Password authentication

Myrto Arapinis
School of Informatics
University of Edinburgh

March 18, 2019

Password authentication

- The question: "who is allowed to access the resources in a computer system?"
- How does the operating system securely identify its users?
- Authentication: determination of the identity of a user

Password authentication

- The question: "who is allowed to access the resources in a computer system?"
- How does the operating system securely identify its users?
- Authentication: determination of the identity of a user
- Standard authentication mechanism: **username** and **password**

User

Username / Password →

← *Success / Failure*

OS

Password authentication

- The question: "who is allowed to access the resources in a computer system?"
- How does the operating system securely identify its users?
- Authentication: determination of the identity of a user
- Standard authentication mechanism: **username** and **password**

User

Username / Password →

← *Success / Failure*

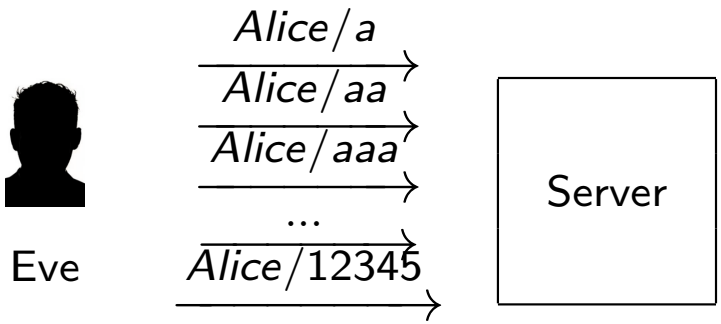
OS / Server

How should passwords be stored?

- Most common password-related attacks target the server

How should passwords be stored?

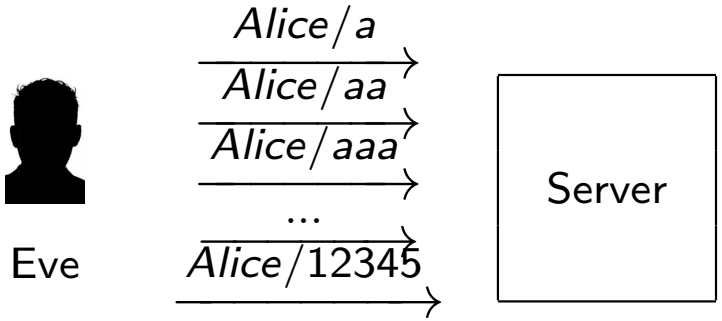
- Most common password-related attacks target the server



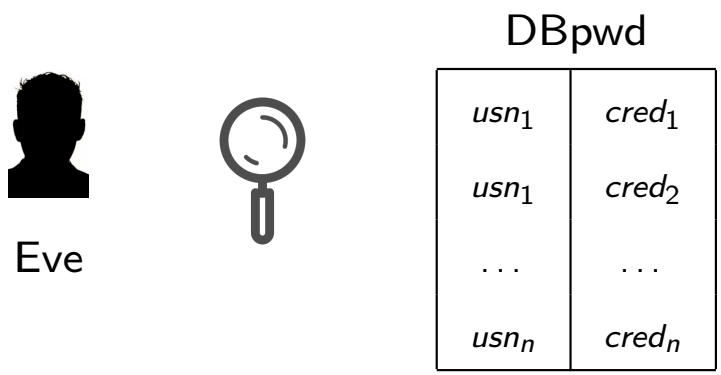
Online attack

How should passwords be stored?

- Most common password-related attacks target the server



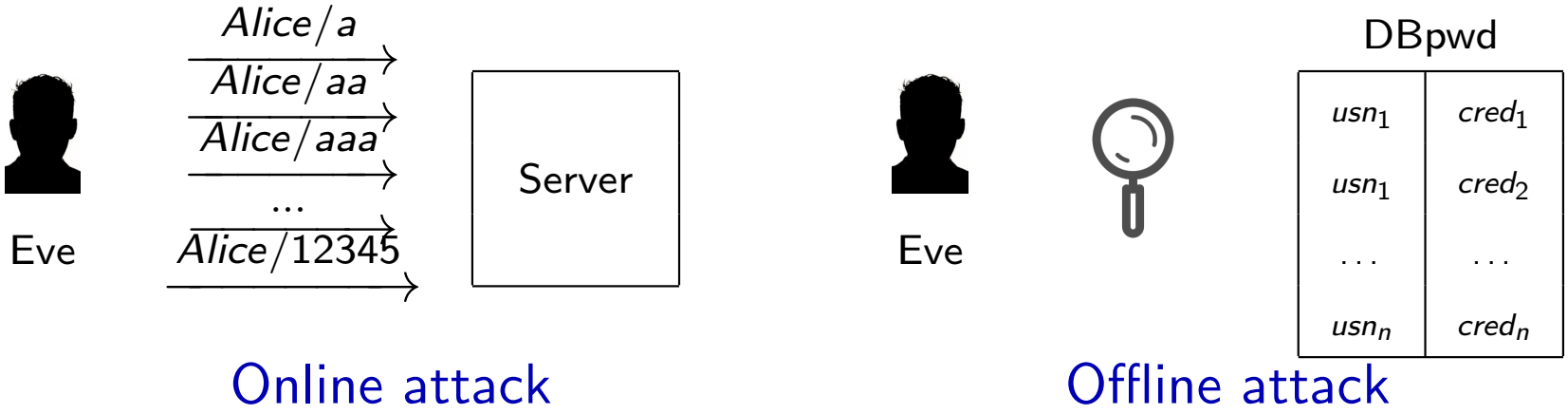
Online attack



Offline attack

How should passwords be stored?

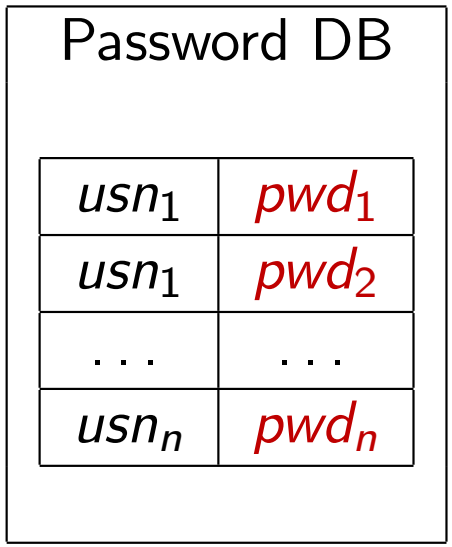
- Most common password-related attacks target the server



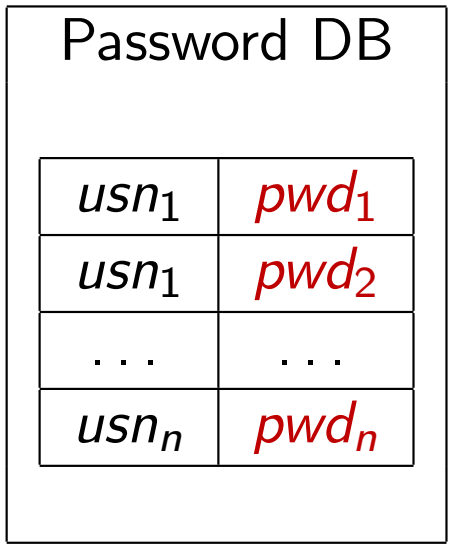
Our goal

Defend from attacks that leak the password database

Attempt #1: store passwords **unencrypted**

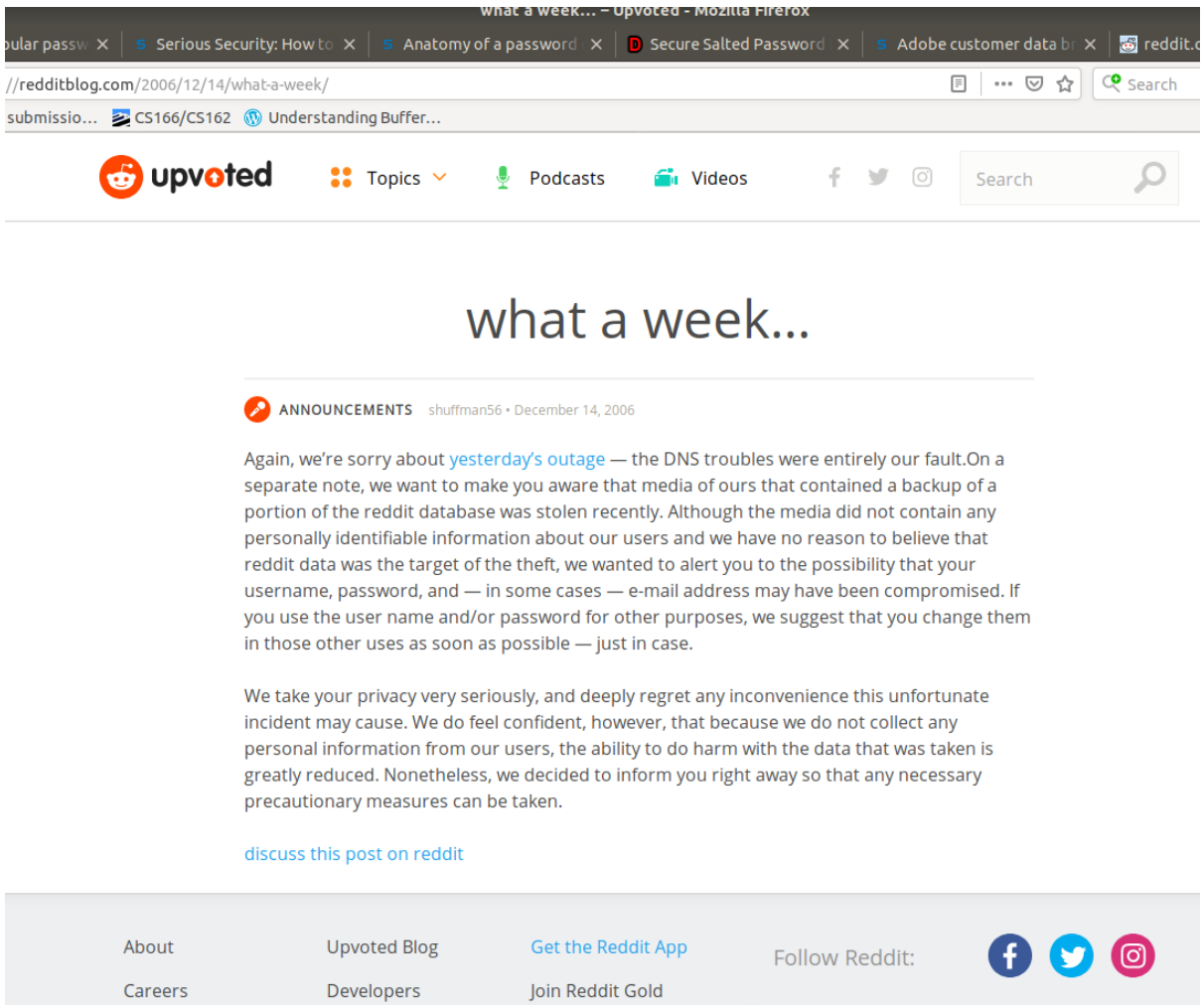


Attempt #1: store passwords **unencrypted**



- Whoever accesses the password DB can login as any user
- Might leak user login information to other services/accounts

Redit password leak (2006)



Attempt #2: **encrypt** passwords

Password DB

k

usn_1	$c_1 = E(k, pwd_1)$
usn_2	$c_2 = E(k, pwd_2)$
...	...
usn_n	$c_n = E(k, pwd_n)$

Attempt #2: **encrypt** passwords

Password DB

k

usn_1	$c_1 = E(k, pwd_1)$
usn_2	$c_2 = E(k, pwd_2)$
...	...
usn_n	$c_n = E(k, pwd_n)$

+ Stolen encrypted passwords cannot be decrypted.
+ Only admins have the key. If a user forgets their password, admins can just look it up for him.

– If attacker managed to steal passwords, why assume the key cannot be stolen?
– Anyone with the key (admins) can view passwords.

Adobe password leak (2013)

- Information on 38 million user accounts leaked
- Adobe pays US $1.2M plus settlements to end breach class action

`https://nakedsecurity.sophos.com/2013/11/04/anatomy-of-a-password-disaster-adobes-giant-sized-cryptographic-blunder/`

Attempt #3: **hash** passwords

Password DB

usn_1	$d_1 = H(pwd_1)$
usn_2	$d_2 = H(pwd_2)$
...	...
usn_n	$d_n = H(pwd_n)$

Attempt #3: **hash** passwords

Password DB

usn_1	$d_1 = H(pwd_1)$
usn_2	$d_2 = H(pwd_2)$
...	...
usn_n	$d_n = H(pwd_n)$

? Stolen hashed passwords cannot easily be cracked (?!)

- Once a hash is cracked, the password is know for all accounts using the same password
- Humans tend to pick weak/guessable passwords
  - Frequency analysis
  - Dictionary attack

Brute force attack

- Try all passwords in a given space
  - κ: number of possible characters
  - ℓ: password length
  - $\rightsquigarrow$ ℓ^κ possible passwords

Brute force attack

- Try all passwords in a given space
  - κ: number of possible characters
  - ℓ: password length
  - $\rightsquigarrow$ ℓ^κ possible passwords

Tips for safe (strong) passwords

Hackers are very good at finding out passwords. They don't simply try to guess them, they get very fast computer programs to try out millions, very quickly. Hackers also know the kind of "tricks" that people use to try to strengthen their passwords.

We advise you memorise a few strong passwords for the systems you use regularly. For services you use less often, find a way to manage those passwords that works for you so that you can look them up, or work them out when you need them.

- University systems require a password length of seven. We recommend you choose more. See "Long passwords" below.
- Use a mix of upper- and lower-case letters, numbers and punctuation marks
- A strong password looks like a random sequence of symbols - use some non-alphabetic characters such as @#$!%+-/:?_
- Use non-dictionary words - like XKCD or one of the other approaches, described below

UoE password guidelines

$\rightsquigarrow$ Assuming a standard 94 characters keyboard, there are $94^7 = 6.4847759e^{+13}$ possible passwords.

Do we need to try all ℓ^κ passwords?

Do we need to try all ℓ^κ passwords?

Rank	2011[4]	2012[5]	2013[6]	2014[7]	2015[8]	2016[3]	2017[9]	2018[10]
1	password	password	123456	123456	123456	123456	123456	123456
2	123456	123456	password	password	password	password	password	password
3	12345678	12345678	12345678	12345	12345678	12345	12345678	123456789
4	qwerty	abc123	qwerty	12345678	qwerty	12345678	qwerty	12345678
5	abc123	qwerty	abc123	qwerty	12345	football	12345	12345
6	monkey	monkey	123456789	123456789	123456789	qwerty	123456789	111111
7	1234567	letmein	111111	1234	football	1234567890	letmein	1234567
8	letmein	dragon	1234567	baseball	1234	1234567	1234567	sunshine
9	trustno1	111111	iloveyou	dragon	1234567	princess	football	qwerty
10	dragon	baseball	adobe123[a]	football	baseball	1234	iloveyou	iloveyou
11	baseball	iloveyou	123123	1234567	welcome	login	admin	princess
12	111111	trustno1	admin	monkey	1234567890	welcome	welcome	admin
13	iloveyou	1234567	1234567890	letmein	abc123	solo	monkey	welcome
14	master	sunshine	letmein	abc123	111111	abc123	login	666666
15	sunshine	master	photoshop[a]	111111	1qaz2wsx	admin	abc123	abc123
16	ashley	123123	1234	mustang	dragon	121212	starwars	football
17	bailey	welcome	monkey	access	master	flower	123123	123123
18	passw0rd	shadow	shadow	shadow	monkey	passw0rd	dragon	monkey
19	shadow	ashley	sunshine	master	letmein	dragon	passw0rd	654321
20	123123	football	12345	michael	login	sunshine	master	!@#$%^&*
21	654321	jesus	password1	superman	princess	master	hello	charlie
22	superman	michael	princess	696969	qwertyuiop	hottie	freedom	aa123456
23	qazwsx	ninja	azerty	123123	solo	loveme	whatever	donald
24	michael	mustang	trustno1	batman	passw0rd	zaq1zaq1	qazwsx	password1
25	Football	password1	000000	trustno1	starwars	password1	trustno1	qwerty123

- (2016) the 25 most common passwords made up more than 10% of surveyed passwords.
- Most common password of 2016, "123456", makes up 4% of surveyed passwords.
- 30% of password surveyed in top 10000

Dictionary attack

- Try the top N most common passwords,
- Try words in English dictionary,
- Try names, places, notable dates,
- Try Combinations of the above,
- Try the above replacing some characters with digits and symbols e.g. : iloveyou, il0vey0u, i10v3y0u, ...

Dictionary attack

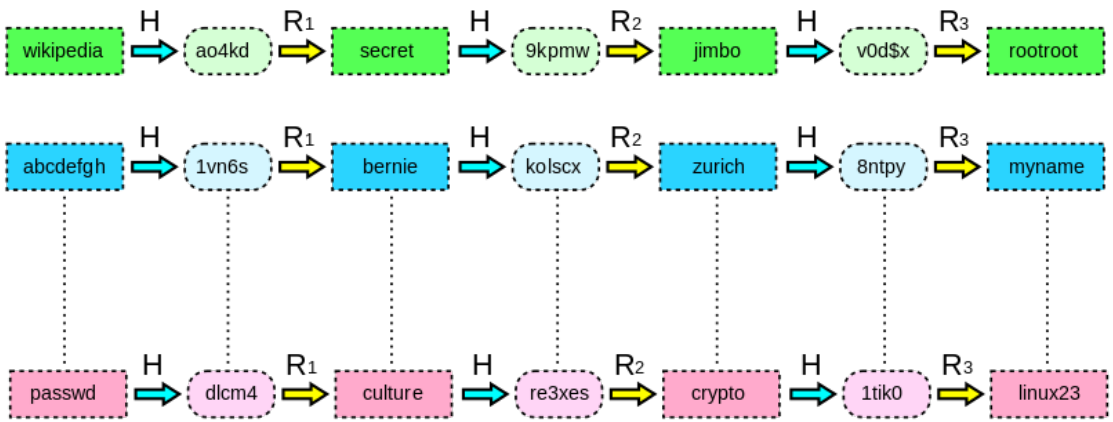
- Try the top N most common passwords,
- Try words in English dictionary,
- Try names, places, notable dates,
- Try Combinations of the above,
- Try the above replacing some characters with digits and symbols e.g. : iloveyou, il0vey0u, i10v3y0u, ...
- UoE: password guidelines `https://www.ed.ac.uk/infosec/how-to-protect/lock-your-devices/passwords`

Rainbow tables - the basic idea

- Assume H is a one-way hash function mapping n bits to n bits. Let $N = 2^n$.
- Assume H cycles through all the values of the domain: starting with a password p, and then applying $H()$ 2^n times will cycle through all possible values ($\{0,1\}^n$).
- We can then crack a password using $sqrt(N)$ space in $sqrt(N)$ time.
- Start with an arbitrary password p, and compute h_1, h_2, $\ldots h_N$ and store in a hash table
  - h_1 $h_{sqrt(N)}$
  - $h_{sqrt(N)+1}$ $h_{2sqrt(N)}$
  - $\ldots$
  - $h_{(sqrt(N)-1)sqrt(N)}$ h_N
- Given a hash to break h_i, start computing $h(\ldots H(H(h_i))\ldots)$ and you will hit an endpoint above. Get the starting point and start developing the chain until you hit the password.

Rainbow tables

- Assuming that H cycles through all values of the domain is unrealistic.
- Assume t "reduction functions" $R_i : D_H \to D_P$ where D_P is the domain of passwords, and D_H is the range of the H.
  Example reduction function: If passwords are 16 bits and hashes are 256 bits, keep 16 equally distributed bits from the 256 bits
- Pick m passwords $p_1, p_2, \ldots, p_m$ and develop chains, each containing t elements. Store the start points and the endpoints. Then given a hash h, start developing chains until an endpoint is hit. Then go the start point to retrieve the password



Wikipedia: Simplified rainbow table with 3 reduction functions

LinkedIn password leak (2012)

- In June 2012, it was announced that almost 6.5 million linked in passwords were leaked and posted on a hacker website

Attempt #3: **salt and hash** passwords

Password DB

usn_1	s_1	$d_1 = H(s_1 \| \| pwd_1)$
usn_2	s_2	$d_2 = H(s_2 \| \| pwd_2)$
...	...	
usn_n	s_n	$d_n = H(s_n \| \| pwd_n)$

Attempt #3: **salt and hash** passwords

Password DB

usn_1	s_1	$d_1 = H(s_1 \| \| pwd_1)$
usn_2	s_2	$d_2 = H(s_2 \| \| pwd_2)$
...	...	
usn_n	s_n	$d_n = H(s_n \| \| pwd_n)$

+ Since every user has different salt, identical passwords will not have identical hashes
+ No frequency analysis
+ No precomputation: when salting one cannot use preexisting tables to crack passwords easily

What we learned today

1. Password authentication
   - 1.1 principles
   - 1.2 online/offline attacks
2. Password cracking
   - 2.1 Brute force attack
   - 2.2 Dictionary attack
   - 2.3 Rainbow tables
3. How to store passwords:
   - store salted hashes of passwords