Sushma Shivshankar Nandiyawar

(sushnand@iu.du)

Lab 6 : Cracking Passwords

Qusetion 4: Password Cracking

Q 4.1:

Answer: **ozzy**

```
root@sushnand:~# john sushnand_1
Created directory: /root/.john
Loaded 1 password hash (md5crypt [MD5 32/64 X2])
Press 'q' or Ctrl-C to abort, almost any other key for status
ozzy             (sushnand)
1g 0:00:00:23 100% 2/3 0.04306g/s 17880p/s 17880c/s 17880C/s ozzy..packrat
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

Q 4.2:

Answer: **apples85**

The rule used is : $[0-9]$[0-9]

```
root@sushnand:~# john sushnand_2
Loaded 1 password hash (md5crypt [MD5 32/64 X2])
Press 'q' or Ctrl-C to abort, almost any other key for status
apples85         (sushnand)
1g 0:00:00:16 100% 2/3 0.05920g/s 17910p/s 17910c/s 17910C/s apples85..barbara85
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

Q 4.3:

Answer: **be22lize21**

The rule used is: i2[1-2]i3[1-2]$[1-2]$[1-2]

```
root@sushnand:~# john sushnand_3
Loaded 1 password hash (md5crypt [MD5 32/64 X2])
Press 'q' or Ctrl-C to abort, almost any other key for status
be22lize21       (sushnand)
1g 0:00:00:22 100% 2/3 0.04411g/s 17919p/s 17919c/s 17919C/s be22lize21..be22lle21
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

Q 4.3:

Answer: **pragmatieker**

```
root@sushnand:~# john --wordlist=lower sushnand_4
Loaded 1 password hash (md5crypt [MD5 32/64 X2])
Press 'q' or Ctrl-C to abort, almost any other key for status
pragmatieker     (sushnand)
1g 0:00:00:06 100% 0.1497g/s 17890p/s 17890c/s 17890C/s pragmatieke..pragmatieker
Use the "--show" option to display all of the cracked passwords reliably
Session completed
root@sushnand:~#
```

Question 5:

Q 5.1: Explain how a "white hat" security professional might use john-the-ripper to

make her institution more secure.

Answer :

1. John the Ripper, as an open-source password cracking software, is valuable for ethical hackers in assessing password security.
2. It employs various cracking algorithms and techniques across different operating systems.
3. Once weak passwords are identified, the tool enables the implementation of rules to enforce stronger password practices.
4. A white hat security expert can utilize John the Ripper, an open-source password cracking tool, to:
    a. Evaluate password strength and pinpoint weak passwords.
    b. Simulate brute-force and dictionary attacks to assess password resilience.
    c. Examine password hashing methods for potential vulnerabilities.
    d. Generate audit trails for analyzing patterns and identifying potential threats.
    e. Educate users on adopting strong password practices.
    f. Implement and update password policies based on assessment findings.
    g. Monitor and adjust security measures continually to ensure ongoing protection.

Q 5.2: Why can john-the-ripper crack the passwords even though they are not in a form that is directly readable?

Answer:

1. John the Ripper employs a combination of techniques to crack passwords, utilizing both brute-force and dictionary attacks.
2. Initially, it adopts a single crack approach, a swift and efficient method effective against weak passwords.
3. Should this fail, it switches to a brute force attack, systematically trying every conceivable character combination.
4. While this process can be slow, it proves effective against strong passwords.
5. Moreover, John the Ripper supports the use of rainbow tables, which are precomputed tables containing hash values for common passwords.
6. The tool's ability to crack passwords, even when they are not in a directly readable form (such as hashed or encrypted), stems from its diverse strategies.

7. These include dictionary attacks, systematically testing words from predefined lists, and brute-force attacks, attempting every possible password combination.
8. Advanced algorithms further exploit weaknesses in password hashing methods, enabling the decryption of hashed or encrypted passwords.
9. This capability highlights the importance of implementing robust password policies and secure hashing methods to bolster overall system security.

Q 5.3: Explain how John the Ripper limits/reduces password cracking time?

Answer:

1. John the Ripper employs multiple strategies in its password cracking approach.
2. Initially, it attempts a single crack by using user-specific information like usernames, email addresses, or dates of birth.
3. Simultaneously, it checks against a list of common passwords from a rainbow table.
4. If these methods fail, John the Ripper employs a mangling technique, rapidly trying variations like adding numbers, special characters, or altering case to enhance the efficiency of the password cracking process.
5. Overall, it minimizes cracking time through optimized algorithms, parallel processing, GPU acceleration, and the systematic use of wordlists and rules, tailoring its approach to diverse hashing algorithms.

Q 5.4: What system policies for passwords would make user passwords considerably harder to crack by a password cracker such as john-the- ripper? What are the downsides of enforcing such policies?

Answer:

1. Implementing robust system password policies significantly improves security and poses challenges for password-cracking tools like John the Ripper. Here are effective policies:
    a. **Password Length Requirements:** Mandate a minimum password length, prompting users to opt for longer, more intricate passwords. Extended passwords generally heighten the difficulty of brute-force attacks.
    b. **Complexity Requirements:** Demand a mix of uppercase and lowercase letters, numbers, and special characters. This fortifies passwords against dictionary attacks, expanding the search space for password crackers.
    c. **Password Expiry:** Institute a policy requiring regular password changes. This minimizes the time compromised passwords are exposed, reducing the likelihood of unauthorized access.
    d. **Account Lockout Policies:** Deploy mechanisms locking accounts after a set number of failed login attempts. This obstructs brute-force attacks by slowing down the password-cracking process.
    e. **Password History:** Prohibit users from recycling previous passwords, ensuring they choose new and unique passwords for each change.
    f. **Two-Factor Authentication (2FA):** Advocate or enforce the use of two-factor authentication, introducing an extra layer of security beyond passwords. Even if a password is compromised, an additional authentication step is necessary.

2. **Downsides of Enforcing Strict Password Policies:**
    a. User Frustration: Complex password requirements may frustrate users, potentially leading to weaker password choices or writing them down.
    b. Forgotten Passwords: Frequent password changes may increase the likelihood of users forgetting passwords, resulting in more frequent requests for resets.
    c. Resistance to Change: Users might resist frequent changes or complexity requirements, leading to a lack of cooperation and potential security vulnerabilities.
    d. Increased Support Overhead: Strict policies may result in more support requests related to forgotten passwords, account lockouts, and other issues.
    e. Security Theater: Overly stringent policies may foster a false sense of security if users find ways to circumvent requirements, such as using easily guessable variations or writing down passwords.

3. Finding the right balance between strong security practices and user convenience is crucial. Achieving this balance enhances security without imposing unnecessary obstacles on users. Regular education and communication about the importance of strong passwords, along with explanations of policy choices, can help alleviate some of the downsides.

Q 5.5: How much does a salt of size N increase the processing required by precomputed dictionary offline attacks?

Answer:

1. As the size of the salt (denoted as N) increases, the time required for precomputation experiences an exponential growth of 2^N.
2. This is due to the doubling of necessary storage and computation for salts with each incremental growth in salt size.
3. The use of a salt in password hashing is specifically designed to thwart precomputed dictionary attacks, such as those involving rainbow tables.
4. A salt, a random value unique to each user, is combined with the password before hashing.
5. This ensures that even if two users have the same password, their hashed values will differ due to the unique salt.
6. The inclusion of a salt significantly increases the processing required by precomputed dictionary attacks.
7. Without a salt, attackers can create a precomputed table (rainbow table) containing the hashes of common passwords, allowing them to quickly look up the hash and determine the corresponding password.
8. However, with salts in use, each password must be hashed with its unique salt.
9. This means that attackers would need to create a separate precomputed table for each potential salt value, substantially increasing the computational effort required.
10. The size of the salt (N) adds complexity by expanding the number of potential salt values.
11. In essence, the use of salts makes precomputed dictionary attacks more challenging and resource-intensive for attackers, as they must generate tables for every possible salt, greatly reducing the efficiency of such attacks.

12. The larger the size of the salt (N), the more computational resources are needed to carry out precomputed dictionary attacks successfully.

Ref:

https://stackoverflow.com/questions/3566504/why-do-salts-make-dictionary-attacks-impossible

https://cybernews.com/best-password-managers/password-cracking-techniques/

https://ro.ecu.edu.au/cgi/viewcontent.cgi?referer=&httpsredir=1&article=1144&context=theses_hons#:~:text=John%20the%20Ripper%20(JtR)%20attempts,of%20the%20password%2Dcracking%20task.

https://web.njit.edu/~rxt1077/security/