, physical key, smartphone

Drawbacks
Can be forgotten, lost or stolen

- **Something you are (static biometrics)**, e.g., fingerprint, retina, iris, hand geometry, facial characteristics
- **Something you do (dynamic biometrics)**, e.g., voice recognition, handwriting, typing rythm

Drawbacks
Errors, problems with acceptance, cost

Multifactor authentication: Use of more than one of the above

Common Passwords

- Stolen from RockYou.com 2009 (SQL-injection)
- Stored in clear text (32 Million passwords)
- **Note:** People may or may not regard RockYou.com as a place where you need complex passwords.

Password length	
5	4%
6	26%
7	19%
8	20%
9	12%
10	9%
11	4%
12	2%
13	1%

Characters used	
Numbers	16%
Letters	43%
Alphanumeric	37%
Other	4%

10 most common
1. 123456
2. 12345
3. 123456789
4. Password
5. iloveyou
6. princess
7. Rockyou
8. 1234567
9. 12345678
10. abc123

10 years later…

- Data taken from public sources with data breaches from 2019.
- 5 million passwords in total.

1	123456
2	123456789
3	qwerty
4	12345678
5	111111
6	1234567890
7	1234567
8	password
9	123123
10	987654321
11	qwertyuiop
12	mynoob
13	123321
14	666666
15	18atcskd2w

- 3% used 123456
- Top 25 constitute over 10% of the total 5 million passwords

The Password File

- System needs to verify password
- Password needs to be stored somewhere
  - file, database,…
- Users should not be allowed to see other user's password
  - → Password file must be protected

Protection:

- One-way (hash) function is used so passwords are not in clear
- Additional cryptography and/or access control is possible

password → Hash function → Hash value

We will improve this further soon!!!

Password File Protection

- We want to protect the hashed passwords
- Access control – Only priviliged users can access the file
  - In Unix (and Linux) hashed passwords are usually stored in a file only readable by root (shadow password file)
  - Windows NT used a proprietary binary format for the file. (Security by obscurity)
  - In Windows 2000 and later, the SAM file is (optionally) encrypted with SysKey
  - The SAM file cannot be moved or copied when windows is running.
    - » Still, there are tools to dump the content, see Laboratory 1

Obtaining Passwords

- Spoofing Attacks
- Obtaining file with hashed passwords
  - Brute force
  - Dictionary attack
  - Time-memory tradeoff
- Social engineering
- Guess password online
- Guess answer to secret question

Spoofing Attacks

- Username and password give *unilateral* authentication
  - System authenticates user but user does not authenticate system

Spoofing Attack:

- The attacker runs a program that presents a fake login screen.
- User enters username and password, and is then directed to the real login screen.

What to do?

- Prevention
  - » Trusted path (CTRL+ALT+DEL in Windows)
  - » *Mutual authentication*
  - » One-time passwords
- Detection
  - » Information about previous logon session
  - » Display number of failed logins

Obtaining Hashed Passwords

- There are tools to dump the password database (SAM) in Windows
- Security vulnerabilities in other programs may allow you to read a password file in Unix or Linux
- Online forums, social networks, webmail providers, etc., often have databases with hashed passwords. These can be obtained through security bugs
  - Some methods will be discussed in the course ”Web Security” in HT1

Username	Password
Alice	g6F4fdsg8h...h5NHa
Bob	dsjk7H5dg0...d2a5V
Charlie	KJ7YtrcZa2...l9j7G
David	p09J7h6bD3...73Dnt

Brute Force

- Go through all possible passwords
  - Will take a long long time.
  - Can restrict to only test common characters (alphanumerical).
- 26+26 letters + 10 numbers
  - **Example:** Testing all alphanumerical passwords up to length 7 requires

$$\sum_{i=1}^{7} 62^i = 3579345993194 \approx 2^{42}$$

hash invocations

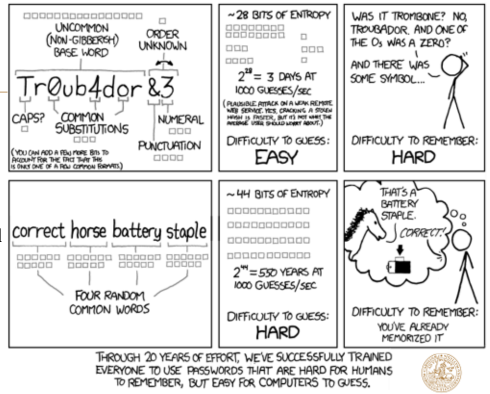
- Is this computationally possible?
  - Depends on which hash function is used, how many computers you have and how much time you have, but basically, yes.

Dictionary Attack

- Passwords are often based on words – try common words
- Consider *Oxford English Dictionary*
  - Contains about 200,000 words
- **Complexity**
  - Trying 100 variations of each word require about 2^{24} hash invocations.
  - Doing the same thing for 50 languages require 2^{30} hash invocations
  - → Still about 4,000 times faster than trying all alphanumerical passwords up to 7 characters
- ”Easy” passwords can also be included in dictionaries
  - qwerty, q1w2e3r4t5, zaxscdvfbg, qwaszx, etc...
  - …and the 32,000,000 from RockYou.com and similar

Password Entropy

- The amount of information your password contains
- Related to compression
- Between 0.6 and 1.3 bits per character, or about 3 to 7 bits per (English) word according to Shannon's estimate



Time-Memory Tradeoff Attack

- In some sense a brute-force attack
  - Done in a clever way and partly in advance
- Require lots of memory
- Attack introduced by Hellman in 1980
  - Explained for block cipher but works equally well for any one-way function
- Attack consists of two stages

Preprocessing
- Build tables
- Takes a long time

Capture hashes →

Realtime (or online) phase
- Use tables
- Relatively fast

The two phases have **different** complexity

Preprocessing Phase

- Let N be the search space
  - **Example:** alphanumerical passwords with length ≤ 7 gives $N = 2^{42}$
- Let h be the one-way function to invert
  - $y = h(x)$
- Let R be a reduction function mapping an output to a new password
  - $x_2 = R(h(x_1))$
- Idea:
  1. Pick random password $x_{1,0}$
  2. Compute $x_{1,1}=R(h(x_{1,0}))$, $x_{1,2}=R(h(x_{1,1}))$,…,$x_{1,t}=R(h(x_{1,t-1}))$
  3. Save $x_{1,0}$ as starting point and $x_{1,t}$ as ending point for this chain
  4. Pick new starting point $x_{2,0}$ and compute ending point $x_{2,t}$
  5. Do this for m starting points → we cover mt passwords

The Table

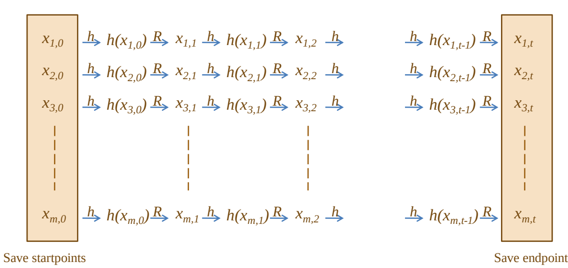


Table Coverage

- We cover mt points
- If $x_{i,j} = x_{u,v}$ the two chains will merge and we will not cover any new points
- Avoid merging: stop when $mt \cdot t = N$
  - Intuitive explanation: We have mt different points. If we add t points there are $mt \cdot t$ possibilities of collision
- We only cover a fraction $mt/N = 1/t$ of the search space
  - We need t tables, each with different reduction function R
- Cost for preprocessing phase $P \approx N$ ← **NOTE!!!**
  - All points are processed
- Memory usage $M \approx mt$
  - m points saved for each table, and there are t tables

Actually 2m, but we do not care about small constants (and we did not specify unit anyway)

In total t tables, each with new reduction function

Realtime Phase

- Goal: Find x when we know $h(x)$
- For each of the t tables do
  1. Apply reduction function R
  2. If $R(h(x))$ is a saved endpoint, then go to 4.
  3. If $R(h(x))$ is not a saved endpoint, find $R(h(R(h(x))))$, etc... until endpoint is found. Then go to 4.
  4. When endpoint is found, take corresponding startpoint and iterate until $h(x)$ is found. Then x is the password!
- Cost for realtime phase $T \approx t^2$

Summary of Attack

- Realtime computations: $T = t^2$
- Preprocessing time: $P = N$
- Memory needed: $M = mt$
- Matrix stopping rule: $N = mt^2$

$$\Downarrow$$

$$N^2 = M^2 T$$
$$P = N$$

- Example: $T = N^{2/3}$ and $M = N^{2/3}$
  - $N=2^{42}$ can be broken with table of size 2^{28} and 2^{28} computing steps
  - Thus after producing the table ONCE with cost 2^{42}, any password can be broken with cost 2^{28}. You just need to have the table.
  - Any parameters satisfying the tradeoff curve can be chosen
    - » More memory → less time
    - » Less memory → more time

Improvement: Rainbow Tables

- Oechslin 2003
- Practical improvement, but asymptotic complexities are the same as in Hellman's attack
- Idea: Use different reduction function for each computation of the hash function
- Collisions will merge chains with probability *1/t*
  - Only collisions in the same column will merge chains
- Only one large table needed
  - In practice, a few tables
- Realtime speedup factor approximately 2-10 (debated)
- See Laboratory 1 for more info

Downloadable Rainbow Tables

- Examples from CryptoHaze

Algorithm: MD5	*Algorithm:* MD5
Number of characters: 1-6	*Number of characters:* 1-8
Characters: !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~	*Characters:* !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
Size of table: 1.0 GB	*Size of table:* 1.5 TB

Does the choice of hash function matter to table size?

In the news, December 2009

Password Salting

- Add some extra info, **salt**, to the password before hashing.
  - Username
  - Randomly generated characters
- Salt stored with hashed password.

Username	Salt	Password
Alice	Gfgh5	g6F4fdsg8h...h5NHa
Bob	kd6sd	dsjk7H5dg0...d2a5V
Charlie	dsfjh	KJ7YtrcZa2...l9j7G
David	J7Fj2	p09J7h6bD3...73Dnt

- **Three advantages:**
  1. Two users with same password will have different hashes.
  2. Slows down dictionary attacks when trying to break several passwords at once.
  3. One Rainbow table needed for each salt.
- Not possible to know if same user has same password on two different systems.

Guess Passwords Online

- Possible targets: webmail, forums, communities, web shops,…
- Enter username + password and see if it works
  - Takes a long time
- Better
  1. Write program that sends the correctly formed HTTP requests (username + password) and analyzes the response
  2. Wait...
- **Protection:**
  - Do not allow many automated login attemps in a short time
  - Force user to verify that he/she is human after some failed attempts

Example, Online Password Guessing

- ”Twitter” was compromised in the beginning of January 2009
  - Dictionary attack used to try passwords online for a specific account
  - Password was ”happiness”
  - Account turned out to belong to a staff member
- **Consequence:** Attacker had control over all accounts on ”Twitter”
  - Fake comments from e.g., Barack Obama, Britney Spears and Fox News were sent out.
- Article on Wired

”Improved” Twitter

- Some time after the Jan 2009 attack Twitter decided to make improvements
  - The ”brute-force” dictionary attack no longer worked
- In Sept 2012 another Twitter account was online brute-forced?!?
- Turned out the login attempt **limitations was per IP**, not per account
- Article on Buzzfeed
- Perhaps now ”improved Twitter” can be called improved Twitter?

On the positive side, but completely unrelated, Twitter uses bcrypt to hash passwords

Guess Answer to Security Questions

- Security question common to allow users that forgot their password to recover it
  - In some cases the right answer will give you immediate access
- But obviously:
  - Password difficulty is upper bounded by the problem of answering the question
- Makes no sense to pick ”Hd#6%5Sue!7s” as password and ”What is my mother’s name?” as question.

Surely no one would do that.....or?

Example, Guessing Password Question

- Sarah Palin’s Yahoo account was compromised in Sept 2008
- **Required info:** Her birthday, her zip code and answer to security question
- **Security question:** ”Where did you meet your spouse?”
  - **Answer:** Wasilla High (the high school where she studied)
- According to attacker:
  - It took about 45 minutes in total
  - He found nothing of interest

System Help Mechanisms

- Password checkers
  - Proactive checking
  - Reactive checking
- Automated password generation
- Password ageing
  - Require user to change password after some given time
- Limit login attempts
  - Lockout user after a number of failed logins
- Show audit information
  - Inform user about number of failed logins before each login

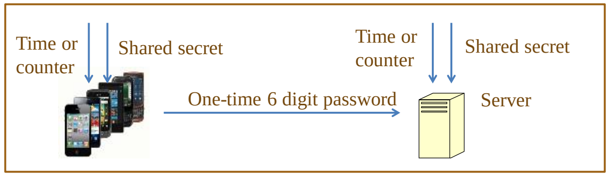
Even More Twitter Hacks

- In Aug 2019, Twitter CEO Jack Dorsey’s account was hacked
- SIM swap attack
  - Convice or bribe operator to switch to new phone number
- Article on Wired
- Compromizes two-factor authentication based on SMS
- Better protection with authenticator application

On the positive side, but completely unrelated, Twitter uses bcrypt to hash passwords

HOTP and TOTP (Something You Have)

- Use a device ”we all have” – the smart phone



- See RFC 4226 for HOTP specification (HMAC-based one-time password algorithm)
  - Based on counter
- See RFC 6238 for TOTP specification (time-based one-time password algorithm)
  - Based on time
- See e.g., Google authenticator for implementation of TOTP

Something You Are (Static Biometrics)

- Examples:
  - Hand geometry
  - Fingerprint
  - Iris
  - Facial characteristics
  - Retinal pattern
  - Voice
  - DNA

Image by John LeMasney, background changed, CC BY-SA 3.0.

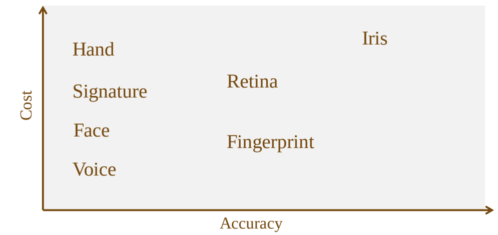
Image by AnaWer, Wikimedia Commons, CC BY-SA 3.0.

Image by Petr Novák, Wikipedia, CC BY-SA 2.5.

Requirements: Uniqueness, Universality, Permanence, Measurability, User friendliness

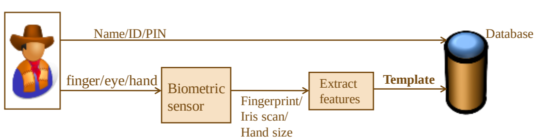
Can be used for both identification and verification

Cost vs. Accuracy



Biometric Systems

- Enrollment, create reference templates

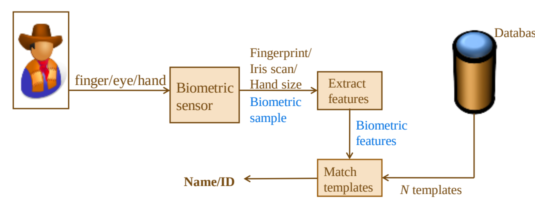


Several templates can be collected for one person

FER – Failure to Enroll Rate

Biometric Systems

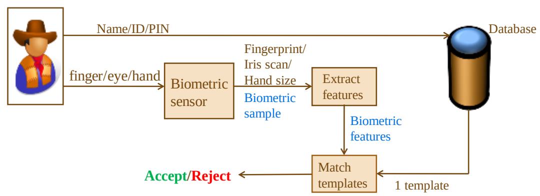
- Identification, 1:N comparison



Identify a user from a database of N people

Biometric Systems

- Verification, 1:1 comparison



Errors

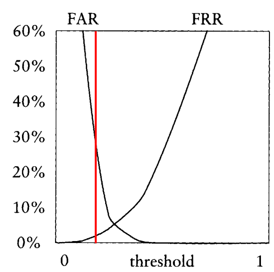
- Measurement will not exactly match reference template. (Different from passwords)
- Two kinds of errors
  - False positives or false matches (Accepting wrong user, security related)
  - False negatives or false non-matches (Rejecting legitimate user, comfort related)
- Matching algorithm used to compare with templates
- The matching is converted to a *score*. Better match gives higher score
- A threshold will determine what the minimum score must be to accept user as valid

FAR & FRR

Graph for typical system

FAR – False Acceptance Rate
FRR – False Rejection Rate

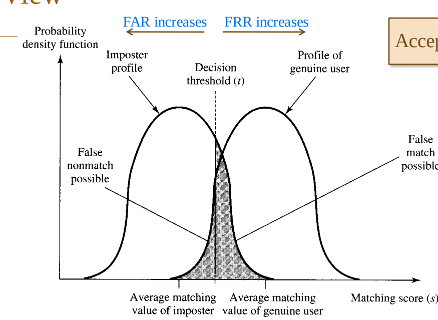
Equal Error Rate (EER) when FAR=FRR



Normalized score on *x*-axis

Another View

Accept if s > t



Placement of threshold *t* provides a tradeoff

LUND UNIVERSITY