**CASE STUDY REPORT**

The Imperva Application Defense Center (ADC) did an analysis on the strength of the passwords. Some of the key findings were:

1. About 30% of users chose passwords whose length was below seven characters,
2. Almost 60% of users chose their passwords from a limited set of alpha-numeric characters, and
3. nearly 50% of users used names, slang words, dictionary words, or trivial passwords that employed weak constructs such as consecutive digits and/or adjacent keyboard keys — case in point, the most common password among Rock Your account owners was simply “123456”.

**PASSWORD CRACKING**

In cryptanalysis and computer security, password cracking is the process of recovering passwords from data that has been stored in or transmitted by a computer system in scrambled form.

The time to crack a password is related to bit strength which is a measure of the password's entropy, and the details of how the password is stored.

Passwords that are difficult to remember will reduce the security of system

1. Users might need to write down or electronically store the password using an insecure method,
2. Users will need frequent password resets, and
3. Users are more likely to re-use the same password.

When password requirements require a long minimum length such as 16 characters, people tend to repeat characters or even entire words within their passwords. As a result, passwords may be much more easily cracked than their mathematical probabilities would otherwise indicate.

**PASSWORD SPRAYING**

An attack that attempts to access many accounts (services username) by looping a few commonly used passwords.

**ATTACKS**

Following is some of the attacks to decrypt the passwords.

1. Brute-force attack
2. Cold boot attack
3. Dictionary attack
4. Pattern checking
5. Word list Substitution

**HASHING**

Hashing consists of converting a general string of information into an intricate piece of data. This is done to scramble the data so that it completely transforms the original value, making the hashed value utterly different from the original.

Examples: MD5, SHA1, etc.

**PRIMARY USE CASES OF HASHING**

1. **Password Verification:** It is common to store user credentials of websites in a hashed format to prevent third parties from reading the passwords. Since hash functions always provide the same output for the same input, comparing password hashes is much more private.
2. **Integrity Verification:** Some files can be checked for data corruption using hash functions. Like the above scenario, hash functions will always give the same output for similar input, irrespective of iteration parameters.

**MD5**

MD5 (Message Digest Method 5) is a cryptographic hash algorithm used to generate a 128-bit digest from a string of any length. It represents the digests as 32-digit hexadecimal numbers.

A minor change in the input string generate a drastically different digest. This is essential to prevent similar hash generation as much as possible, also known as a hash collision.

**STEPS INVOLVED IN MD5**

1. Padding Bits
2. Padding Length
3. Initialize MD Buffer
4. Process each block

**ADVANATAGES**

1. **Easy to Compare:** Unlike the latest hash algorithm families, a 32-digit digest is relatively easier to compare when verifying the digests.
2. **Storing Passwords:** Passwords need not be stored in plaintext format, making them accessible for hackers and malicious actors. When using digests, the database also gets a boost since the size of all hash values will be the same.
3. **Low Resource:** A relatively low memory footprint is necessary to integrate multiple services into the same framework without a CPU overhead.
4. **Integrity Check:** You can monitor file corruption by comparing hash values before and after transit. Once the hashes match, file integrity checks are valid, and it avoids data corruption.

**DISADVANTAGES**

1. MD5 generates the same hash function for different inputs.
2. MD5 provides poor security over SHA1.
3. MD5 has been considered an insecure algorithm. So now we are using
4. SHA256 instead of MD5
5. MD5 is neither a symmetric nor asymmetric algorithm.

**POSSIBLE IMPROVEMENTS**

1. Use a dedicated password hashing algorithm bcrypt, scrypt or PBKDF2 as this will greatly increase the time needed to crack individual password.
2. Implement salting to prevent usage of rainbow tables to speed up cracking.
3. Increase the minimum password length requirement to 10 characters – this will increase the computational effort required to crack password and will give additional time to change all passwords in the event of the password database being leaked.
4. Prevent passwords to be the same as usernames or reused as part of the password – such password combination is easy to check without gaining access to the password database itself.
5. It is advised to educate users on creating safe and easy to remember passwords. Having a password policy requiring long passwords with several special characters results in user writing passwords down or constantly resetting them.
6. Educate users on the benefits of passwords managers. Having a password manager allows having very long and completely random passwords (e.g. M>?{tk6Cfep6BrZ4J)KZWQ8j) without the need to remember/write down. A strong passphrase is still required as a master key for to access the password manager.
7. One method of preventing a password from being cracked is to ensure that attackers cannot get access even to the hashed password.

For ex: on Unix OS - hashed passwords were originally stored in publicly accessible file /etc/passwd. On modern unix systems, on the other hand, they are stored in shadow password file /etc/shadow., which is accessible only to programs running with enhanced privileges.