Password Cracking & Cipher Challenges

Rashed Al Fuhed
rashalyami79@gmail.com
Advisor: Dr. Mohammad Al Hasan
03/29/2024

Abstract

This study investigates the potency of password recovery using Hashcat against MD5 and SHA1 hashes and explores the subsequent re-encryption of compromised passwords with the classical Caesar and Vigenère ciphers. Detailed experimentation reveals significant vulnerabilities in MD5 and SHA1—outdated yet persistently used hash functions. The project's success in cracking numerous passwords showcases these weaknesses and assesses the protective strength of the classical ciphers in the face of contemporary cryptographic challenges. Furthermore, the research extends into the automation of encryption and decryption processes, employing advanced Python scripting to handle complex cipher operations. The findings emphasize the critical need for progressive cryptographic protocols and contribute meaningful discourse to the cybersecurity community. Ultimately, this report calls for a reevaluation of current security measures and promotes the advancement of encryption standards to counteract the evolving landscape of cyber threats.

Keywords: Python, Cybersecurity, Password Cracking, Ciphers, Hash, Hashing Algorithms, Hashcat, Caesar, Vigenère

1. Introduction

In the digital age, the security of information systems is very important, yet it remains under constant threat from various types of cyber attacks. Among these, the compromise of password-secured systems is a significant vulnerability. Passwords are often protected by cryptographic or salted hashes, which should theoretically obfuscate the original password in a manner that is computationally impractical to reverse. However, older hashing algorithms such as MD5 and SHA1 have been proven vulnerable to various attacks, making them a point of concern for cybersecurity professionals. This project addresses the vulnerabilities associated with these outdated cryptographic hashes by employing modern password cracking tools, specifically focusing on Hashcat's capabilities to break passwords protected by MD5 and SHA1 hashes. Despite their known weaknesses, these hash functions continue to be used in many legacy systems, presenting an ongoing security risk. Further, the study explores the application of classical encryption methods—Caesar and Vigenere ciphers—as a means of encrypting the cracked passwords.

Although these ciphers are from much earlier cryptographic eras and are not secure by today's standards, examining their efficacy in a controlled setting provides valuable insights into basic cryptographic principles and their resilience against low-level attacks. Through this dual approach, this research not only demonstrates the ease with which certain hashes can be cracked but also evaluates the subsequent encryption of data with classical methods. This comprehensive examination helps highlight the advancements in cryptographic techniques and underscores the need for continuous improvement in encryption practices to safeguard sensitive information against evolving cyber threats. This project aims to contribute to the broader discourse on cybersecurity by bridging the gap between outdated cryptographic practices and modern requirements for secure digital communications. By understanding the limitations of past and present technologies, we can better prepare for the security challenges of the future.

2. Problem Definition

The digital age has ushered in an era where data security is crucial, with cryptographic hash functions playing a pivotal role in the protection of information. However, the integrity of cryptographic systems is under constant threat due to the advancement of password cracking techniques. This project specifically addresses the vulnerabilities associated with two such hash functions: MD5 and SHA1.

2.1. MD5 Vulnerabilities:

MD5 has been widely used for password hashing due to its speed and simplicity. However, it is fundamentally flawed in today's security landscape. The hash function is susceptible to collision attacks, where two different inputs produce the same output hash, compromising data integrity and allowing malicious actors to deceive systems by substituting a legitimate hash with a fraudulent one. Additionally, MD5's susceptibility to brute-force and rainbow table attacks further diminishes its reliability as a secure cryptographic solution.

2.2. SHA1 Vulnerabilities:

SHA1, similarly, has shown weaknesses despite being a step up from MD5 in terms of complexity and supposed security. Like MD5, SHA1 is vulnerable to collision attacks, with practical collisions now being demonstrated, signaling its unsuitability for continued cryptographic use. The computational power available today allows attackers to break SHA1 with a feasible amount of resources and time, making it an insecure choice for safeguarding sensitive data.

Despite these vulnerabilities, both MD5 and SHA1 are still in use in various applications, from legacy systems that have not been updated to current security standards, to active systems where updates might disrupt service or functionality. This continued use in the face of known weaknesses presents a significant risk to data security. The encryption phase of this project, involving the Caesar and Vigenere

ciphers, addresses another dimension of the problem: the security of classical encryption methods in contemporary contexts. These ciphers, though historically significant, offer minimal security against modern decryption techniques but are used here to illustrate fundamental cryptographic principles and their limitations. This project aims to expose the specific vulnerabilities of MD5 and SHA1 through practical cracking experiments and to discuss the efficacy of classical encryption methods when applied to data compromised from insecure hashing. The goal is to advocate for stronger, more secure cryptographic practices and to inform about the necessity of updating and securing cryptographic implementations in various technological infrastructures.

3. Solution

This project employs a two-pronged approach to addressing the vulnerabilities in MD5 and SHA1 hashes, followed by the application of classical encryption techniques to provide a layer of security to the compromised data. Each aspect of the solution is detailed below:

3.1. Password Cracking with Hashcat:

- **Tool Selection**: Hashcat was chosen for its robust performance and versatility in handling different types of hash functions. It is renowned for its ability to leverage the power of both CPUs and GPUs, making it highly effective for password cracking tasks.
- **MD5 and SHA1 Cracking**: Specific techniques and strategies were utilized to crack these hashes:
    - **Brute Force & Mask Attacks**: This method was employed to test the resilience of both hash functions against exhaustive key searches. Hashcat's capability to perform rapid brute-force attacks allowed for an extensive testing of password strength within feasible time frames.
    - **Dictionary Attacks**: Leveraging a comprehensive dictionary of commonly used and breached passwords, this approach tested the hashes against a realistic set of potential passwords, reflecting common user behaviors and password creation patterns.

3.2. Encryption with Caesar and Vigenere Ciphers:

- **Implementation Details**: Once passwords were successfully cracked, they were encrypted using Caesar and Vigenere ciphers. The implementation was conducted as follows:
    - **Caesar Cipher**: A simple shift cipher that replaces each plaintext letter with a letter a fixed number of positions down the alphabet. The shift value was varied across experiments to analyze the impact on cipher robustness.
    - **Vigenère Cipher**: This method uses a keyword to generate a repeat pattern that shifts plaintext letters based on the corresponding letter in the keyword, offering a more complex encryption than the Caesar cipher.
- **Security Analysis**: The security of each cipher was evaluated by attempting to decrypt the encrypted passwords without access to the key, using statistical analysis and pattern recognition techniques to assess potential vulnerabilities.

4. Implementation/Code

This section is for how we integrated the ideas with Python. The following code snippets are as follows:

```python
def caesar_cipher(text, shift, direction='encrypt'):
    result = ""
    for char in text:
        if char.isalpha():
            start = ord('a') if char.islower() else ord('A')
            shift_amount = shift if direction == 'encrypt' else -shift
            shifted_char = chr((ord(char) - start + shift_amount) % 26 + start)
            result += shifted_char
        elif char.isdigit():
            start = ord('0')
            shift_amount = shift if direction == 'encrypt' else -shift
            shifted_char = chr((ord(char) - start + shift_amount) % 10 + start)
            result += shifted_char
        else:
            result += char
    return result
```

The caesar_cipher function implements the Caesar cipher encryption technique. It takes a string of text, a shift value, and an optional direction ('encrypt' or 'decrypt') and returns the encrypted or decrypted text accordingly. It processes each character in the input text, checking if it's a letter or a digit, applying the cipher accordingly, and preserving the case.

```python
def vigenere_cipher(text, key, direction='encrypt'):
    result = ""
    key_length = len(key)
    key_as_int = [ord(i) - ord('a') if i.isalpha() else ord(i) - ord('0') for i in key.lower() if i.isalnum()]  # Extend
 key to include digits
    text_int = [ord(i) for i in text]
    for i in range(len(text_int)):
        if text[i].isalpha():
            start = ord('a') if text[i].islower() else ord('A')
            shift = key_as_int[i % key_length]
            if direction == 'encrypt':
                offset = (text_int[i] - start + shift) % 26
            else:
                offset = (text_int[i] - start - shift) % 26
            result += chr(offset + start)
        elif text[i].isdigit():
            start = ord('0')
            shift = key_as_int[i % key_length] % 10  # Normalize shift for numbers
            if direction == 'encrypt':
                offset = (text_int[i] - start + shift) % 10
            else:
                offset = (text_int[i] - start - shift) % 10
            result += chr(offset + start)
        else:
            result += text[i]
    return result
```

The vigenere_cipher function performs encryption or decryption using the Vigenere cipher, which is more complex than the Caesar cipher. It takes additional parameters, including a key, and handles alphabetic and numeric characters by applying the cipher to shift each character based on the corresponding character in the key.

```python
def process_passwords(file_path, shift, vigenere_key):
    with open(file_path, "r") as file:
        lines = file.readlines()

    passwords = [line.strip().split(':')[1] for line in lines if ':' in line]

    caesar_encrypted = [caesar_cipher(password, shift, 'encrypt') for password in passwords]
    caesar_decrypted = [caesar_cipher(password, shift, 'decrypt') for password in caesar_encrypted]

    vigenere_encrypted = [vigenere_cipher(password, vigenere_key, 'encrypt') for password in passwords]
    vigenere_decrypted = [vigenere_cipher(password, vigenere_key, 'decrypt') for password in vigenere_encrypted]

    return passwords, caesar_encrypted, caesar_decrypted, vigenere_encrypted, vigenere_decrypted
```

The process_passwords function in Python is created to automate the encryption and decryption of passwords from a file. Initially, the function reads passwords from a specified file, assuming each password is located after a colon on each line. These passwords are then encrypted and decrypted using both the Caesar and Vigenere ciphers, according to the provided shift value and Vigenere key. The function outputs a comprehensive collection of the original passwords, their encrypted and decrypted forms using Caesar cipher, and similarly, their encrypted and decrypted versions using the Vigenere cipher. This allows for a streamlined process of demonstrating the application of these classical cryptographic methods on a set of data.

```
def delete_file_if_exists(file_path):
    """Delete file if it exists."""
    if os.path.exists(file_path):
        os.remove(file_path)
        print(f"Deleted existing file: {file_path}")

def run_hashcat(hash_file, wordlist_file, hash_type, output_file, log_file):
    delete_file_if_exists(output_file) # Ensure the output file is deleted before running Hashcat
    try:
        hashcat_cmd = f"hashcat -m {hash_type} -a 0 -o {output_file} {hash_file} {wordlist_file} --force -n 2"
        with open(log_file, "w") as log:
            subprocess.run(hashcat_cmd, shell=True, check=True, stdout=log, stderr=subprocess.STDOUT)
        print("Hashcat completed successfully. Check the log for details.")
        return True
    except subprocess.CalledProcessError as e:
        print("Hashcat failed with error:", e)
        return False
```

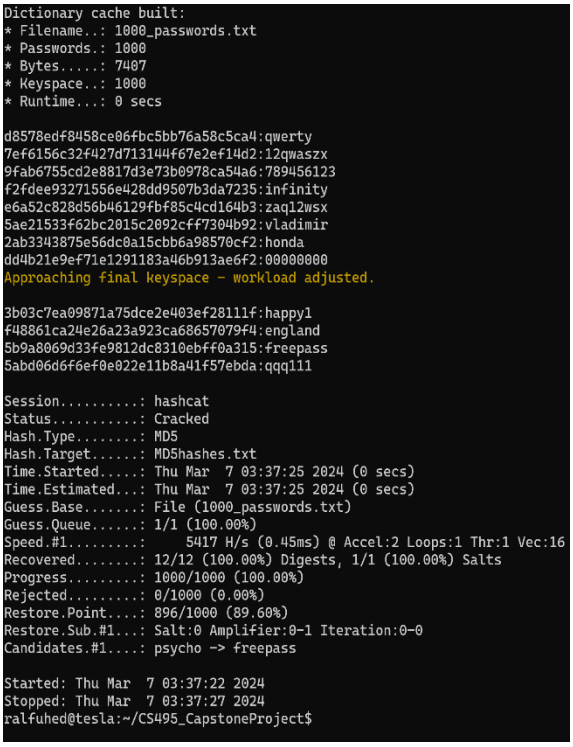
The last snippet defines two functions related to file handling and running Hashcat commands. The delete_file_if_exists function is a utility to remove a file if it already exists, preventing potential write conflicts. The run_hashcat function is a higher-level function designed to construct and execute a Hashcat command. It utilizes a subprocess to run the Hashcat command with given parameters, captures the output to a log file, and handles any errors that occur during the execution.

Together, these functions represent a suite of tools for password security analysis, allowing for both classical encryption techniques and modern hash-cracking operations. They illustrate a practical approach to understanding and applying cryptographic concepts and highlight the integration of subprocess management for using external tools like Hashcat within a Python script.

5. Experimental Evaluation

5.1. MD5 Hashes

- **Dictionary Attack**: This attack was done with the following command in Hashcat:
    - hashcat -m 0 -a 0 {hashes_file} {wordlist} --force -n 2
    - “-m” is for the hashing algorithm, “-a” is for the attack mode, “–force” forces hashcat to start the attack and ignores any warnings, and “-n 2” sets the number of GPU threads per computer to 2 (This one is completely optional but recommended).



- **Brute Force and Mask Attack:** This attack was done with hashcat using the following command:
    - hashcat -m 0 -a 3 -o {output file} {hashes file} -1 aeiouAEIOU -2 bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ ?1?2?2?d?1?1 --force -n 2
    - “-1” and “-2” are custom sets to make it easier for us to crack the passwords faster. “-1” is for vowels, and “-2” is for consonants.
    - “?1?2?2?d?1?1” This is the pattern for the brute-force attack. It uses the custom character sets where “?1” is for vowels (custom set 1), “?2” is for consonants (custom set 2), and “?d” is for digits

```
Session..........: hashcat
Status...........: Cracked
Hash.Type........: MD5
Hash.Target......: MD5hashes.txt
Time.Started.....: Thu Mar  7 03:56:36 2024 (4 secs)
Time.Estimated...: Thu Mar  7 03:56:40 2024 (0 secs)
Guess.Mask.......: ?1?2?2?d?1?1 [6]
Guess.Charset....: -1 aeiouAEIOU, -2 bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ, -3 Undefined, -4 Undefined
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  1118.6 kH/s (0.44ms) @ Accel:2 Loops:13 Thr:1 Vec:16
Recovered........: 13/13 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.........: 4030432/17640000 (22.85%)
Rejected.........: 0/4030432 (0.00%)
Restore.Point....: 9520/42000 (22.67%)
Restore.Sub.#1...: Salt:0 Amplifier:273-286 Iteration:0-13
Candidates.#1....: oDB7io -> ukk7io

Started: Thu Mar  7 03:56:26 2024
Stopped: Thu Mar  7 03:56:41 2024
ralfuhed@tesla:~/CS495_CapstoneProject$ ls
1000_passwords.txt  MD5hashes.txt  result.txt  SHA1Hashes.txt
ralfuhed@tesla:~/CS495_CapstoneProject$ cat result.txt
d5e16706ecd8a736cc7e763c6a47d5f3:Agb9io
```

5.2. SHA1 Hashes

- **Dictionary Attack**: This was also done using hashcat with the following command:
    - hashcat -m 100 -a 0 {hashes_file} {wordlist} --force -n 2
    - Same command as MD5's, the only difference is the "-m" and hashes file

```
b1b3773a05c0ed0176787a4f1574ff0075f7521e:qwerty
4b4b04529d87b5c318702bc1d7689f70b15ef4fc:789456123
aebc3ebee2f0c8b08b43d26c2b0055b19caeaf4a:12qwaszx
0c4c6b12888e68a0828006f4e252af0b387cc357:infinity
cdf547ed4c64e6994af35cfcd69c4204c9227a97:zaq12wsx
dc9186a06078733915a6fcbab34e59120be2b484:vladimir
70352f41061eda4ff3c322094af068ba70c3b38b:00000000
de3d5bd1e1b72410a8786678ee4408d6a9cf7061:honda
Approaching final keyspace - workload adjusted.

1461b0d8355715b741f294780f7721b0f16f4094:happy1
1a84cea55deef3ba2367609768375cb79d50df5c:qqq111
b6ce68526de3e64f062e958666d9e8d5766b37e3:england
206fcb206c16c939510412c5ce5bedac33b45b75:freepass

Session..........: hashcat
Status...........: Cracked
Hash.Type........: SHA1
Hash.Target......: SHA1Hashes.txt
Time.Started.....: Thu Mar  7 03:42:06 2024 (0 secs)
Time.Estimated...: Thu Mar  7 03:42:06 2024 (0 secs)
Guess.Base.......: File (1000_passwords.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:     5279 H/s (0.44ms) @ Accel:2 Loops:1 Thr:1 Vec:16
Recovered........: 12/12 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.........: 1000/1000 (100.00%)
Rejected.........: 0/1000 (0.00%)
Restore.Point....: 896/1000 (89.60%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-0
Candidates.#1....: psycho -> freepass

Started: Thu Mar  7 03:41:54 2024
Stopped: Thu Mar  7 03:42:08 2024
ralfuhed@tesla:~/CS495_CapstoneProject$
```

- **Brute Force & Rule-based Attack**
    - hashcat -m 100 -a 3 -o {output file} {hashes file} -1 aeiouAEIOU -2 bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ ?1?2?2?d?1?1 --force -n 2
    - This is also the same command is MD5's, but different "-m" and hashes file

```
Session..........: hashcat
Status...........: Cracked
Hash.Type........: SHA1
Hash.Target......: SHA1Hashes.txt
Time.Started.....: Thu Mar  7 04:00:26 2024 (4 secs)
Time.Estimated...: Thu Mar  7 04:00:30 2024 (0 secs)
Guess.Mask.......: ?1?2?2?d?1?1 [6]
Guess.Charset....: -1 aeiouAEIOU, -2 bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ, -3 Undefined, -4 Undefined
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  1115.5 kH/s (0.41ms) @ Accel:2 Loops:13 Thr:1 Vec:16
Recovered........: 13/13 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.........: 4030432/17640000 (22.85%)
Rejected.........: 0/4030432 (0.00%)
Restore.Point....: 9520/42000 (22.67%)
Restore.Sub.#1...: Salt:0 Amplifier:273-286 Iteration:0-13
Candidates.#1....: oDB7io -> uKk7io

Started: Thu Mar  7 04:00:16 2024
Stopped: Thu Mar  7 04:00:32 2024
ralfuhed@tesla:~/CS495_CapstoneProject$ cat result.txt
e178372e5cb9cadfa79dcc69bf5059dd4893060e:Agb9io
ralfuhed@tesla:~/CS495_CapstoneProject$
```

5.3. Evaluation

Upon analyzing the password cracking process for MD5 and SHA1 hash algorithms using Hashcat, several performance metrics were observed. The hash rates recorded for MD5 varied significantly, with one instance showing 5,417 hashes per second (H/s) and another demonstrating a substantially higher rate at 1,118,600 H/s. In comparison, SHA1 exhibited hash rates of 5,279 H/s and 1,115,500 H/s. Despite these variations, which suggest the use of different hardware or configurations, Hashcat achieved a 100% recovery rate for both hashing algorithms, cracking all the provided hashes efficiently, with the reported times suggesting an extremely fast operation.

Considering that SHA1 is theoretically more complex and secure than MD5 due to its larger 160-bit hash value compared to MD5's 128-bit value, the performance of Hashcat in this instance did not differentiate significantly between the two in terms of resistance to cracking. However, the cryptographic community recognizes SHA1 as stronger than MD5, although both are deemed obsolete for secure applications. It's worth noting that modern and secure alternatives like SHA-256 and SHA-3 are now preferred for cryptographic purposes due to their resistance to vulnerabilities known to affect MD5 and SHA1. The continued effectiveness of Hashcat against these older algorithms underscores the necessity of migrating to more advanced cryptographic solutions in security-sensitive environments.

5.4. Output

```
ralfuhed@tesla:~/CS495_CapstoneProject$ python3 main.py
Hashcat completed successfully. Check the log for details.
Cracked passwords: ['chelsea', '1234qwer', '123456789q', 'hendrix']
Hash type: MD5

*Caesar Cipher*
Encrypted passwords: ['inkrykg', '7890wckx', '789012345w', 'nktjxod']
Decrypted: ['chelsea', '1234qwer', '123456789q', 'hendrix']

*Vigenere Cipher*
Encrypted passwords: ['mlcazva', '1679xnej', '167923763a', 'rilsyzx']
Decrypted: ['chelsea', '1234qwer', '123456789q', 'hendrix']
```

6. Conclusion

The results of this project highlight the inadequacies of MD5 and SHA1 hashing algorithms, which were consistently breached using Hashcat through brute-force, dictionary, and rule-based attacks. These vulnerabilities reflect the algorithms' susceptibility to collision finding, a process expedited by modern computational capabilities, thus proving them insufficient for secure cryptographic applications. Furthermore, the experiment with classical ciphers, namely Caesar and Vigenère, served as a pedagogical exploration of basic encryption techniques, demonstrating significant weaknesses when tested against contemporary decryption tactics. Both ciphers were easily deciphered using methods like frequency analysis and pattern recognition, emphasizing their ineffectiveness in any practical scenario where data security is paramount.

The implications of these findings are profound for current cybersecurity practices. They underscore an urgent need for the adoption of more secure, sophisticated cryptographic algorithms across both commercial and private sectors to protect sensitive data against evolving cyber threats. This project also points to the necessity for continuous education and awareness about cryptographic security, aiming to enhance the knowledge of developers, IT professionals, and end-users alike. The journey through this project not only deepened the understanding of cryptographic principles but also spotlighted the practical challenges in safeguarding digital information. It became evident that historical encryption methods, while valuable for educational purposes, fail to meet the security requirements of today's digital systems. As we advance, it is imperative that we commit to adopting stronger encryption technologies and updating security protocols. This approach will be crucial in defending against sophisticated cyber threats and ensuring the integrity and confidentiality of digital communications.

7. Personal Gain

Through the course of this project, I have achieved a deeper understanding of cryptographic methods and password security, which has solidified my foundational knowledge and allowed me to appreciate the complexities of protecting sensitive information. My proficiency in advanced Python scripting and subprocess management has notably increased, facilitating the creation of efficient and robust security tools. This practical application has also provided me with valuable insights into the practical challenges that cybersecurity professionals face, particularly in the realms of data protection and threat mitigation. Moreover, I've developed enhanced problem-solving skills and honed my abilities in tool development, significantly contributing to my growth as a cybersecurity practitioner. This project has been instrumental

in advancing my technical skills and has prepared me for tackling complex security issues with a comprehensive and analytical approach.

8. Future Development

Building on this project's achievements, I am eager to enhance its capabilities. Future iterations will expand to include more advanced hashing algorithms and encryption techniques, embracing the latest in cryptographic security. The application of machine learning for predictive analysis of password vulnerabilities is also on the horizon. Moreover, efforts will be made to improve user interaction with the development of an intuitive interface, making these powerful tools more accessible. Continuous innovation and updates will ensure the project remains at the cutting edge of cybersecurity developments.

References

[1] Antigone, “Ancient Cybersecurity II: Cracking the Caesar Cipher,” Antigone, Sep. 16, 2021. https://antigonejournal.com/2021/09/cracking-caesar-cipher/

[2] B. Gleeson, “5 Password Cracking Techniques Used in Cyber Attacks | Proofpoint US,” Proofpoint, Sep. 12, 2023. https://www.proofpoint.com/us/blog/information-protection/password-cracking-techniques-used-in-cyber-attacks#:~:text=Password%20cracking%20typically%20refers%20to

[3] G. Simmons, “Vigenère Cipher | Cryptology,” Encyclopedia Britannica, Apr. 20, 2023. https://www.britannica.com/topic/Vigenere-cipher

[4] Hashcat, “hashcat - advanced password recovery,” Hashcat.net, 2018. https://hashcat.net/hashcat/

[5] “MD5 Hash Generator,” www.md5hashgenerator.com. https://www.md5hashgenerator.com

[6] “SHA1 Hash Generator,” www.md5hashgenerator.com.https://www.md5hashgenerator.com/sha1-generator.php