SECURITY

Passwords, Hashing, and API Keys

COS216
AVINASH SINGH
DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF PRETORIA

WEBSITE HACKS

- Company websites constantly get hacked

WEBSITE HACKS

Entity	Year	Records	Organization type	Method
Betsson Group	2020	unknown	gaming	unknown
CheckPeople	2020	56,000,000	background check	unknown
Clearview AI	2020	3 billion (Number of photos obtained)	information technology	hacked
JailCore	2020	36,000	government	poor security
Koodo Mobile	2020	unknown	mobile carrier	hacked
Marriott International	2020	5,200,000	hotel	poor security/inside job
Nintendo (Nintendo Account)	2020	160,000	gaming	hacked
SlickWraps	2020	377,428	phone accessories	poor security
Tetrad	2020	120,000,000	market analysis	poor security
Dis-Chem	2022	3,700,000	pharmacies	Unauthorised access
TransUnion	2022	3,083,227	Credit bureau	hacked

WEBSITE HACKS

- In October 2017, 60 million South African's details were leaked
- Included names, ID numbers, addresses, municipality accounts, etc
- Considered the biggest data breach in South Arica's history
- The database was not hacked, a noob web developer put a dump/backup of the database in a publicly accessible directory
- No one knows how long it has been there, but it is suspected to be years

WEBSITE HACKS

- A good guy has created a database with many hacks/leaks
- You can check if your email address or password has been leaked

https://haveibeenpwned.com

CHOOSING PASSWORDS

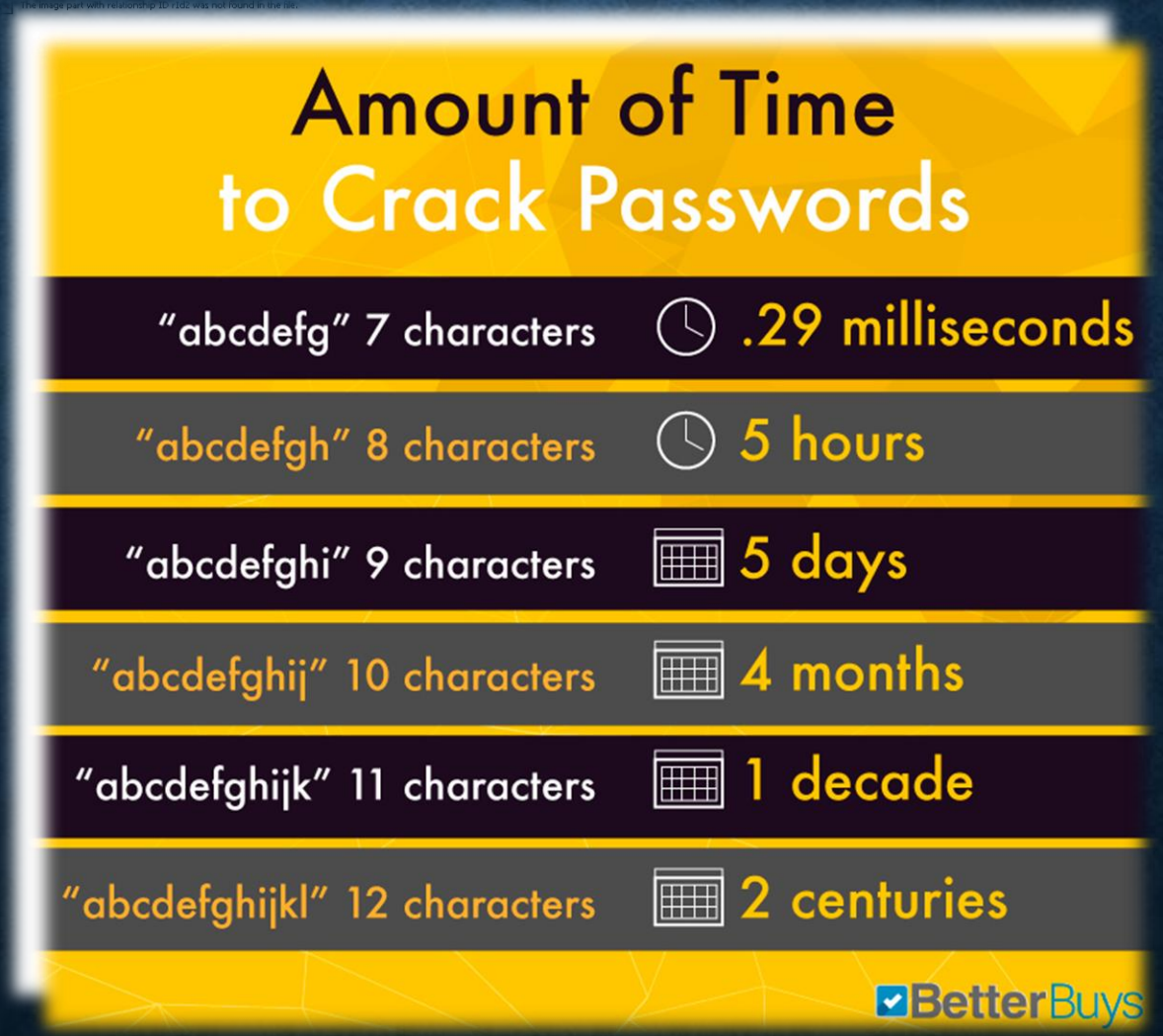
- Most people use the same password on all sites
  - If one site gets hacked, the hacker can access your accounts on other sites as well
  - Always choose different passwords for different sites, might be difficult to remember all
  - Or use variations ("MyPass#401Gmail" and "MyPass#401Twitter"), could still be guessed
- Don't use common passwords ("password123", birthdays, names, dictionary words)
- Choose long passwords from different letter-categories (upper and lower case, digits, symbols)

BRUTE FORCE

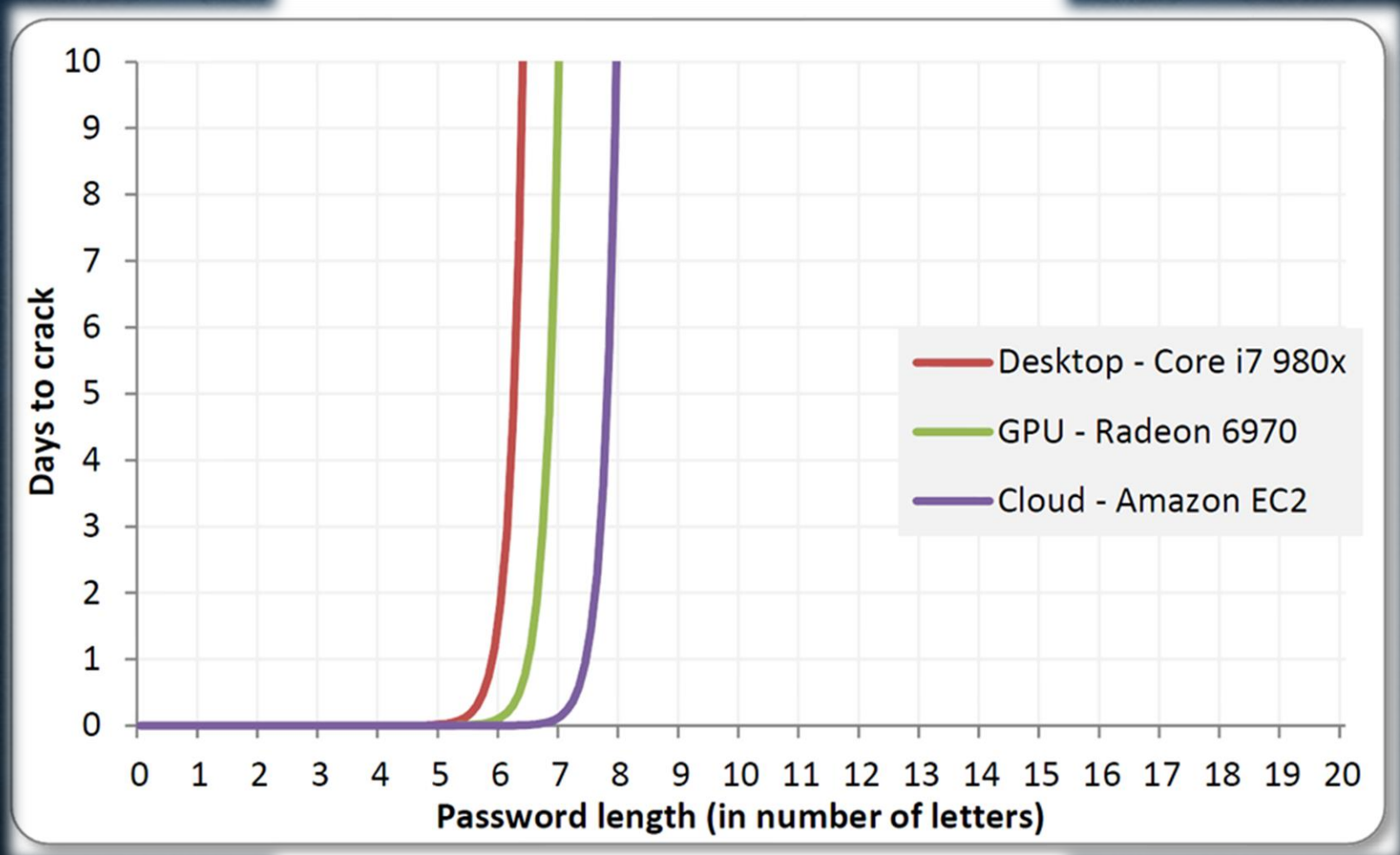
- The most common "attack" to break passwords is brute force
- Works on all algorithms, even those that are mathematically sound
- Can be written in a few lines of code
- Simply try all different letter/digit/symbol combinations

- Might take very long time to test all password
  - To reduce time, start with commonly used password, dictionaries, names, dates, etc
  - Hence, do not choose a password like "BlueClouds", "1990-01-20", "password"

BRUTE FORCE


Amount of Time
to Crack Passwords
"abcdefg" 7 characters
.29 milliseconds
"abcdefgh" 8 characters
5 hours
"abcdefghi" 9 characters
5 days
"abcdefghij" 10 characters
4 months
"abcdefghijk" 11 characters
1 decade
"abcdefghijkl" 12 characters
2 centuries
BetterBuys

BRUTE FORCE


Days to crack
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Password length (in number of letters)
Desktop - Core i7 980x
GPU - Radeon 6970
Cloud - Amazon EC2

STORING SENSITIVE INFORMATION

- Even if your website gets hacked, restrict the sensitive info that gets leaked
- Cannot really encrypt the database
  - Key has to be stored on the server for encryption and will also be leaked during a hack
  - Decreases the performance of the database, especially very large databases
  - Many SQL lookups won't work with encrypted values
- Try to obfuscate sensitive information
  - Certain fields cannot be obfuscated because they are being used (eg: email address to send emails to users)
  - However, passwords can and should be obfuscated

STORING SENSITIVE INFORMATION

- Passwords are generally not used, except for authentication
  - Hence, we only have to check if the password in the database and the password send by the login request is the same, nothing more
- If you store the password in plain text in your database, hackers have the password and can do anything with it
  - Many global companies store passwords in plain text, because their web developers are noobs
- Obfuscate passwords before storing them
  - The most common approach is hashing & salting passwords

HASHING

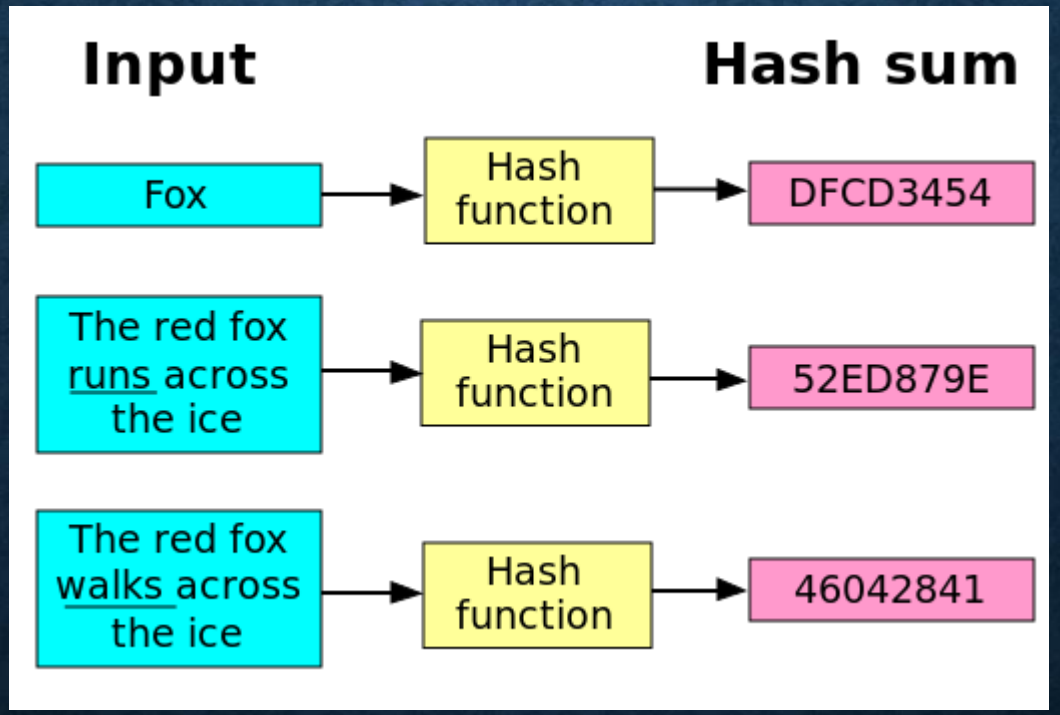
- A hash is a string of "random" characters (typically given in hexadecimal), eg:

912ec803b2ce49e4a541068d495ab570

- Since hashes are calculated by a mathematical function, they are not really random, although they might seem that way

- Hashes have various uses, such as file integrity verification, ID generation, API keys, passwords, blockchains, and various other cryptographic applications

HASHING


Input
Hash sum
Fox
Hash function
DFCD3454
The red fox runs across the ice
Hash function
52ED879E
The red fox walks across the ice
Hash function
46042841

HASHING

- Any text or file is send through a hashing function to generate the hash
  - The function is one-way, hence you cannot use a hash to get back to the original text
  - Unlike encryption, where you can encrypt and then decrypt to get the original text
- Hashes are often used as “unique” values
  - Due to the limited string length of hashes, there are a limited number of hashes
  - You can end up with 2 hashes that are the same, although the probability is very low
- If a single character/byte is changed in the original text, the new generated hash will be completely different

HASHING

- Many different hashing functions exist, each with a different algorithm and hash length
  - MD5 (128 bits): Most widely known, has been shown to be unsafe, **avoid at all costs!!!**
  - CRC (16 – 64 bits): Typically used to detect errors in files (eg: archives or downloads)
  - SHA (160 – 512 bits): Family of different functions (SHA-0, SHA-1, SHA-2, SHA-3) with various hash lengths and complexity. Most widely used hashing function at the moment. Created by the NSA. Hash collisions demonstrated in 2017
  - BLAKE (256 – 512 bits): Uses HAIFA structure
  - RIPEMD (128 – 320 bits)
  - Blowfish (32 - 448 bits): **Recommended** by PHP for password hashing
  - Many more …

HASHING

- Many different hashing functions are already implemented in PHP

```php
<?php
        // Use the hash function with the algorithm as first parameter
        echo hash('sha256', 'Just some raw text to be hashed');
        echo hash('ripemd160', 'Just some raw text to be hashed');

        // Hash an entire file
        echo hash_file('sha512', 'files/data.txt');
        echo hash_file('crc32', 'images/logo.png');

        // Common hashes have their own function
        echo md5('Just some raw text to be hashed');
        echo sha1('Just some raw text to be hashed');
?>
```

HASHING

- Hashing functions are not natively implemented in JavaScript
- You can write your own, or simply download an existing JS hash library
- Just Google “javascript <hash name here>” and look at the libraries available on GitHub

- The major ones are listed here:

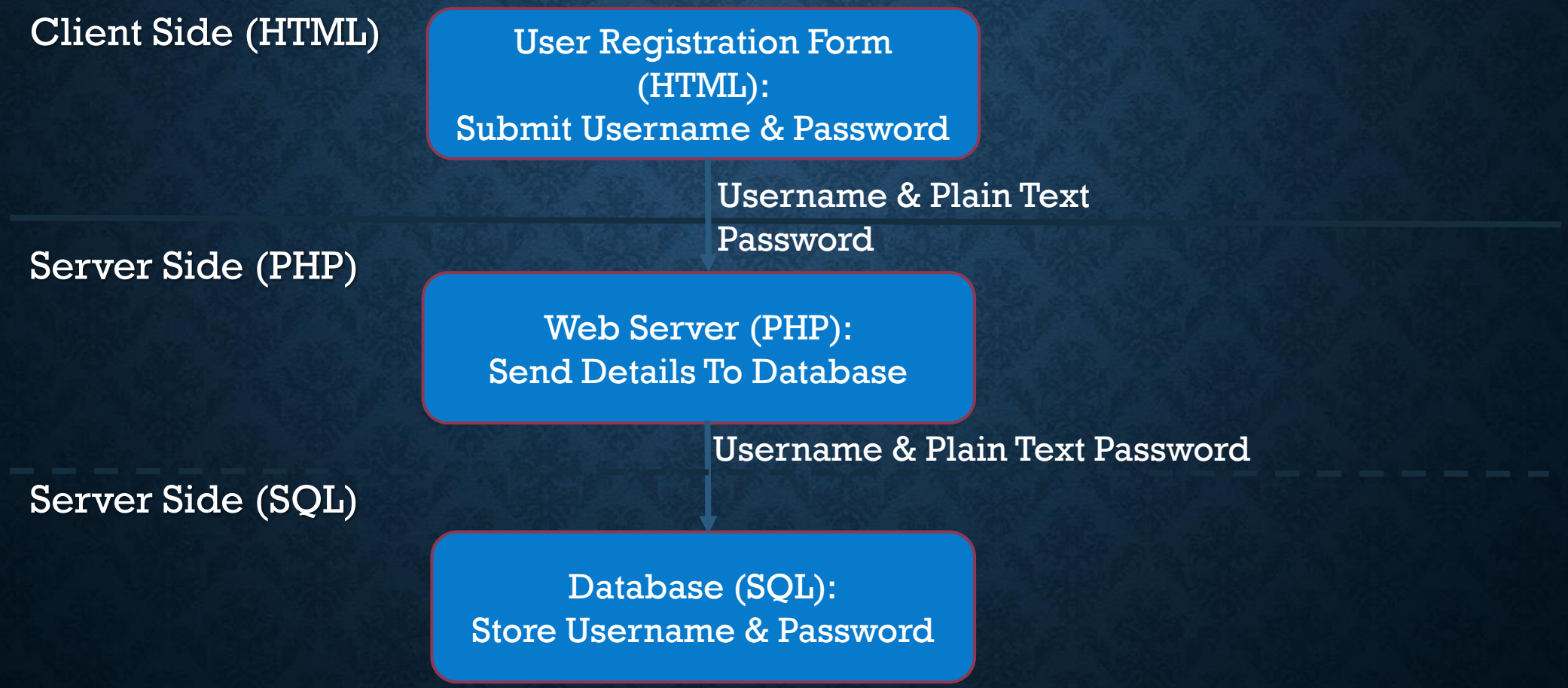
https://gist.github.com/jo/8619441

PASSWORDS – PLAIN TEXT

- Store passwords as plain text in your database
- Easy to implement, but is extremely unsafe
- If user logs in, the password is send via POST/GET and simply compared on the server to the password in the database

username	password
grandma91	icantremember
dummy_the_dummy	password123

PLAIN TEXT - REGISTRATION



PASSWORDS – HASHING

- This approach is a lot safer than plain text passwords
- However, they are still subject to "Rainbow Hash Tables"
  - These are very large pre-computed tables (GBs or TBs in size)
  - Can be freely downloaded from many places
  - Or use something like: https://crackstation.net
  - They have plain text passwords and their corresponding hash, so you search for the hash, and if found, you have the plain text password
  - Only a security issue for commonly used passwords or passwords that were previously leaked
  - If you use a very complex password that has not been leaked before, you are safe from rainbow tables

PASSWORDS – HASHING

- Example of a rainbow table (note these or not real hashes, they were truncated to fit them into the table)

original_password	md5_password	sha1_password
passy123	4a4d993ed7bd7d46…	5f4dcc3b5aa765d6…
MyComplex$Pass478	7b27af52d2aaa800…	1d8327deb882cf99…
…		
…		
…		

PASSWORDS – HASHING

- If the user registers with the plain text password, calculate the hash of the password and save only the hash in the database
- If user logs in, the password is send in plain text via POST/GET.
- On the server, calculate the hash of the login password and compare it to the hash in the database

username	password
grandma91	4a4d993ed7bd7d46
dummy_the_dummy	7b27af52d2aaa800

PASSWORDS – HASHING & SALTING

- If order to avoid the rainbow hash table problems, salt your hashes
- Salting means you add some random string to the password before hashing it
- This makes every hash different and very unlikely to appear in any rainbow table

username	password	salt
grandma91	70e5e1c938cf19ec	df564dcvb4oip
dummy_the_dummy	95c7b43570370125	zxc54as78we3

PASSWORDS – HASHING & SALTING

1. If the user registers with the plain text password, generate a random string (salt)
2. Concatenate the plain text password and the random string into one long string
3. Calculate the hash of the long string
4. Store the hash and the random string in the database
5. Each time the user logs in with the plain text password, retrieve the salt from the database, append it to the password, and calculate the hash
6. Now compare the login hash with the hash in the database

PASSWORDS

Approach	Security
Plain Text	None
Hashing	Medium
Hashing & Salting	High

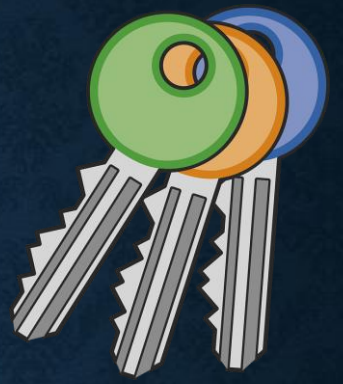
MAN IN THE MIDDLE ATTACK

- Attacks that happen in the middle
  - That is anytime between when a message leaves the client and when it arrives at the server
  - Local network, ISP's network, routing servers, etc
- If you send you password in plain text to the server
  - If the server uses hashing & salting, your password is safely stored
  - However, anyone can intercept the password on the network if you register/login
- To solve this, hash (and possibly salt) the password in JS on the website before sending it to the server
  - Hash it a second time on the server
  - Don't rely on only JS hashing, since the code is on the client and can be manipulated

API KEYS

- API keys are common practice to replace username and passwords
- They are typically a random hash or string of characters
- HTTP is stateless, if you want to keep state (eg: logged in) you can:
  - Use sessions which in turn uses cookies, subject to CSRF and cookie stealing, additionally it requires that a cookie jar (cookie functionality) is implemented on the client, which sometimes it is not (eg: mobile apps)
  - Use API keys which does not require and cookies or advanced functionality on the client
- Even if the API key gets leaked (eg: laptop is stolen), you do not have to reset your password (which you might have used on other websites as well), but simply generate a new random API key

API KEYS



- API keys typically follow this process
  1. User registers with username and password
  2. During registration, the username and salted password, as well as a randomly generated API key is stored in the database
  3. For all subsequent API calls, the API key is always appended to the requests as a parameter, the username and password is not used anymore
  4. In case the API key gets leaked, the user can simply go onto the website and generate a new key without having to change the password

- Although the key should be send via POST, if you really have to use GET, sending the API key in the URL is a lot safer than sending the user/pass

REGISTRATION VERIFICATION

- If you register on a website, you often get a verification email to verify your account:
    1. If the user registers, generate a random string (similar to API key), add the account details together with the random string to the database
    2. Also add a state to the database, saying the user is awaiting verification
    3. Also add a maximum time (eg: 1 hour) after which the random string should become unusable
    4. Send a GET URL with the random string via email to the user, eg: https://mywebsite.com/verify/DFvgt45GF3s30Df5
    5. If this URL is visited within the given time limit, change the user's status to active. If the time limit expired, email a new random string to the user, and redo step 3 and 4

SOME VIDEOS

- Choosing Passwords: https://www.youtube.com/watch?v=3NjQ9b3pgIg
- Storing Passwords: https://www.youtube.com/watch?v=8ZtInClXe1Q
- Password Cracking: https://www.youtube.com/watch?v=7U-RbOKanYs
- Hashing: https://www.youtube.com/watch?v=b4b8ktEV4Bg

NEVER STORE PASSWORDS IN PLAIN TEXT