# ***Password Cracking***

## Task 1


- *Nate Anderson* is an ‘Ars’ deputy editor, new to password cracking. He has downloaded a list of more than 16,000 cryptographically hashed passwords.
- Within a few hours, he deciphered almost half of them.
- Moral of the Story: If a reporter with zero training in the ancient art of password cracking can achieve such results, imagine what more seasoned attackers can do.

- 3 cracking experts were asked to attack the same list that *Nate Anderson* targeted. The list contained 16.449 passwords converted into hashes using the ***MD5 cryptographic hash function***.
    - The ***MD5 message-digest algorithm*** is a cryptographically broken but still widely used hash function, producing a 128-bit hash value.
    - It can be used as a checksum to verify data integrity, but only against unintentional corruption.
        - Data Integrity = The maintenance and assurance of data accuracy over its life-cycle.
- Security-conscious websites never store passwords in plaintext. They are always encrypted and then stored. They work only with ***one-way hashes,*** which are impossible to be mathematically converted back into the original sequence of characters chosen by the original user.
- In the event of a security breach that exposes the password data, an attacker must still guess the plaintext for each has
    - E.g. they must guess that "5f4dcc3b5aa765d61d8327deb882cf99" and "7c6a180b36896a0a8c02787eeafb0e4c" are the MD5 hashes for “password” and “password1” (respectively).

- The ‘Ars’ password team included a developer of cracking software, a security consultant, and an anonymous cracker. The most thorough of the 3 cracks was carried out by *Jeremi Gosney*, a password expert with ‘Stricture Consulting Group’.

- Using a commodity computer with a single AMD Radeon 7970 graphics card, it took him 20 hours to crack 14,734 of the hashes, a 90% success rate.
    - Commodity = An economic good/resource, that has a full/substantial fungibility, that is if the market treated the good fairly with no regard to who produced them.
        - Fungibility = The ability of a good/asset to be interchanged for another good of the same kind.
    - Commodity Computing = The use of large numbers of available computing components for parallel computing, to get the greatest amount of useful computation for a low cost.
        - Parallel Computing = A type of computation in which many calculations/processes are carried out simultaneously.
- *Steube* unscrambled 13,486 hashes (82% success rate) in a little more than an hour, using a slightly more powerful machine that contained 2 AMD Radeon 6990 graphics cards.
- A 3rd cracker (*Moniker radix*) deciphered 62% of the hashes using a computer with a single 7970 card in about 1 hour.

- Like SHA1, SHA3 and most other algorithms, MD5 was designed to convert plaintext into hashes, also known as 'message digests', quickly and with a minimal amount of computation. That works in the favor of crackers.
- Armed with a single graphics processor, they can cycle through more than 8 billion password combinations each second when attacking "fast" hashes. By contrast, algorithms specifically designed to protect passwords need significantly more time and computation.

## ***Salt Cryptography***

- In cryptography, a salt is a random data that is used as an additional input to a one-way function that hashes data, a password/passphrase.
    - Salts are used to safeguard passwords in storage.
- A new salt is randomly generated for each password.

## Example usage [edit]

Here is an incomplete example of a salt value for storing passwords. This first table has two username and password combinations. The password is not stored.

| Username | Password |
|---|---|
| user1 | password123 |
| user2 | password123 |

The salt value is generated at random and can be any length; in this case the salt value is 8 bytes long. The salt value is appended to the plaintext password and then the result is hashed, which is referred to as the hashed value. Both the salt value and hashed value are stored.

| Username | Salt value | String to be hashed | Hashed value = SHA256 (Password + Salt value) |
|---|---|---|---|
| user1 | E1F53135E559C253 | **password123**E1F53135E559C253 | 72AE25495A7981C40622D49F9A52E4F1565C90F048F59027BD9C8C8900D5C3D8 |
| user2 | 84B03D034B409D4E | **password123**84B03D034B409D4E | B4B6603ABC670967E99C7E7F1389E40CD16E78AD38EB1468EC2AA1E62B8BED3A |

As the table above illustrates, different salt values will create completely different hashed values, even when the plaintext passwords are exactly the same. Additionally, dictionary attacks are mitigated to a degree as an attacker cannot practically precompute the hashes. However, a salt cannot protect common or easily guessed passwords.

## ***Cryptographic Hash Function***

- Cryptographics hash function is a mathematical algorithm that maps data of arbitrary size to a bit array of a fixed size.

# Task 2

Links used to decrypt MD5 codes:

- https://md5decrypt.net/en/
- https://www.dcode.fr/md5-hash

- Encrypted 1 - e10adc3949ba59abbe56e057f20f883e
  - Decoded 1 - 123456
- Encrypted 2 - 25f9e794323b453885f5181f1b624d0b
  - Decoded 2 - 123456789
- Encrypted 3 - d8578edf8458ce06fbc5bb76a58c5ca4
  - Decoded 3 - qwerty
- Encrypted 4 - 5f4dcc3b5aa765d61d8327deb882cf99
  - Decoded 4 - password
- Encrypted 5 - 96e79218965eb72c92a549dd5a330112
  - Decoded 5 - 111111
- Encrypted 6 - 25d55ad283aa400af464c76d713c07ad
  - Decoded 6 - 12345678
- Encrypted 7 - e99a18c428cb38d5f260853678922e03
  - Decoded 7 - abc123
- Encrypted 8 - fcea920f7412b5da7be0cf42b8c93759
  - Decoded 8 - 1234567

- Encrypted 9 - 7c6a180b36896a0a8c02787eeafb0e4c
    - Decoded 9 - password1
- Encrypted 10 - 6c569aabbf7775ef8fc570e228c16b98
    - Decoded 10 - password!
- Encrypted 11 - 3f230640b78d7e71ac5514e57935eb69
    - Decoded 11 - qazxsw
- Encrypted 12 - 917eb5e9d6d6bca820922a0c6f7cc28b
    - Decoded 12 - Pa$$word1
- Encrypted 13 - f6a0cb102c62879d397b12b62c092c06
    - Decoded 13 - bluered
- Encrypted 14 - 9b3b269ad0a208090309f091b3aba9db
    - Decoded 14 - Flamesbria2001
- Encrypted 15 - 16ced47d3fc931483e24933665cded6d
    - Decoded 15 - Oranolio1994
- Encrypted 16 - 1f5c5683982d7c3814d4d9e6d749b21e
    - Decoded 16 - Spuffyffet12
- Encrypted 17 - 8d763385e0476ae208f21bc63956f748
    - Decoded 17 - moodie00
- Encrypted 18 - defebde7b6ab6f24d5824682a16c3ae4
    - Decoded 18 - nAbox!1
- Encrypted 19 - bdda5f03128bcbdfa78d8934529048cf
    - Decoded 19 - Banda11s

# Task 3

1) What type of hashing algorithm was used to protect passwords?
   The MD5 cryptographic hash function.
2) What level of protection does the mechanism offer for passwords?

> Why is MD5 not secure?
>
> **MD5 is a cryptographic algorithm, often used to store passwords in a database**
>
> **But this algorithm is no longer safe**
>
> **Brute force attacks are faster than ever, dictionary tables are big and there are other potential problems with the MD5 algorithm**

3) What controls could be implemented to make cracking much harder for the hacker in the event of a password database leaking again?

## Making an Md5 Hash More Secure

To make the md5 hash more secure we need to add what is called "salt". Salt in this sense of the meaning is random data appended to the password to make the hash more complicated and difficult to reverse engineer. Without knowing what the salt is, rainbow table attacks are mostly useless.

```php
<?php
$password = 'mypassword';
$salt='i_always_take_a_sentence_or_two_and_add_some_numbers_8342394';
$saltedHash = md5($pass . $salt);
echo $saltedHash;
?>
```

Now obviously if an attacker figures out what salt you use the entire hash system is flawed. So keep your salt safe.

As Jade suggested in his comment below, you can decrease the chance of someone reversing an md5 hash by simply having a more complex password. See, Picking Strong Passwords that you can Remember for more info on picking a complex password that is easy to remember.

4) What can you tell about the organization's password policy (e.g. password length, key space, etc.)?

Most of their passwords are significantly insecure as they are too common. Some have a more heightened sense of security compared to the first few. Regardless, this means that the company's password policy is terrible, and is in need of an update.

5) What would you change in the password policy to make breaking the passwords harder?

I would add a formal criteria of what a password should be. Any password that does not include these requirements is insecure:

### Characteristics of strong passwords

- At least 8 characters—the more characters, the better
- A mixture of both uppercase and lowercase letters
- A mixture of letters and numbers
- Inclusion of at least one special character, e.g., ! @ # ? ]
  **Note:** do not use < or > in your password, as both can cause problems in Web browsers