Automated Intrusion Detection: Evaluating Password Policies and Credential Security

Introduction:

Today, many organizations and users are still implementing weak password policies causing personal information to be vulnerable to cyberattacks. From confidential information like social security numbers to banking information, companies need to implement strong password policies to protect them. Some common user habits are reusing simple passwords such as “123456” or “qwerty” making them easily cracked by modern tools such as John the Ripper, Hashcat, Medusa, and more. Despite increased awareness of cybersecurity, weak and predictable passwords are a significant reason for data breaches. Therefore, throughout this semester I devoted my time and effort to exploring the effectiveness of three different password policies against cracking techniques using Hashcat to simulate real-world attacks. Through experimentation and analysis, the research aims to identify the strengths and limitations of the password policies to suggest improvements to overcome the growing threats from dictionary and brute-force attacks.

Literature Review:

Before diving into experimentation and analysis, a thorough literature review was necessary to obtain prior knowledge about common practices, existing password policies and cracking methods. An important factor about password security is the continuous reuse of passwords. As of 2023, 52% of users reuse the same password for multiple accounts, 35% use a different password for each account and 13% reuse the same password for all accounts (Redzepagic et al.,

2023). These statistics demonstrate a critical security vulnerability: if an attacker was able to crack one password then 13% of users would have their entire online presence exposed. Consequently, the problem of reusing passwords highlights the risk of personal information being leaked or compromised.

To address some of the standard password policies that helped shape my own in the experiment, it was necessary to examine the guidelines that organizations and cybersecurity frameworks implement. For example, traditional password policies generally require a minimum length of 6 to 8 characters, a mixture of uppercase and lowercase letters, numbers, and special characters. Many organizations also enforce mandatory password changes ranging from 30 to 180 days. Similarly, at the University of Miami students and faculty must change their passwords every 180 days to provide increased protection against attacks. However, with the standard policies enforced, they may lead to unintended user behaviors. For example, some users may create short, memorable passwords and when required to update their password they make short alterations. A specific case is where a user creates a password like “Strawberry123” and then changes it to “Strawberry12” when required. These limitations demonstrate the need to reevaluate password policies to enhance usability and security (Redzepagic et al., 2023).

In terms of the current password-cracking techniques, there are a range of methods used to recover or guess a user’s password. For example, a brute-force attack is where the attacker will attempt all possible combinations, but it is time coming and becomes impractical with longer passwords. Therefore, increasing the complexity of passwords such as requiring longer ones with mixtures of uppercase, numbers, and special characters makes brute-force attacks ineffective. However, attackers can utilize rainbow table attacks, which involve precomputed tables of hash values for common passwords that allow for faster matching against hashed databases.

Therefore, it is important to find a solution to reduce the vulnerabilities such as developing more advanced hashing algorithms (Kanta et al., 2020).

Experiment & Setup:

The experiment utilized a variety of tools to test the password policies, including Pycharm, Kali Linux, PyCryptodome, and Hashcat. For the first part of the experiment, PyCharm was used as an integrated development environment (IDE) for writing and testing code. The custom-built code included libraries like the password-strength package, which helped to define and evaluate password policies like the minimum length, upper case letters, numbers, and special characters. While the PyCryptodome library allowed for the creation of secure hashes. Furthermore, Kali Linux served as a powerful environment to test the varying tools such as John the Ripper and Hashcat. After evaluating both tools, Hashcat was the best choice for the project as it allowed the simulation of cracking hashed passwords efficiently.

Data Set:

There were a total of two datasets throughout the experiment. The first dataset is called "Rockyou.txt", which is a wordlist accessed from Github. The wordlist contains leaked passwords from a 2009 data breach, which fails many password policies making them an ideal sample for testing password strength and cracking techniques (Lurey). Due to the list containing millions of passwords, I used a manageable subset of the dataset for the experiment. The second dataset was custom-built, containing passwords that had varying complexities from easy to hard, generated using AI tools, text generators, and manually created inputs. These datasets allowed for the evaluation of the three password policies throughout the experiment.

Password Policies and Metrics:

During the experiment, a total of three password policies were tested. The first was a basic policy requiring a minimum of 8 characters, including one uppercase letter, one lowercase letter, one number, and one special character. This policy served as a reference point against stricter or customizable policies. The second policy became slightly more stringent by requiring a minimum length of 10 characters all kept with the same characteristics. Lastly, the final policy added further constraints by avoiding the use of personal information or sequences. However, this final policy had limitations as the implemented code on Pycharm didn't account for all possible variations and cases. This issue presents an area for improvement for future research on improving password policies and credential security. However, the data collected still remained accurate and provided insights on developing a solution.

All three password policies were evaluated based on three metrics: password strength, cracking time, and cost. Password strength was determined by whether the passwords met the criteria outlined in the policies, and were labeled as strong or weak. For example, the passwords had to adhere to the length, and character variety, and avoided personal information or common sequences. Furthermore, cracking time was measured by testing varying hashing techniques like MD5, SHA-256, and salted SHA-256, and simulating cracking attempts using Hashcat. Lastly, the cost was assessed by taking note of the computational resources required to perform the cracking attempts. One example of the cost metric would be the time consumption involved in executing each cracking attempt. Together, these metrics helped to understand the effectiveness of each policy and to make suggestions for future implementations.

To better understand the effectiveness of each password policy, it is necessary to go into depth between MD5 and SHA-256. The MD5 algorithm (Standard Message Digest 5) outputs a 128-bit hash and is known for its fast speeds. However, MD5 is no longer considered secure because it is

vulnerable to collisions and attacks. For example, two inputs such as passwords can output the same hash result. In contrast, SHA-256 (Secure Hash Algorithm 256-bit), a part of the SHA-2 family, is considered one of the most secure hashing algorithms. As a result, the algorithm is adopted by many industries due to its strong resistance to collisions. The property of salting in addition to the SHA-256 algorithm introduces randomness into the hashing process making it even more difficult for attackers to perform efficient brute-force attacks ((Prasanna et al., 2021). Therefore, the differences between MD5 and SHA-256 play a significant role in assessing how each password policy holds up against cracking methods.

Results & Analysis:

After testing the data in Pycharm and Kali Linux, I evaluated each password policy with the help of the three metrics: password strength, cracking time, and cost (*Note: results differ from presentation results due to code revisions*). The results were that hashing times were slightly longer for the custom data set, which included stronger and more complex passwords. To compare, the longest hashing time in the custom data set was 0.04906 seconds (MD5) compared to 0.04843 (MD5) seconds in the RockYou dataset. While the difference is small, it demonstrates the increased password complexities in the custom dataset. It is also crucial to note that the two datasets vary in size and content, with RockYou containing 1168 passwords and the custom dataset containing 185 passwords. Additionally, none of the passwords in the RockYou dataset passed the custom-built password strength policy indicating that all of them are not secure and would easily be vulnerable to attacks. In contrast, roughly 84% of the passwords passed all the policies, highlighting the effectiveness of stricter password requirements in improving security. The analysis of these results suggest that stronger password policies are the best choice even though the trade-off may be slightly longer hashing times and increased computational cost.

To better understand password security, a deeper analysis of each of the three password policies was conducted. As mentioned, all of the password policies for Rockyou.txt had a 0% pass rate, indicating failure to meet even basic requirements. Meanwhile, for the custom data set, there was a significantly higher pass rate. For example, under Policy #1 (Basic) and Policy #2 (Minimum 10 Characters), there was a pass rate between ~84%-86% and ~92% for Policy #3 (No Common Patterns). As a result, these statistics indicate that strong, well-crafted passwords that adhere to stricter password policies are more secure.

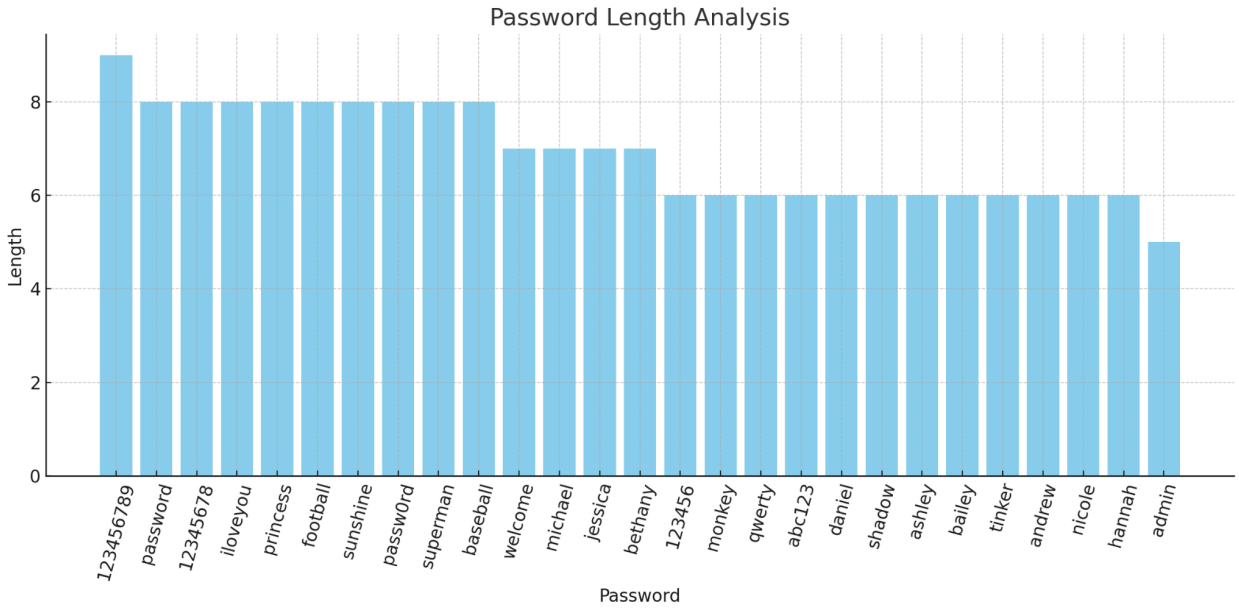
Figure 1: Comprehensive Password Security Metrics: Analyzing Strength, Hashing Algorithms, and Cracking Times

Metric	Rockyou.txt	Custom
Weak Passwords	1168	31
Strong Passwords	0	154
MD5- Avg. Hash Time	0.04843s	0.04906s
SHA-256 Avg. Hash Time	0.04834s	0.04852s
Salted SHA-256 Avg. Hash Time	0.04768s	0.04842s
Policy #1 (*Basic*) Passing & Failing Rates	Passed: 0% Failed: 100%	Passed: 85.95% Failed: 14.05%
Policy #2 (*10+)* Passing & Failing Rates	Passed: 0% Failed: 0%	Passed: 84.32% Failed: 15.68%
Policy #3 (No *Common Patterns*) Passing & Failing Rates	Passed: 0% Failed: 0%	Passed: 91.89% Failed: 8.11%

Not only did all of the passwords in Rockyou.txt fail to meet the 3 basic password policies, but many of them also lacked additional necessary complexities like special characters. For example, the password “michael” in the subset shown below is a model example of this vulnerability as it

is a common first name with no uppercase letters, numbers, or special characters. Therefore, this password makes it incredibly quick to guess or crack through brute force attacks.

Figure 2: Subset of Leaked Passwords from Rockyou.txt

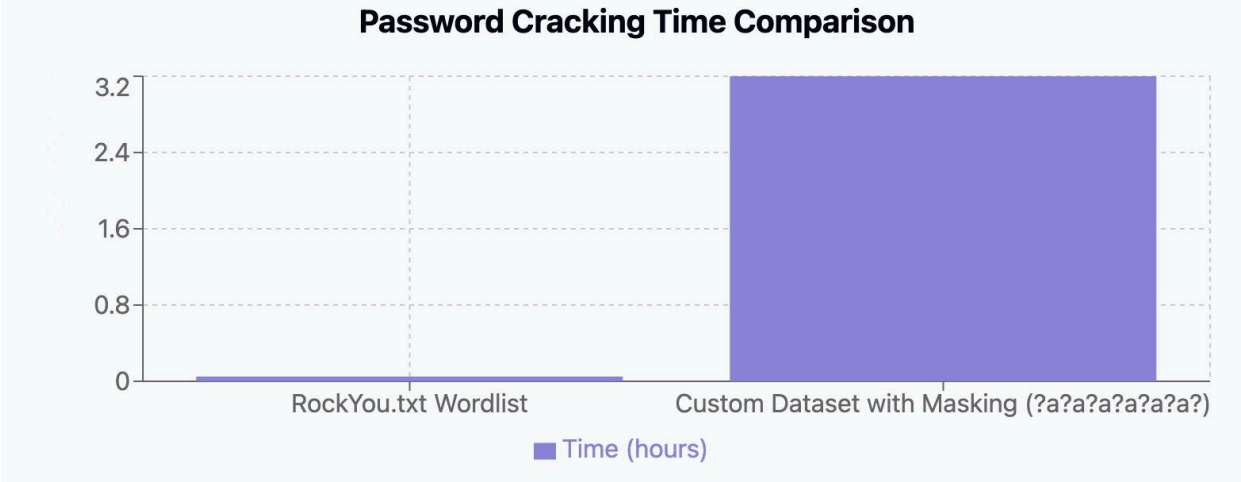


The second part of the experiment was switching to Kali Linux and implementing Hashcat where the cracking time was evaluated to test the effectiveness of the passwords on both datasets. For this part, a custom cracking list was implemented by manually supplying passwords into the list. If a password already exists in the cracking list then Hashcat can find it almost instantaneously. However, if the password is not in the list, Hashcat uses brute-force or mask attacks resulting in increased cracking time.

Next, to assess the metrics cost, a computationally expensive mask (?a?a?a?a?a?a?) was implemented on Dataset 2. The purpose of the mask is to increase security by accounting for all possible character combinations, including lowercase, uppercase, numbers, and special characters. After implementing Hashcat, the results were that on the Rockyou dataset, the

passwords were instantly cracked while the custom dataset took over three hours. Since the RockYou dataset was cracked instantly it demonstrates the ineffectiveness of security. For example, one of the passwords is “123456”, which is an extremely common pattern that can easily be guessed by many individuals. In contrast, the prolonged cracking time of the custom data set has more complex passwords with masking heavily increasing the resistance to brute-force attacks. This highlights the importance of enforcing stricter password policies and using comprehensive cracking techniques like masking.

Figure 3: Bar Graph Showing Password Cracking Time Comparison



Solution:

Conducting and analyzing the results allowed for the formulation of a generalized solution to the problem. First, implementing sophisticated hashing and masking techniques can significantly increase security by overcoming basic cracking tools like Hashcat. As shown in Figure 3, the custom dataset with masking resulted in over three hours of cracking. Furthermore, password policies should also require complex character patterns that include a combination of uppercase letters, lowercase letters, and special characters. Along with having complex combinations, the prevention of dictionary words and common sequences is encouraged, as these make passwords

vulnerable to brute-force and dictionary attacks. Finally, incorporating a minimum length preferably 12 or more characters adds additional security making it more computationally expensive for attackers to crack. Therefore, combining all of these recommendations based on the analysis of the results would likely enhance password security.

The conclusion of enforcing a minimum length of 12 or more with complexity requirements was obtained through the analysis. For example, testing passwords like those found in the Rockyou.txt dataset like “michael” again is extremely easy to crack. Therefore, requiring longer passwords with a mixture of uppercase and lowercase letters, numbers, and special characters significantly increases the security and resources needed to crack the password. As shown by the custom dataset, which included passwords like “MKz%YOFPIW03,” these passwords are both long and complex, making them more secure and resistant to common cracking methods.

To examine other current research available that helps solve the problem of evaluating password policies and credential security, a literature review was conducted again. The authors suggested utilizing a minimum length requirement of 16 characters instead of my solution of 12 characters as longer passwords result in longer cracking times due to more keyspace. Additionally, enforcing a mandatory unique password change increases security by prohibiting users from reusing passwords. Lastly, the authors recommended user training where they are informed on creating a password. For example, the passwords should not be overly complicated but rather long and memorable. The key component is to emphasize the importance of password length and simplicity while maintaining security. Therefore, this perspective demonstrates the importance of how a password should prioritize usability and security while being effective (Redzepagic et al., 2023).

Limitations:

While implementing strong password masking techniques such as hashing with salting significantly increases security as shown by the experiment, it can also be computationally expensive. Processing times can take longer, especially with large datasets or complex password policies. Furthermore, the additional processing power may lead to slower system performance, which can be a limitation in cases where quick access is necessary. The user experience may also be impacted if the system has limited resources.

There is also a usability versus security tradeoff: as password complexity increases users are more likely to reuse their passwords, write them down, and resort to common patterns. These behaviors are common when dealing with highly complex passwords, which can become overwhelming. However, this challenge was addressed in the literature review through user training. But if users follow the password policies that were conducted in this study then there could be vulnerabilities. For instance, when users reuse their passwords across multiple platforms their credentials are easily exposed to cyber attacks. These behaviors such as password reduction or resorting to predictable patterns can create vulnerabilities that attackers could exploit, ultimately undermining the goal of protecting user data. Therefore, while strong password policies are essential, they must be implemented with the possibility of user behavior like password reuse to minimize these limitations.

Conclusion:

To summarize the experiment, Rockyou.txt was consistently faster to hash due to weaker password policies, which made it more susceptible to attacks. In contrast, the custom data set significantly increased resistance to Hashcat attacks due to stronger password policies. While

salting and SHA-256 slightly improved security, it was the overall strength of the passwords that had the most significant impact on defense. This highlights the importance of strict password policies to solve our semester-long problem of enhancing credential security and preventing unauthorized access through automated intrusion.

The findings emphasize the need for regularly assessing password policies as attacking techniques are continuing to advance. By focusing on password complexity and implementing stronger hashing, the experiment contributed valuable insight toward effectively protecting user credentials.

Future Directions:

There are many future ways to improve the project and continue the contribution of enhancing password security. A way to improve the project would be expanding and adding more password variations to better simulate real-world testing environments. While the project only tested Hashcat it would be interesting to see how passwords are cracked under John the Ripper and other similar software. Furthermore, implementing more complex passwords and other hashing algorithms such as scrpyt and bycrpty will help to improve the defense. These enhancements would provide a more comprehensive analysis of password security and continue to provide sufficient protection.

Lastly, implementing Zero Knowledge Proofs (ZKPs) into the project would enhance privacy and security during validation by verifying password strength without revealing user credentials. This is an important component because disclosing sensitive information will reduce potential attacks. Although ZKP wasn't in the initial experimental setup, implementing them in future

research would reduce the vulnerability of password reuse, adding additional protection. (“CNIL”).

Finally, the semester-long project was informative and impactful because it provided an insight on how to improve password security. In the future, I can incorporate the solutions I’ve learned throughout the project such as having a longer and more complex password to provide the highest level of security.

Works Cited

CNIL. "Demonstration of a privacy-preserving age verification process." *Laboratoire d'Innovation Numérique de la CNIL (LINC)*, 2023, linc.cnil.fr/en/demonstration-privacy-preserving-age-verification-process. Accessed 27 Apr. 2025.

Kanta, Aikaterini, Iwen Coisel, and Mark Scanlon. "A survey exploring open source intelligence for smarter password cracking." *Journal of Cybersecurity*, vol. 7, no. 3, 2021, pp. 171–179, doi:10.1016/j.jcyber.2021.100123. Accessed 27 Apr. 2025.

Lurey, Craig. "Understanding RockYou.txt: A Tool for Security and a Weapon for Hackers." *Keeper Security Blog*, 4 Aug. 2023, www.keepersecurity.com/blog/2023/08/04/understanding-rockyou-txt-a-tool-for-security-and-a-weapon-for-hackers/. Accessed 30 Apr. 2025.

Prasanna, Shwetha R., and B.S. Premananda. "Performance Analysis of MD5 and SHA-256 Algorithms to Maintain Data Integrity." *2021 6th International Conference on Computing, Communication and Security (ICCCS)*, IEEE, 2021, doi:10.1109/ICCCS51439.2021.9573660. Accessed 27 Apr. 2025.

Redzepagic, Jasmin, Vedran Dakic, Josip Stanesic, and Matej Basic. "Analysis of Password Security Policies and Their Implications on Real-Life Security." *Proceedings of the 34th DAAAM International Symposium*, 2023, www.daaam.info/Downloads/Pdfs/proceedings/proceedings_2023/011.pdf. Accessed 27 Apr. 2025.