



EECS 3482

Introduction to Computer Security

Password Cracking

Instructor: N. Vlajic, Fall 2021

TIME IT TAKES A HACKER TO BRUTE FORCE YOUR PASSWORD

Number of characters	Number only	Lower case letters	Upper and lowercase letters	Numbers, upper and lowercase letters	Numbers, upper and lowercase letters, symbols
4	Instantly	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	Instantly	Instantly	Instantly
6	Instantly	Instantly	Instantly	1 sec	5 sec
7	Instantly	Instantly	25 secs	1 min	6 min
8	Instantly	5 secs	22 mins	1 hour	8 hour
9	Instantly	2 mins	19 hours	3 days	3 weeks
10	Instantly	58 mins	1 month	7 months	5 years
11	2 secs	1 day	5 years	41 years	400 years
12	25 secs	3 weeks	300 years	2k years	34k years
13	4 mins	1 year	16k years	100k years	2m years
14	41 mins	51 years	800k years	9m years	200m years
15	6 hours	1k years	43m years	600m years	15bn years
16	2 days	34k years	2bn years	37bn years	1tn years
17	2 weeks	800k years	100 bn years	2tn years	93tn years
18	9 months	23m years	6tn years	100tn years	7qd years

Go to howsecureismypassword.net for more information



BEAUCERON
SECURITY

Introduction

- **Password** – a secret word/string of characters used to authenticate a user into a system
 - ◆ critical (often only) defense against intruders
 - ◆ ideal password: easy to remember, hard to 'crack'
 - ◆ Google frequently releases **lists of common password types which are insecure** as they are too easy to guess / get off social media
 - name of a pet, child, family member, spouse
 - names of birthplaces, favorite sports teams
 - birthdays, anniversary dates
 - ◆ **overly complex passwords are as dangerous as very simple ones**
 - the user likely to write it down or to reuse it

In 2019, the UK's **National Cyber Security Centre (NCSC)** has released a list of the 100,000 most common passwords to appear in breached accounts.

Using data from [Have I Been Pwned](#) (HIBP), a website that allows users to check if their email addresses or passwords have appeared in a known data breach, the United Kingdom's [National Cyber Security Centre \(NCSC\)](#) [has found](#) that 23.2 million user accounts worldwide were “secured” with ‘123456’.

Most used in total	Names	Premier League football teams	Musicians	Fictional characters
123456 (23.2m)	ashley (432,276)	liverpool (280,723)	blink182 (285,706)	superman (333,139)
123456789 (7.7m)	michael (425,291)	chelsea (216,677)	50cent (191,153)	naruto (242,749)
qwerty (3.8m)	daniel (368,227)	arsenal (179,095)	eminem (167,983)	tigger (237,290_)
password (3.6m)	jessica (324,125)	manutd (59,440)	metallica (140,841)	pokemon (226,947)
111111 (3.1m)	charlie (308,939)	everton (46,619)	slipknot (140,833)	batman (203,116)



Pwned websites

Breached websites that have been loaded into Have I Been Pwned

Here's an overview of the various breaches that have been consolidated into this Have I Been Pwned. These are accessible programmatically via [the HIBP API](#) and also [via the RSS feed](#).



000webhost

In approximately March 2015, the free web hosting provider 000webhost suffered a major data breach that exposed almost 15 million customer records. The data was sold and traded before 000webhost was alerted in October. The breach included names, email addresses and plain text passwords.

Breach date: 1 March 2015

Date added to HIBP: 26 October 2015

Compromised accounts: 14,936,670



123RF

In March 2020, the stock photo site 123RF suffered a data breach which impacted over 8 million subscribers and was subsequently sold online. The breach included email, IP and physical addresses, names, phone numbers and passwords stored as MD5 hashes. The data was provided to HIBP by dehashed.com.

Breach date: 22 March 2020

Date added to HIBP: 15 November 2020

Compromised accounts: 8,661,578

Compromised data: Email addresses, IP addresses, Names, Passwords, Phone numbers, Physical addresses, Usernames

[Permalink](#)

[Home](#)[Notify me](#)[Domain search](#)[Who's been pwned](#)[Passwords](#)[API](#)[About](#)[Donate !\[\]\(3211b5d1d968fc1665909b34f9f16010_img.jpg\) !\[\]\(d47ad152ec3d86a04ad64c8049e1f17f_img.jpg\)](#)

';--have i been pwned?

Check if you have an account that has been compromised in a data breach

pwned?

Pwned Passwords

Pwned Passwords are 555,278,657 real world passwords previously exposed in data breaches. This exposure makes them unsuitable for ongoing use as they're at much greater risk of being used to take over other accounts. They're searchable online below as well as being downloadable for use in other online systems. [Read more about how HIBP protects the privacy of searched passwords.](#)

pwned?

Oh no — pwned!

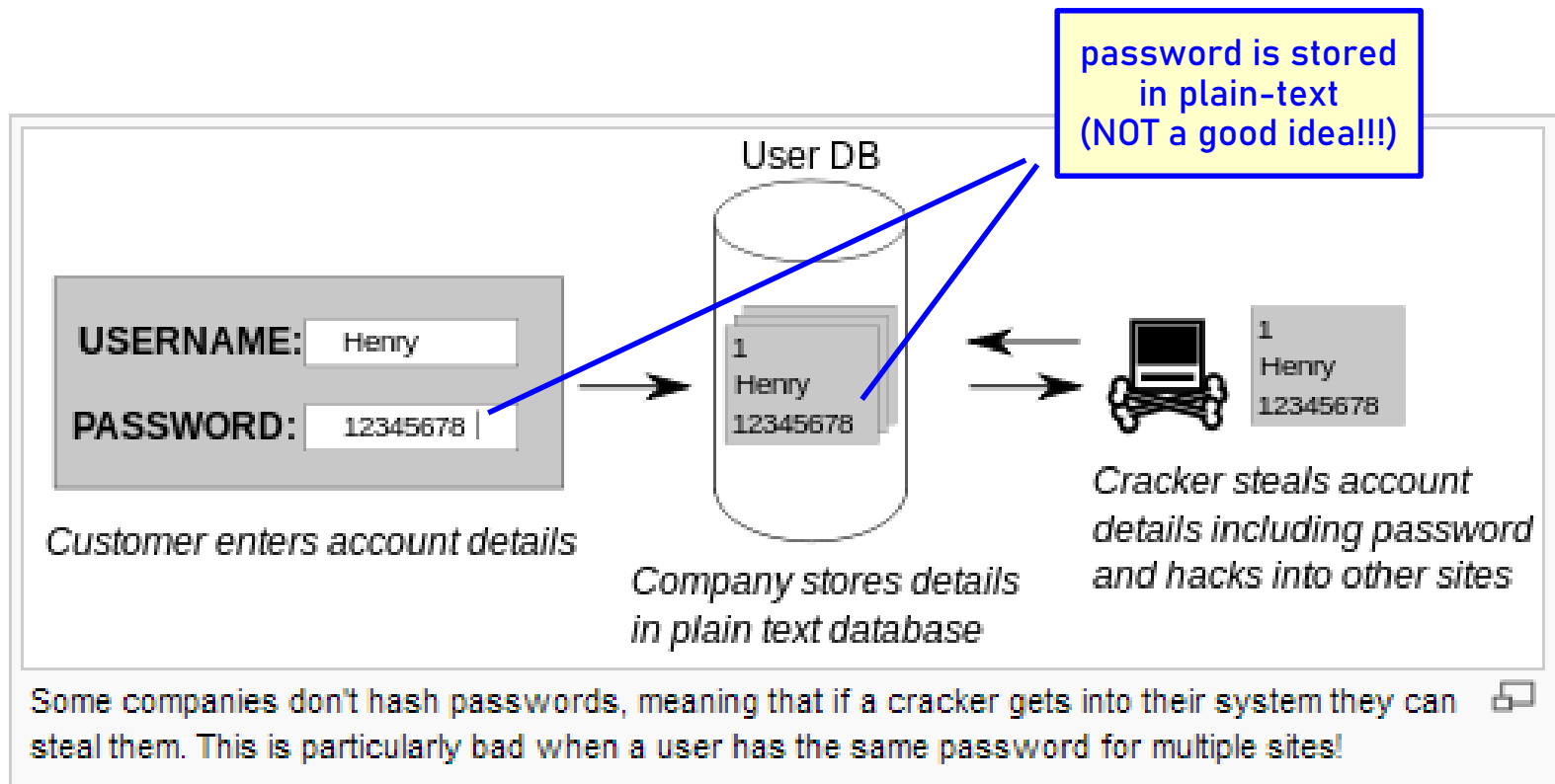
This password has been seen 23,547,453 times before

This password has previously appeared in a data breach and should never be used. If you've ever used it anywhere before, change it!

<https://haveibeenpwned.com/Passwords>

Introduction (cont.)

How are passwords stored in a computer/system???

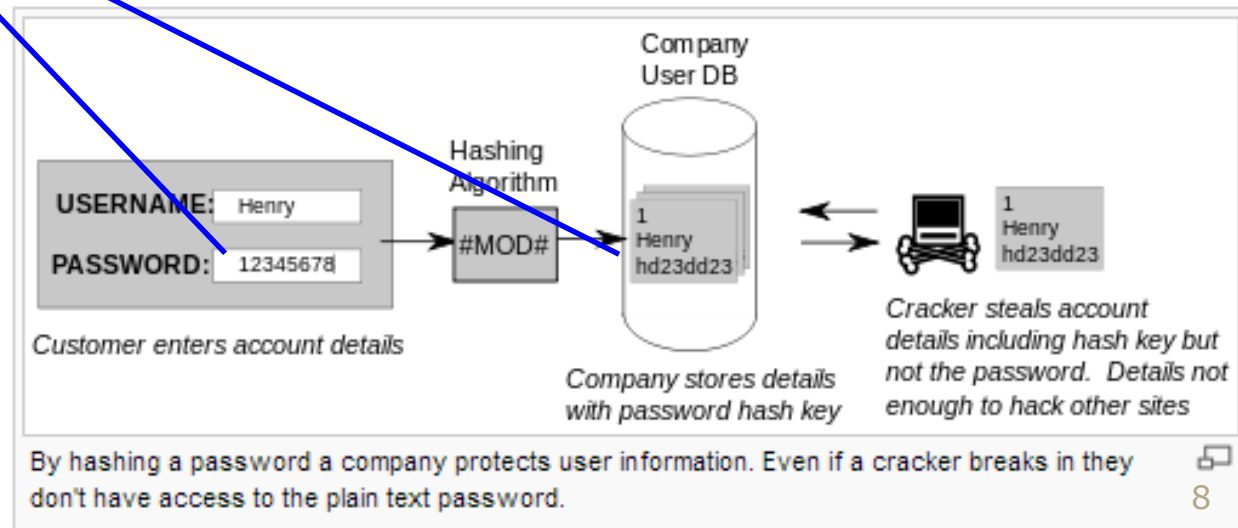


Introduction (cont.)

- **Passwords in a System**

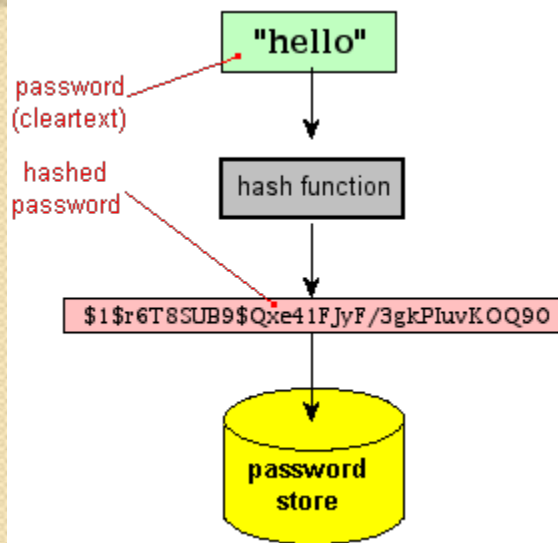
password is stored in a 'hashed form'

- ◆ in most systems, passwords are stored in a protected (hash) form \Rightarrow snooper that gains internal access to system cannot easily retrieve/steal passwords
 - every time a user logs in, password handling software runs the hash algorithm
 - if (new hash = stored hash), access is granted



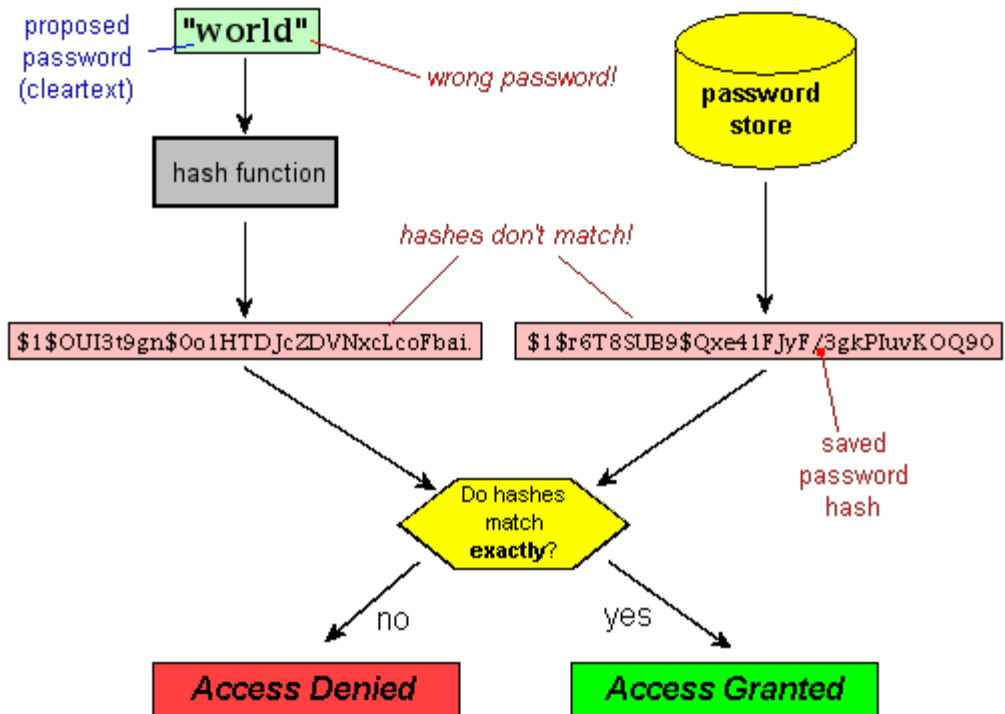
Introduction (cont.)

Example: Password hashes



account creation stage:

storing hash instead
of password



logging into an existing account:

testing a password against stored hash

Password Cracking (cont.)

- **Online vs. Offline Password Cracking**

- ◆ **online cracking**

- try every password at login prompt in real time

- very slow!

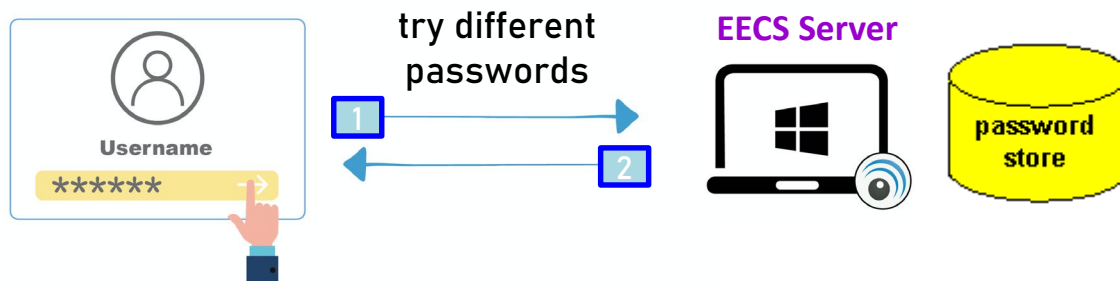
8-character password of 76 possible characters (upper & lower case, digits, common symbols) = 1.1×10^{15} possibilities

2 to 3 passwords a second \Rightarrow 5,878,324 years to guess a password

- extremely noisy!

most systems block the victim account after several failed login attempts

extremely slow and noisy !



Password Cracking (cont.)

- **Online vs. Offline Password Cracking**

- ◆ **off-line cracking**

- assumes the possession of passwd/hash file

Dictionary

michael	shadow	diamond	myspace	inuyasha	mustang
ashley	melissa	carolina	rebelde	peaches	isabel
qwerty	eminem	steven	angel1	veronica	natalie
111111	matthew	rangers	ricardo	chris	cutieko
iloveu	robert	louisie	babygirl	888888	javier
000000	danielle	orange	heaven	adriana	789456123
michelle	forever	789456	baseball	cutie	123454
tigger	family	999999	martin	james	sarah
sunshine	jonathan	shorty	greenday	banana	bowwow
chocolate	987654321	nathan	november	friend	portugal
password1	computer	alissa	crystal	jesus1	laura
soccer	whatever	madison	celtic	marvin	777777
anthony	dragon	muther	123321	edward	denise
friends	vanessa	hunter	123abc	oliver	tigers
butterfly	cookie	killer	mahalkita	diana	volleyball
purple	naruto	sandra	batman	samsung	jasper
angel	summer	alejandro	september	freedom	rockstar
jordan	sweetie	buster	december	angelo	january
liverpool	spongebob	george	morgan	angel	jackoff
justin	joseph			brwnth	alicia
					nicholas

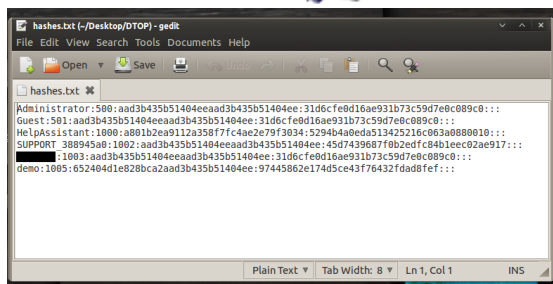
look for matches

hash fuctions are one-way – search process involved !



steal password file

EECS Server

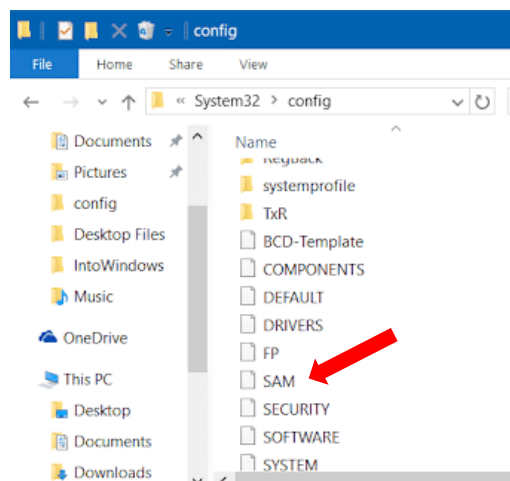


Where is the password file ?!

Introduction (cont.)

- **Password Management in Windows** – password hashes are stored in **Security Account Manager (SAM) file**

◆ stored in C:\Windows\System32\config or HKEY_LOCAL_MACHINE\SAM registry
- neither of them can be opened/copied on normal boot-up of the OS (i.e., while computer running) – file used by OS



- **Accessing SAM File in Windows** – requires **administrative privileges** to be copied / dumped

Just open the Command Prompt as Administrator, and then run the following commands:

```
reg save HKLM\SAM C:\sam  
reg save HKLM\SYSTEM C:\system
```

Best match



Command Prompt

App

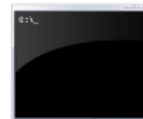
Search the web



cmd - See web results



Settings (1)



Command Prompt

App



Open



Run as administrator



Open file location



Pin to Start



Pin to taskbar



cmd|


```
Select Administrator: Command Prompt

Microsoft Windows [Version 10.0.18363.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>reg save HKLM\SAM C:\sam
The operation completed successfully.

C:\WINDOWS\system32>reg save HKLM\SYSTEM C:\system
The operation completed successfully.
```

Copy of SAM file is now stored on C drive as a file named 'sam'. However, this file is encrypted using SysKey!!! So, a dump of SYSTEM hive/file is also needed!

The SAM file is encrypted with the **SysKey** which is stored in %SystemRoot%\system32\config\system file.

During the boot-time of Windows the hashes from the SAM file get decrypted using the SysKey and these hashes are then loaded to the registry and used for authentication purpose.

Both system and SAM files are unavailable (i.e., locked by kernel) during Windows' runtime.

Tools like **mimikatz** (on Windows) and **samdump2** (on Linux) can be used to extract hashes from SAM:

```
root@kali:~/Desktop# samdump2 system sam
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
:504:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
```

Tools like Cain and Abel or John the Ripper can be used to 'crack' hashed passwords extracted from sam file ...

Introduction (cont.)

- **Accessing Hash File in Unix** – text file: `/etc/shadow` (`/etc/passwd`)
 - ◆ readable by system administrator (root) only

getent = get entry

```
[root@arch ~]# getent shadow aldin
aldin:$6$z75S8I2d3kQL6cI1$DC.Ev/W21DKFCM6BA0VfS1fzW3UJknLHzvEof
```

When a new user is created in Linux it affects 4 files

- `/etc/passwd`
- `/etc/group`
- `/etc/shadow`
- `/etc/gshadow`


`/etc/passwd` file is essentially the *user account database* in which Linux stores valid accounts and related information about these accounts; typically has file system permissions that allow it to be readable by all users of the system

`/etc/shadow` file contains hashed passwords and bookkeeping information; accessible only by the super user

Introduction (cont.)

Example: passwd vs. shadow

cd /etc

 indigo.cs.yorku.ca - PuTTY

```
indigo 28 % ls -l passwd
-rw-r--r-- 1 root root 166910 Mar  6 16:20 passwd
```

```
indigo 29 % ls -l shadow
-rw----- 1 root root 70669 Mar  7 22:50 shadow
```

Structure of /etc/passwd file:

ajay:x:100:100:ajay:/home/ajay:/bin/bash

↓ ↓ ↓ ↓ ↓ ↓ ↓
1 2 3 4 5 6 7

1. Username
2. Password: An x character indicates that password is encrypted and stored in /etc/shadow file.
3. UID (User ID)
4. GID (Group ID)
5. User Information
6. Home Directory: detailed path of home directory of the user.
7. Shell

Introduction (cont.)

Example: entry in etc/shadow

vivek:\$1\$fnfffc\$GteyHdicpGOfffXX4ow#5:13064:0:99999:7:::

1 2 3 4 5 6

ID of used
hashing
algorithm

1. **\$1\$** is MD5
2. **\$2a\$** is Blowfish
3. **\$2y\$** is Blowfish
4. **\$5\$** is SHA-256
5. **\$6\$** is SHA-512

1. **Username** : It is your login name.
2. **Password** : It is your encrypted password. The password should be minimum 6-8 characters long including special characters/digits and more.
3. **Last password change (lastchanged)** : Days since Jan 1, 1970 that password was last changed
4. **Minimum** : The minimum number of days required between password changes i.e. the number of days left before the user is allowed to change his/her password
5. **Maximum** : The maximum number of days the password is valid (after that user is forced to change his/her password)
6. **Warn** : The number of days before password is to expire that user is warned that his/her password must be changed
7. **Inactive** : The number of days after password expires that account is disabled
8. **Expire** : days since Jan 1, 1970 that account is disabled i.e. an absolute date specifying when the login may no longer be used.

Typically
set to 0.

Introduction (cont.)

Example: 'encrypted' password

```
$6$5H0QpwprRiJQR19Y$bXGOh7dIfOWpUb/Tuqr7yQVCqL3UkrJns9.7msfvMg4Z0/PsFC5Tbt32P
| 1 | ----- 2 ----- | ----- 3 -----
```

1. **Hash Algorithm:** This field denotes the hashing algorithm used to create the hashed password. The digit 6 describes that, SHA-512 algorithm is used, in this case. Some more of them are enlisted below:

```
-----
| 1 | MD5 |
-----
| 2 | Blowfish |
-----
| 2a | eksBlowfish |
-----
| 5 | SHA-256 |
-----
| 6 | SHA-512 |
-----
```

2. **Salt Value:** Salt values are used to make the hash value stronger. These are the random type of data that is used to combine with the original password and then the hashed version of that is used as the encrypted password.

3. **Password:** This field stores the hashed version of the combination of original password and salt value.

Password Cracking (cont.)



123456

longer allowed size of the password => more combinations have to be tried

hacker makes a password guess

hash function

\$1\$OUI3t9gn\$0o1HTDJcZDVNxcLcoFbai.

stolen password hash

\$1\$r6T8SUB9\$Qxe41FJyF/3gkPluvKOQ90

Do hashes
match
exactly?

no

yes

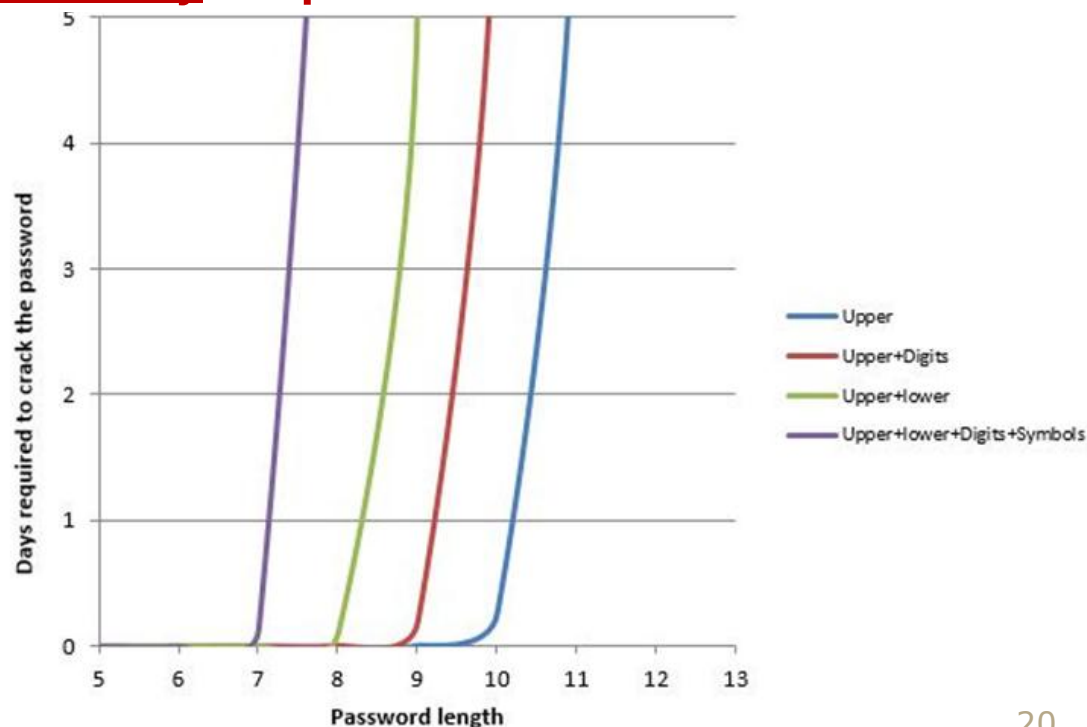
try another password

cracking successful !

In the case of **brute-force** password cracking, there is no particular strategy when generating password guesses. The entire possible space of passwords is explored.

Password Cracking

- **Password Cracking (Guessing)** – a method of gaining unauthorized access to a computer system by trying different passwords
 - ◆ cracking difficulty ~ size of password space & 'diversity' of password characters



Password Cracking (cont.)

- **Brute-Force Password Cracking** – aka exhaustive password search
 - ◆ **entire password space is ‘tried’**
 - ◆ starts by using simple combinations of characters, and then gradually moves to more complex/longer ones
 - ◆ (may be) effective for passwords of small size, but too time consuming for long passwords
 - ◆ examples of brute-force crackers
 - **Cryptool**
 - **Cain and Able**
 - **John the Ripper**
 - **Ophcrack**

Password Cracking (cont.)

What is Password Search Space in Brute-Force Attacks?

a) On 26-letter alphabet, password of length exactly 1/2/n:

?

$$S_{1\text{-Letter}} = 26^1$$

? ?

$$S_{2\text{-Letter}} = 26 * 26 = 26^2$$

? ? ... ?

$$S_{n\text{-Letter}} = 26 * 26 * \dots * 26 = 26^n$$

b) On A-character alphabet (lett. + numb.), passw. of length n:

$$S_{n\text{-character}} = A^n = 36^n$$

c) On A-character alphabet, passwords up-to n characters

$$S_{\text{varying-size}} = \sum_{i=1}^n A^i = \frac{A^{n+1} - 1}{A - 1}$$

Password Cracking (cont.)

Example: Brute-Force Password Search Space

Tina has to create a password for the security of a software program file. She wants to use a password with 3 letters.

How many passwords are allowed if no letter is repeated and the password is not case sensitive?

$$L_1 L_2 L_3 : A (B-Z) (C-Z)$$
$$26 \quad 25 \quad 24$$

$$26 * 25 * 24 = 15,600$$

Password Cracking (cont.)

Example: Brute-Force Password Search Space (3)

A system allows passwords consisting of 4 lower-case letters followed by 3 digit numbers.

How many passwords are possible if there are no restrictions.

$$\underbrace{L_1 \ L_2 \ L_3 \ L_4}_{\text{any combination}} \underbrace{D_1 \ D_2 \ D_3}_{\text{any combination}}$$

$$26^4 * 10^3 = 456,976,000$$

Password Cracking (cont.)

Example: Brute-Force Password Search Space

Compare two systems in which the maximum password length is 16, and passwords may contain any printable ASCII characters.

System A allows passwords to be any length (1-16)

System B requires passwords to be at least 8 characters long (8-16).

Calculate the search space for these two systems.

A=94

System A:
$$S_{1-16} = \sum_{i=1}^{16} A^i = \frac{A^{16+1}-1}{A-1} = 3.75 \cdot 10^{31}$$

System B:
$$S_{8-16} = \sum_{i=8}^{16} A^i = \sum_{i=1}^{16} A^i - \sum_{i=1}^7 A^i =$$
$$\approx 3.75 \cdot 10^{31}$$

Almost the same!!!

Password Cracking (cont.)

- **Biased Attack** ♦ the search space is further reduced by focusing on most likely combinations of words and/or numbers ...

Example: Biased Attack on 4-Digit Pins

Assume a system requires that access passwords be comprised of 4 digits.

Total unbiased search space: any number between 0000 – 9999 (10,000)

Many people use some important personal dates to generate 4-digit passwords.

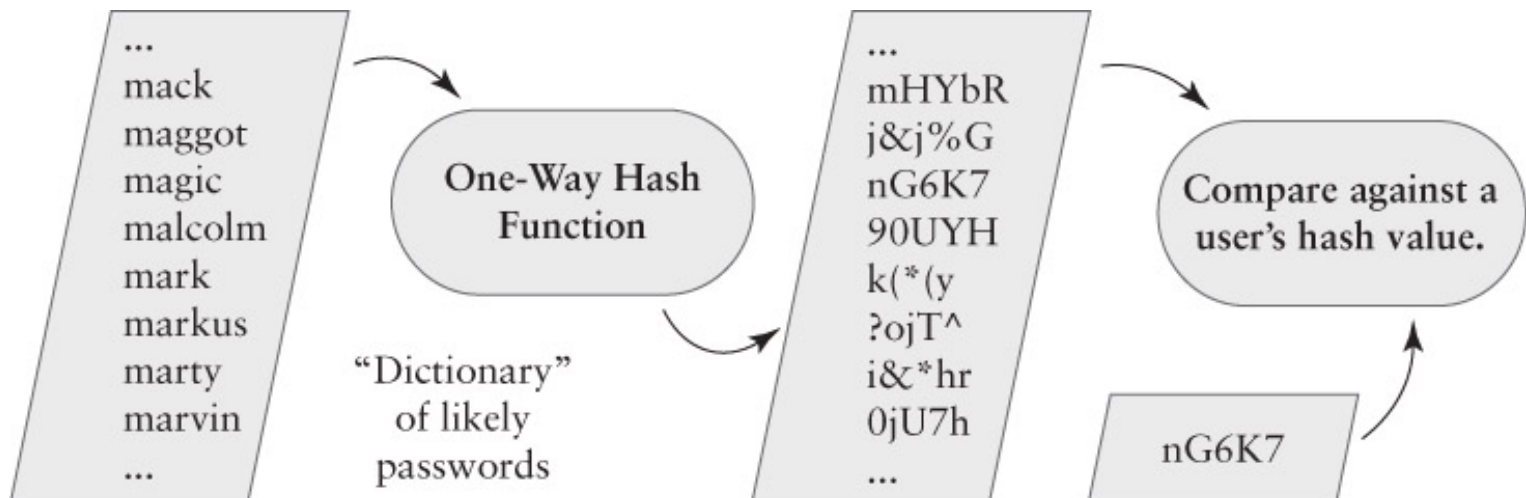
Biased search space: **only 366 possible combinations!**

Password Cracking (cont.)

- **Dictionary Attack**

- ◆ **users often create passwords using common dictionary words**

- instead of trying every password, dictionary attack probes only common dictionary words
- **faster than brute force**, as it uses smaller (more likely) search space
- still might take considerable time, and **might fail in the end**



Password Cracking (cont.)

Example: Dictionary Attacks in Real World

Many studies on effectiveness of dictionary attack have been conducted.

Not 100% effective, but enough passwords were cracked to make the use of this attack worthwhile.

Research or Incident	% Guessed
Morris worm, estimated success (1988)	~50%
Klein's Study (1990)	24.2%
Spafford's Study (1992)	20%
CERT Incident 1998-03	25.6%
Cambridge study by Yan, et al. (2000)	35%
Lulz and Anonymous, estimated success (2011)	30%

Password Cracking (cont.)

- **Pre-Computed Dictionary Attacks**

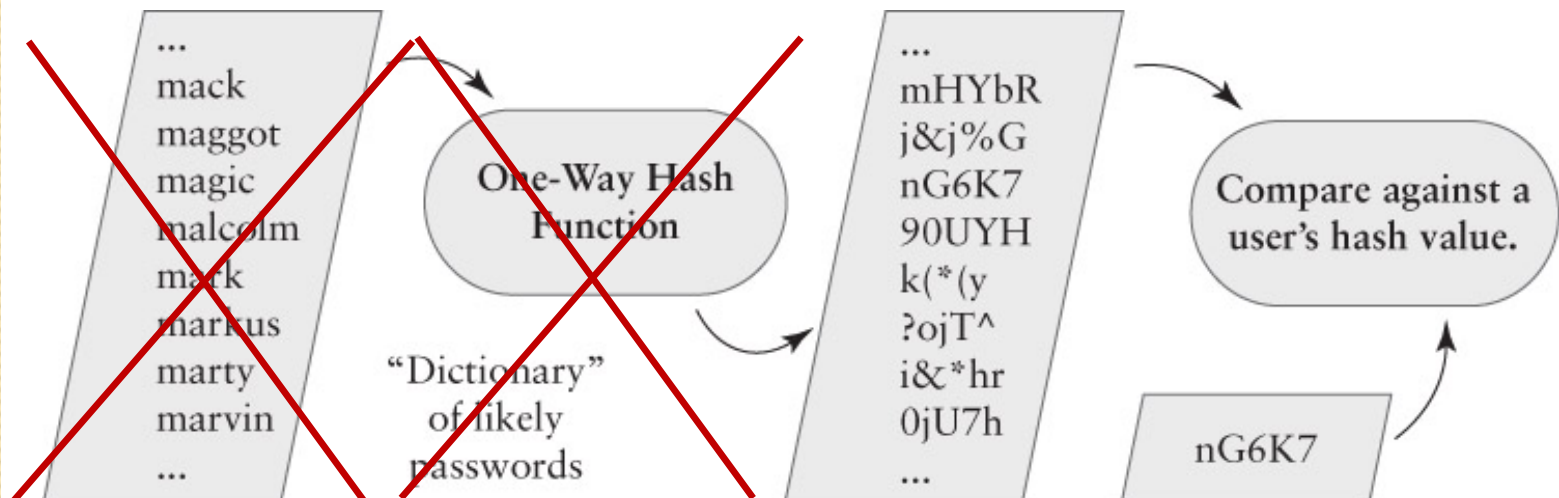
- ◆ achieves **TIME-SPACE tradeoff** by pre-computing a list of hashes of dictionary words

- pre-computed hashes are compared against those in a stolen password file

- **rainbow tables**

- 1) pregenerated sets/lists of hashes – **$n \times \text{Gbyte size!!!}$** 🙅

- 2) allow extremely rapid searching 🙌



Password Cracking (cont.)

Password Characteristics	Example	Maximum time to break using brute force	Maximum time to break using rainbow tables
8-digit password of all letters	abcdefgh	1.6 days	28 minutes
9-digit password of letters and numbers (mixed case)	AbC4E8Gh	378 years	28 minutes
10-digit password of letters and numbers (mixed case)	Ab4C7EfGh2	23,481 years	28 minutes
14-digit password of letters, numbers, and symbols	1A2*3&def456G\$	6.09e + 12 years	28 minutes

Table 7-5 Times to break a hash

Rainbow Table: time/space tradeoff!!!

Password Salting

- **Password Salting** – adding a unique random value to each password before hashing

It is hard, if not impossible, to prevent users from choosing 'weak' passwords
So, ideally, the system would additionally 'strengthen' user passwords.

- ◆ both the hash and salt are stored
- ◆ **does not fully prevent against password cracking, but makes it harder / more time consuming**

Found in most attack dictionaries and rainbow tables!

hello

add salt

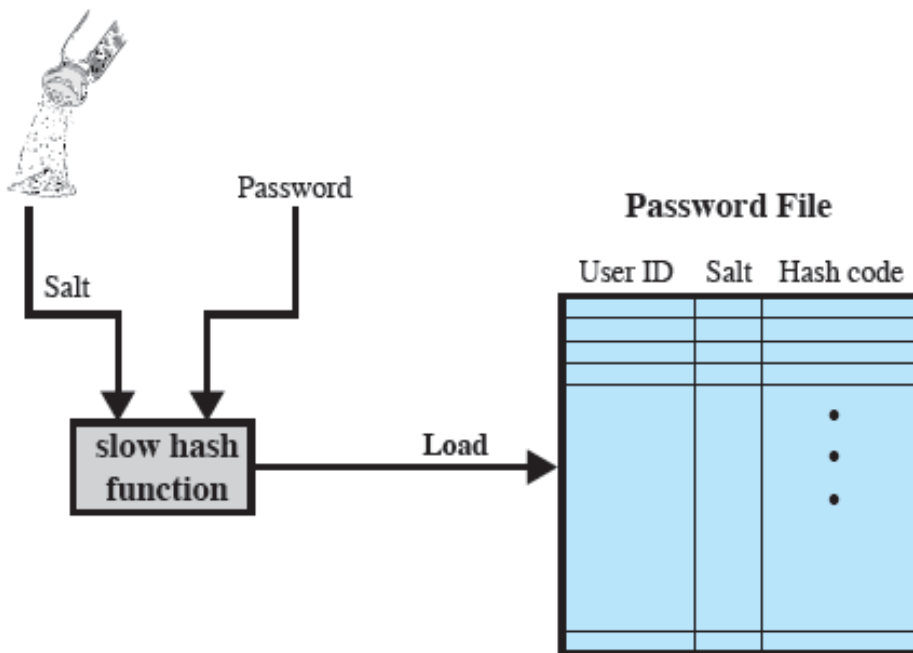
hello3ab9

Cannot be found in common dictionaries or rainbow tables !

hash function

39e19b234...

store hash and salt

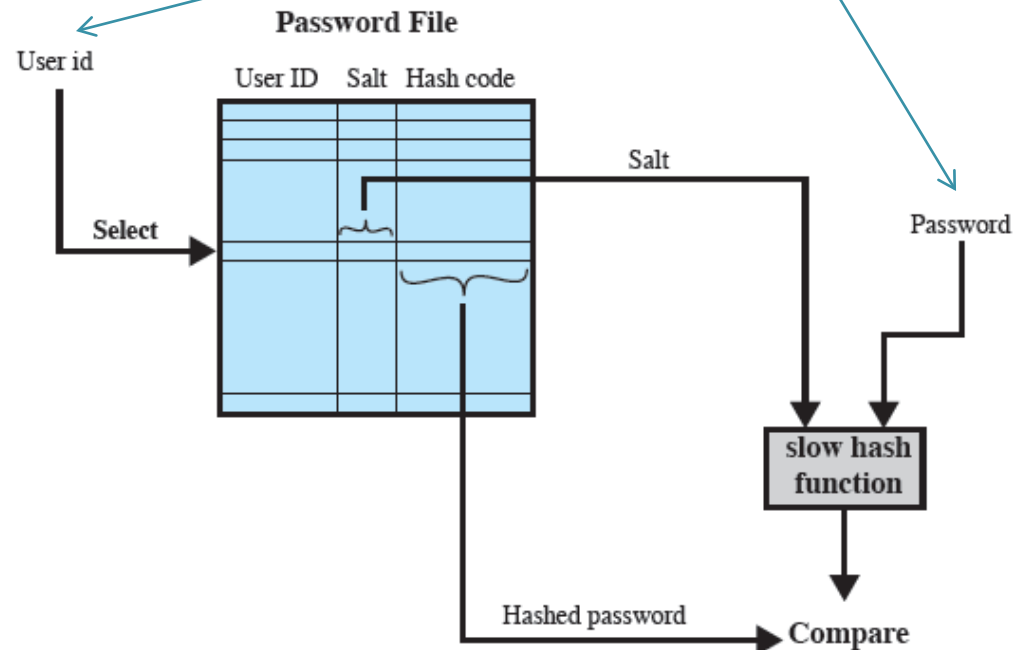


(a) Loading a new password

account creation stage:
storing hash & salt
instead of password

logging into an existing account:
testing a password against stored hash

User id + Password



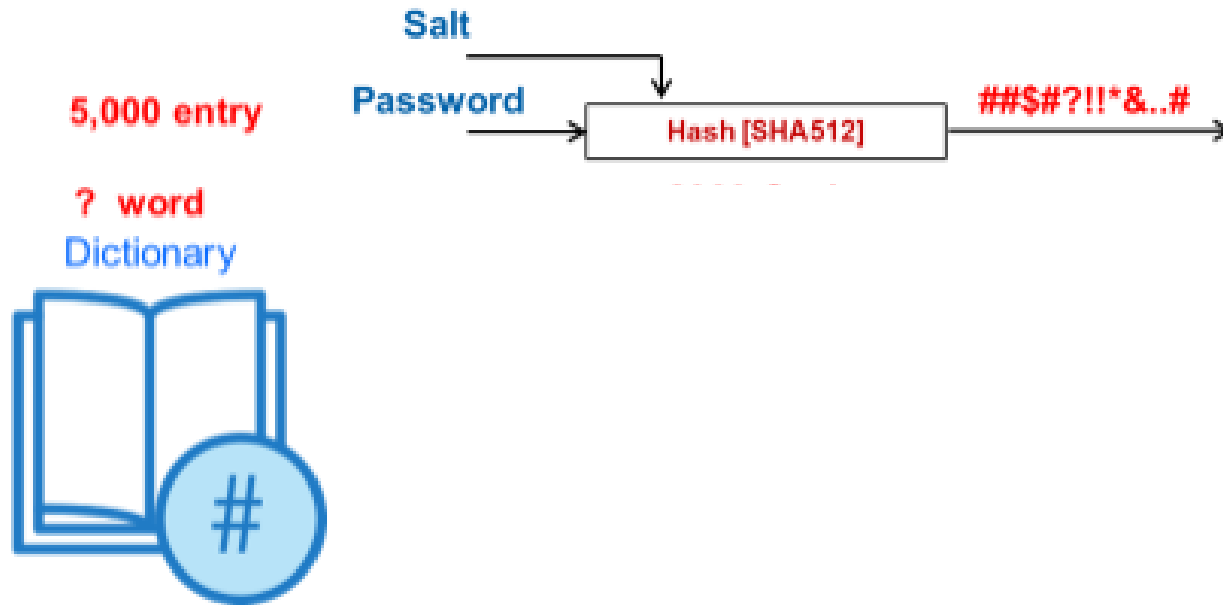
(b) Verifying a password

Password Salting (cont.)

Example: Attack on salted passwords ...

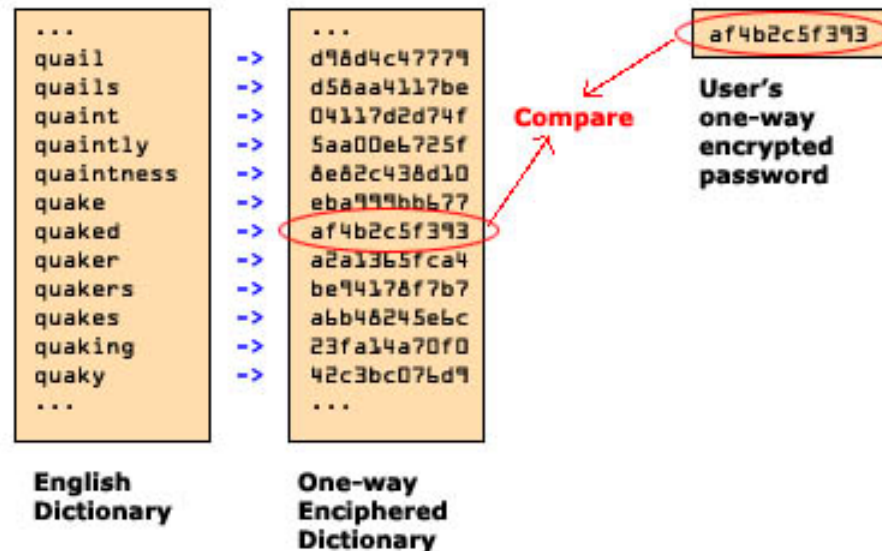
For every word in a dictionary (or an 'extended' dictionary):

- 1) add the User's salt
- 2) hash
- 3) compare



Password Salting (cont.)

- **Password Salting Benefits** – in case of a compromised Password File
 - (simple) **dictionary** and **rainbow attacks** impossible to perform
 - prevents duplicate passwords from being visible in password file
 - becomes impossible to find out whether a person has used the same password on multiple systems



Password Example



Example: Password policies – which one is better?!

Company A requires that its employees pick 6-character passwords made up of combinations of lowercase letters, uppercase letters, and digits (62 possibilities). No other characters are allowed, and a given user's password must not use any character twice.

Example: ab98CD

Company B requires that its employees pick 12-character passwords, where each of the 12 can be any of 100 possible characters. Unlike for Company A, Company B's employees can reuse characters in their passwords. However, Company B finds that users often make mistakes with these long passwords, so if an authentication attempt fails, the login server helps the user by telling them how many of the initial letters were correct. For example, if a password entered was 'ab**cd**efgij' and the server replies "*Wrong, but the first 4 letters were correct*", then 'abcd' are correct, 'e' is wrong, and nothing is revealed about the correctness of the letters after 'e'.

Password Example (cont.)



Example: Password policies – which one is better?!

Suppose an attacker is trying to guess/crack the password of user U1 at Company A, and user U2 at Company B. Both usernames are valid at the respective companies, and the users have chosen passwords that conform with the policy.

a) Write down an expression for the # of attempts the attacker needs for guessing the password of user U1 at Company A.

Solution:

Example: ab98CD

Total # of allowed characters = $26 + 26 + 10 = 62$

**Total # of possible passwords = $62 * 61 * 60 * 59 * 58 * 57 =$
 $= 4.4 * 10^{10}$**

Password Example (cont.)



Example: Password policies – which one is better?!

b) Write down an expression for the # of attempts the attacker needs for guessing the password of user U2 at Company B.

Solution:

The key for this part of the problem is that the attacker can use feedback provided by the login process to speed up the ‘cracking’ process.

To start, the attacker can try 100 passwords that each differ in their first character. One of these must succeed. In addition, when it succeeds, in the worst case the attacker is told that the second character in the attempted password is incorrect. Therefore, once the attacker learns that the first character is correct, they also can eliminate 1 of the possibilities for the second character.

Password Example (cont.)



Example: Password policies – which one is better?!

Password: **bszi1289AMLK**

b is correct, a is not.
In the next round, do
not check a

1st round of 100 guesses: aa, ba, ca, da, ...

2nd round of 99 guesses: bba, bca, bda, bea, ..., bsa, bta, ...

At this point, they make another $100 - 1 = 99$ guesses, each of which uses the first character learned in the previous step, and tries a different second character (excluding the character that the attacker has already learned is not correct for the second position).

This process continues until they try candidates for all 12 positions, requiring at worst a total of:

of possible passwords = $100 + 99 + 99 + \dots + 99 =$
 $= 100 + 99 \cdot 11 = 1189$