12-2021

CollaborCrack: A Collaborative Password Cracking Solution for Windows Penetration Testing

Andrew Griess
agriess@unomaha.edu

CollaborCrack: A collaborative password cracking solution for Windows penetration testing

Andrew Griess

University of Nebraska at Omaha

Advisor: Dr. Matthew Hale

December 2021

Abstract

Cybersecurity professionals attempt to crack password hashes during penetration tests to determine if they are strong enough. A password hash is a way to encode a password securely. This paper describes a proof-of-concept program called CollaborCrack, a team-based password cracking solution. CollaborCrack addresses the issues of computational complexity, remote cracking security, duplication of work, and the cost associated with password cracking. To address computational complexity, CollaborCrack enables remote password cracking. Remote cracking requires additional safeguards, which CollaborCrack mitigates by storing sensitive information locally. To reduce the duplication of work, CollaborCrack provides a shared interface designed around collaboration and teamwork. CollaborCrack reduces costs by decreasing the time it takes to crack groups of passwords and the number of password cracking computers needed. CollaborCrack breaks the traditional password cracking process into two parts: a collaboration client and collaboration server. CollaborCrack's client serves as a shared password cracking interface for collaborating teams. The client organizes notes and facilitates collaboration among team members. CollaborCrack's server increases password cracking efficiency while eliminating duplication of effort by allowing multiple team members to submit passwords to the same cracking server. If security professionals adopt this proof-of-concept, CollaborCrack could establish a more efficient and collaborative password cracking experience.

Keywords: brute force, cybersecurity, password, password cracking, penetration testing, windows

1. Introduction

Passwords are ubiquitous. We use them to log into our personal computers, social media accounts, and even our banks. Despite how critical passwords are, humans are not very good at creating them. People generally pick passwords that are short and easy to remember, contain common words, follow predictable patterns, or include personal information (Luyten, 2020). It would be bad enough if people just used weak passwords, but 53 percent of people also reuse passwords on multiple accounts, and 44 percent of people use their personal passwords at work ("53% of People Admit They Reuse the Same Password for Multiple Accounts," 2020).

Strong passwords are more important now than they have ever been. Computers are becoming faster, which means guessing passwords is getting easier. For many businesses, all it takes is one user having a weak password to enable hackers to get inside a network or increase privileges. This means that good password practices are essential to successful businesses. Penetration testers have been analyzing the security of companies for years. One step in this process is to try and break into user accounts to ensure that their passwords are strong enough. To accomplish this, penetration testers use a technique called password cracking.

In this report, I define background information necessary to understand the complexities and importance of password cracking. Then, I explore the current process of password cracking and describe the challenges that password crackers face. For each challenge, I propose how my proof-of-concept program, CollaborCrack, seeks to mitigate its impact. After this, I break CollaborCrack down into its two parts and describe how they function. Finally, I conclude on the benefits of CollaborCrack if adapted by industry professionals.

2. Background

The following section defines the background information necessary to understand the problem and proposed solution. This background information includes Windows domains, penetration testing, password hashing, and password cracking.

2.1 Windows Domains

Windows is the most common operating system for a computer. In June 2021, Windows was used by 73% of desktop computers (Liu, 2021). A Windows domain interconnects Windows computers so that they can be centrally managed and allow users to access shared resources. Nearly all schools and many larger businesses use Windows domains. One helpful feature of a Windows domain is that users can log in with the same credentials on multiple systems. The centralized management of Windows permits different permission levels for different users. Some users may only have permission to log in to one computer, while others can remotely administer the whole domain. Windows domains can also group users with the same level of permissions. For example, one group could be made for students, and one group could be made for teachers. Some groups exist by default on a windows domain. One default group is domain admin. Domain admins have administrator rights of all domain systems and can configure domain settings such as what type of encryption is used and what users are in what group. Windows domains are rather complex and therefore have complicated password management. Passwords are stored and transmitted in many different ways based on configuration and use. Common windows credential storage methods include LM, NTLM, Net-NTLMv1, and NTLMv2 (Gombos, 2018).

2.2 Penetration Testing

Penetration testing identifies security flaws in a computer system by trying to break into it. When testing a Windows domain, penetration testers try to increase their privileges. One way to do this is to try and access different computer or user accounts. This process helps identify what an attacker may be able to access. The value of an account for a penetration tester is mainly based on what permissions they have. For instance, if a teacher's account were compromised, a penetration tester would be able to do anything that a member of the teacher group could, such as accessing a teacher file share. As an end goal, a penetration tester will try and access a high privilege account on the domain to have as much access as possible. Complete access proves that a motivated attacker could fully compromise the environment and that the company needs to increase its security. To gain this access, penetration testers look to exploit unpatched systems, misconfigurations, and weak passwords. Penetration testers compromise weak passwords by stealing them from systems or collecting them as they travel on the network. Fortunately, stealing these passwords is only the first step. Passwords typically are not stored in clear text; rather, they are *hashed*.

2.3 Password Hashing

Systems should never store passwords in plaintext. If they were and the database holding the passwords was stolen, an attacker would immediately have the passwords to every account. Instead of storing plaintext passwords, the industry standard is that these passwords are stored as a cryptographic hash.

A cryptographic hash is a *one-way* mathematical function that creates a *fixed-length* output typically represented in hexadecimal. One trivial example of a hash is X MOD 2. This equals 1 if a number *X* is odd and 0 if it is even. Thus, if the number 9 were entered into this function, it would output 1. No matter what input size is used, the function will always output a 0 or a 1. Another component of a hashing function is that it is impossible to know what was entered when analyzing the output. This is a very simple and very bad example of a hash. More standard hashes like SHA-256 have 2^256 possible outputs equating to around 1.1579209e+77 output options.

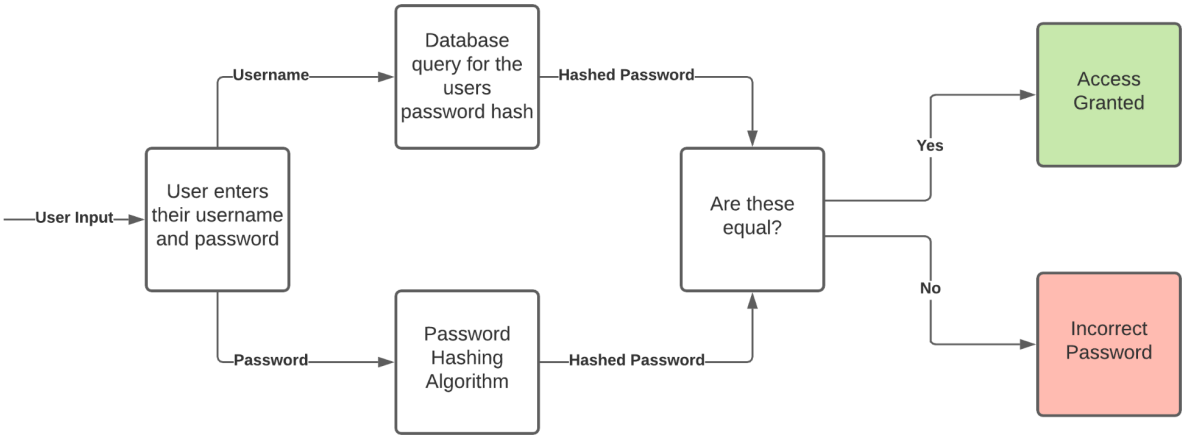

Figure 1: An example of comparing an entered password against a password stored in a database.

The first time a user logs in to an account, their password's hash value is stored in a database. Since password hashes are one way, every time a user enters their password, the application must recalculate its hash value to compare it against the previously stored password hash. **Figure 1** shows how this typically occurs. If the stored hash and the generated hash are equal, then the passwords are equal.

2.4 Password Cracking

Through computational complexity, hashing functions make it incredibly difficult for penetration testers to reverse the function. Generally, this means the only way to know which password created a particular hash is to guess it. Guessing which password made a specific hash is called password cracking. Depending on its speed, a modern computer can guess between 10,000 and 1 billion passwords per second (Scott, 2020). This high guess rate is why passwords must be sufficiently long and complex. A long and complex password is far less likely to be guessed. A penetration tester will typically test if a password is strong by comparing it against a dictionary of the most common passwords. This type of brute force technique is called a dictionary brute force. Then, if the password is not guessed, the tester may try a simple brute force that attempts every possible value of length one, then length two, then length three, and so on. This type of brute force only works on small passwords. **Figure 2** shows how long this type of brute force attack would take based on length and complexity.

Number of Characters	Numbers Only	Lowercase Letters	Uppercase and Lowercase Letters	Uppercase Letters, Lowercase Letter, and Numbers	Uppercase Letters, Lowercase Letters, Numbers, and Symbols
Less than 6	Instantly	Instantly	Instantly	Instantly	Instantly
6	Instantly	Instantly	Instantly	1 Second	5 Seconds
7	Instantly	Instantly	25 Seconds	1 Minute	6 Minutes
8	Instantly	5 Seconds	22 Minutes	1 Hour	8 Hours
9	Instantly	2 Minutes	19 Hours	3 Days	21 Days
10	Instantly	58 Minutes	31 Days	7 Months	5 Years
11	2 Seconds	1 Day	5 Years	41 Years	400 Years
12	25 Seconds	21 Days	300 Years	2k Years	34k Years
13	4 Minutes	1 Year	16k Years	100k Years	2m Years
14	41 Minutes	51 Years	800k Years	9m Years	200m Years
15	6 Hours	1k Years	43m Years	600m Years	Over 1 Billion Years
16	2 Days	34k Years	Over 1 Billion Years	Over 1 Billion Years	Over 1 Billion Years
17	4 Weeks	800k Years	Over 1 Billion Years	Over 1 Billion Years	Over 1 Billion Years
18	9 Months	23m Years	Over 1 Billion Years	Over 1 Billion Years	Over 1 Billion Years
19	7 Years	600m Years	Over 1 Billion Years	Over 1 Billion Years	Over 1 Billion Years
20	79 Years	Over 1 Billion Years	Over 1 Billion Years	Over 1 Billion Years	Over 1 Billion Years
Key:	Trivial	Lowly Motivated Attacker	Highly Motivated Attacker	Future Threat	Good

Figure 2: The time it takes to brute force a password, based on length and complexity.

Note. Data from in table from security.org (*How secure is my password?*).

3. Why Collaboratively Crack?

This section details challenges that penetration testers face while trying to crack passwords. For each of these challenges, I describe how the proof-of-concept program CollaborCrack addresses these issues and strives to reduce them.

3.1 Computational Complexity

For a penetration tester to thoroughly test a password hash, they must guess enough passwords to rule out the use of a weak password. Therefore, testing the strength of a hashed password requires a sufficiently fast computer. However, penetration testers may not use a computer fast enough to adequately test the password's strength within the required time limit. Many testers work off of laptops with weaker hardware.

CollaborCrack addresses this issue by allowing penetration testers to crack passwords on a remote server rather than on their local machine. While not reducing the number of computations required to brute force a password, CollaborCrack provides an extra option to penetration testers with off-site hardware. With this, a penetration testing business would only need to have one computer fast enough to crack passwords on instead of requiring one for every penetration tester.

3.2 Remote Password Cracking Security

When sending a password hash to a remote server to be cracked, the penetration tester must remember what username corresponds with what password hash. It is generally inadvisable to send a username and a password together to another location because this may introduce a security risk. A password by itself is just a password. A username and a password together may

reveal enough information to be sensitive. This imposes an additional challenge to a penetration tester of securely sending a password hash off-site without losing the correlated account.

CollaborCrack addresses this security risk by only transmitting the password and its hash when communicating with the remote cracking server. Additionally, within CollaborCrack, the username and password correlations are tracked automatically with no room for user error. Information about a password hash, including its corresponding username, is stored locally rather than transmitted to the password cracking server.

3.3 Duplication of Work

Password cracking has traditionally been an individual activity. Penetration testers working on a team often struggle to have situational awareness about what their fellow testers have already tried to crack. This can lead to duplication of work between team members - thus decreasing team efficiency. One team member may come across a hashed password and try to crack it, unaware that another team member has already broken it.

CollaborCrack is a tool designed around collaboration and teamwork. As a shared interface, CollaborCrack lets any user on a team see who is trying to crack what, which can help multiple penetration testers from duplicating work. More than just showing what passwords are being cracked, CollaborCrack also lets users add notes about a particular password hash, such as where it is from and what to use it for if it is cracked. Only one team member needs to run CollaborCrack on their system during a penetration test. Letting other team members focus on different aspects of their assessment. Additionally, by storing all of the password cracking notes

in one location, penetration testing reports are more accessible, enabling them to be written up by one person rather than by every member. With additional research, CollaborCrack could be expanded to generate reports for the password cracking component automatically.

3.4 Cost

Password cracking is expensive in how much time it takes and the material costs of a strong password cracking computer. The faster a computer can crack password hashes, the more expensive it will be.

CollaborCrack looks to reduce the time it takes to crack passwords and the number of fast computers needed. CollaborCrack minimizes the number of fast computers by enabling different penetration testers to use the same password-cracking computer. Using the same computer to crack multiple password hashes barely slows it down. CollaborCrack optimizes how it cracks multiple hashes. It groups all hashes of the same type and tries to break them at the same time—this way, the hash value of a guess only needs to be computed once. If penetration testers are responsible for their own password cracking, they may not crack passwords in groups. Cracking passwords individually is slower and uses more computer power.

4. Implementation Details

What makes CollaborCrack unique from a traditional password cracker is that it operates with two intercommunicating components. The first part is a collaboration client served to an internal network via a website. The second part is a collaboration server that takes in password hashes

and tries to guess them. This internal component acts as a client to the remote server. **Figure 3** shows how the CollaborCrack would function with one team with four members. The system works with any number of collaboration clients and team members. The collaboration client and server can reside on the same host if needed. This functionality can provide the advantages of CollaborCrack without needing another computer.

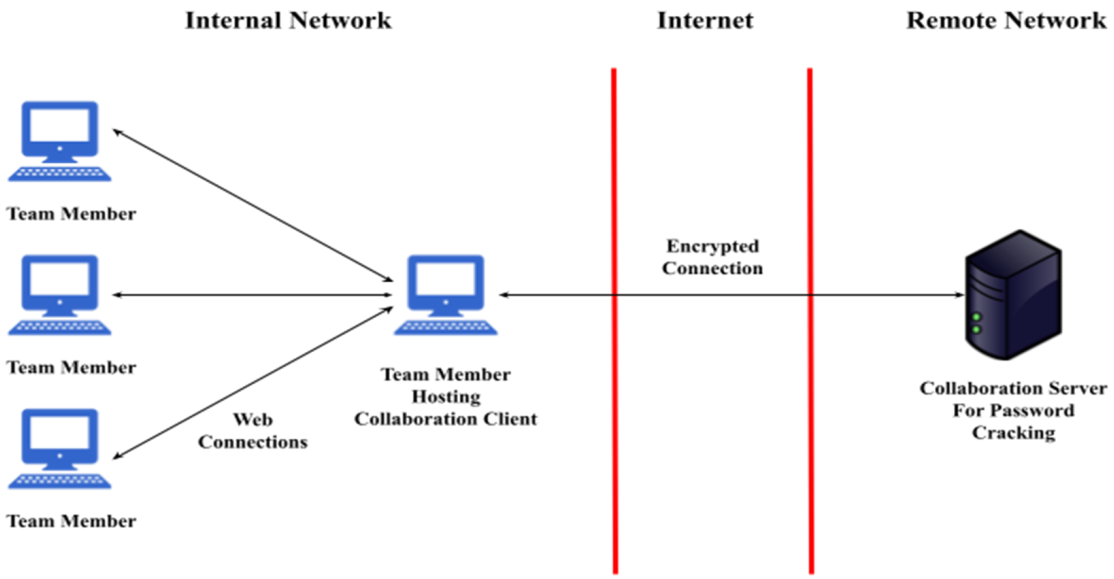

Figure 3: Network diagram showing the flow of the CollaborCrack tool.

4.1 Collaboration Client

The collaboration client resides on an internal network and takes web connections from any number of users. It is written in the programming language Python 3 and uses Flask to serve its web pages. The first time a user attempts to connect the collaboration client, it will prompt them to create an account, as shown in **Figure 4**. Once that account is created, they can log in to the client with their credentials.

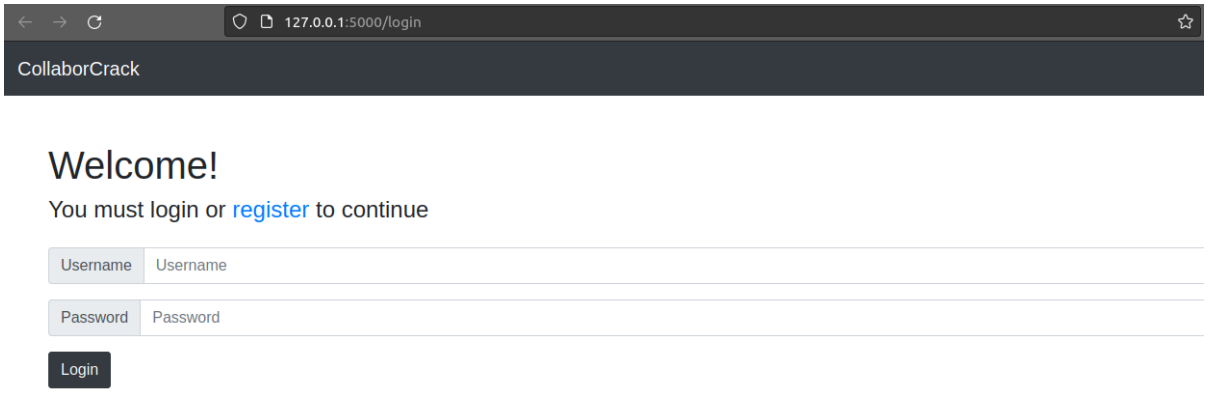

Figure 4: CollaborCrack login page.

Now, the user can join or create a team, as shown in **Figure 5**. In this project, a team is a collection of users working on the same penetration test. Any team member will be able to see all of the passwords and hashes associated with a particular test. A user can be on multiple teams.

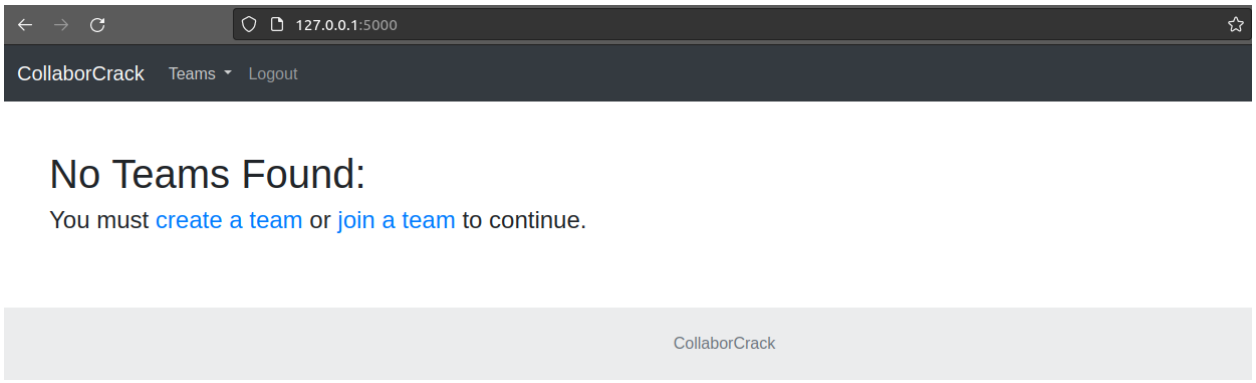

Figure 5: CollaborCrack empty team page.

Once logged in to a team, a user can enter a team's page. This page shows all of the hashes submitted by users and any corresponding notes. Each hash also denotes who submitted which hash on which date. **Figure 6** depicts what a user on a team would see.

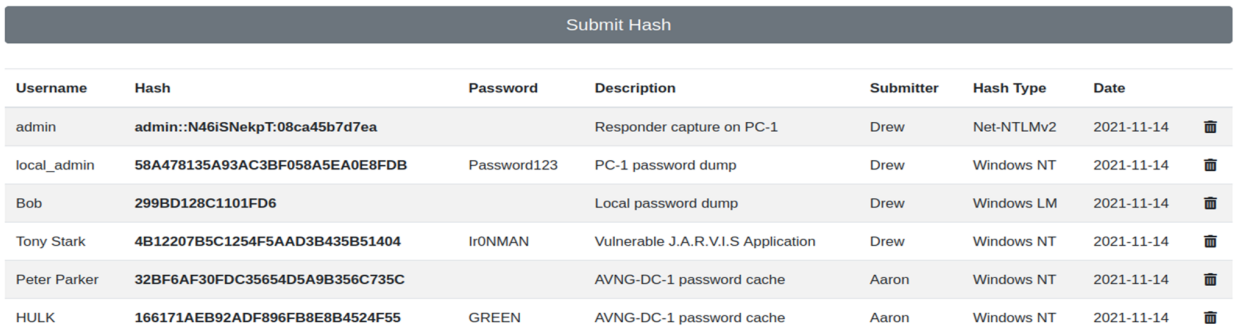

Figure 6: CollaborCrack team with multiple hashes and submitters.

When a user submits a hash to the collaborative client, it is automatically added to the queue on the collaboration server. Then, the collaboration client will continually poll the collaboration server to see if it has completed the client's request to break a particular password hash. If the password is broken, it will be displayed to the penetration tester. This collaboration client only stores and organizes information. CollaborCrack's password cracking ability is only as effective as its collaboration server.

4.2 Collaboration Server

The collaboration server takes in password hash requests and attempts to crack them. The interface for the collaboration server is straightforward. A collaboration client makes a web request to https://<ip>/crack/<hash> where <ip> is the IP address of the server and <hash> is the value of the hash. The collaboration server is written in Python 3 and uses Flask to run the webserver. The server returns an Unknown Hash Type error when receiving an invalid hash. To check if a hash is invalid, it looks at the length and character set of a hash. When a brute force is still running on a hash, it returns the RUNNING, as shown in **Figure 7**.

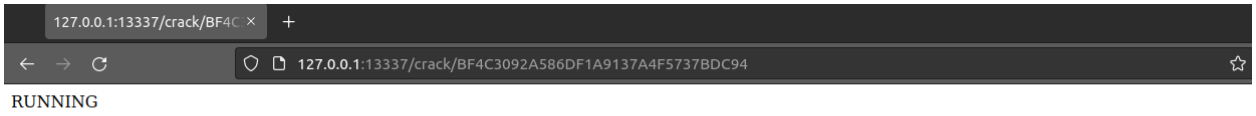

Figure 7: Collaboration server receiving a new hash.

Once the server has finished cracking the hash, it will store the cracked hash in a database. Every time a hash is submitted in the future, it is checked against this database. This datastore prevents a duplication of effort by recalculating the same hash twice. After the server has time to crack the hash shown in **Figure 7**, it will display the cracked password after a refresh, as shown in **Figure 8**.

Figure 8: Collaboration server after cracking a hash.

This interaction uses HTTPS to stay secure. This encrypts the traffic between the collaboration client and collaboration server in the same way as almost every website. CollaborCrack does not currently require authentication to submit a hash to be cracked. Future implementations should include authentication to ensure resources are not wasted and that sensitive information is not disclosed.

The collaboration server uses Hashcat to perform its password cracking. Hashcat is a highly optimized password cracking tool commonly used in the industry for password cracking. By using a pre-established tool to complete password cracking, CollaborCrack benefits from the experience of many others to maximize its cracking speed. The collaboration server manages the

hashes as jobs sent to Hashcat and then waits for them to finish. Once they finish, the collaboration server records the output into a separate file used for tracking completed attempts.

Currently, the collaboration server is configured to use a dictionary attack against any supplied set of password hashes. This means that if the password is not within the selected wordlist, there is no chance of cracking the hash. The collaboration server records when a password hash has exhausted all attempts and displays that to the user, as shown in **Figure 9**.

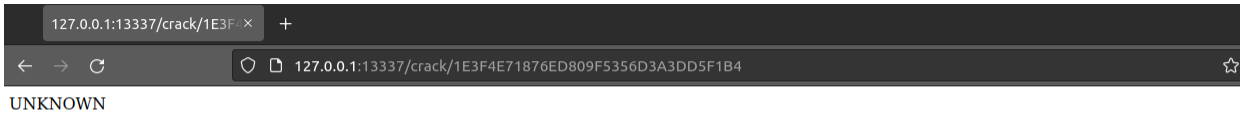

Figure 9: Collaboration server after failing to crack a hash.

When receiving a password to crack, the collaboration server groups hashes of the same hash type to be cracked together. This dramatically increases the speed of breaking a list of hashes. CollaborCrack currently works on the common Windows hash types LM, NTLM, Net-NTLMv1, and NTLMv2, but can be expanded to work on any hash type. If an organization had multiple penetration testing teams, they would be able to use the same powerful computer to crack passwords at the same time without risking data spillage or needing communication.

5. Conclusion

Password cracking is a common activity within security penetration testing methodologies. Teams working on the same penetration test often struggle to collaborate due to technical and tooling limitations within the industry. A particularly confounding problem for teams of password crackers is minimizing overlap (i.e. when two tests try to crack the same password). Overlap significantly slows down the speed of the test and wastes computing resources and power.

CollaborCrack is a web-based tool that seeks to close this gap by facilitating team-based cracking exercises in a way that minimizes wasted resources. CollaborCrack allows multiple collaborating team members to submit passwords for cracking into a shared queue. This queue keeps track of prior password cracking attempts, groups passwords of the same hash together, and removes redundancies.

If adopted by industry professionals, the outcomes of this proof-of-concept have the potential to make significant impacts in several areas. For team members, it can reduce the time required to brute force during a penetration test by a factor of as much as the number of team members on a project (e.g., a factor of 3 for a three-person team, if all team members tried to crack the same type of password hash). This reduction saves time and significantly saves on power consumption since CPUs and GPUs consume significant resources to crack passwords. Thus, savings have implicit broader impacts that make security penetration testing more "green" by helping to reduce fossil fuel consumption (vis-à-vis reduced power consumption).

References

53% of people admit they reuse the same password for multiple accounts. (2020, May 7). Security Magazine. https://www.securitymagazine.com/articles/92331-of-people-admit-they-reuse-the-same-password-for-multiple-accounts

Gombos, P. (2018, February 20). *LM, NTLM, Net-NTLMv2, oh my!* Medium. https://medium.com/@petergombos/lm-ntlm-net-ntlmv2-oh-my-a9b235c58ed4

How secure is my password? (n.d.). Security.org. Retrieved December 10, 2021, from https://www.security.org/how-secure-is-my-password/

Liu, S. (2021). *Desktop OS market share*. Statista. https://www.statista.com/statistics/218089/global-market-share-of-windows-7/

Luyten, D. (2020, October 27). *The human side of password security*. Geant. https://connect.geant.org/2020/10/27/the-human-side-of-password-security

Scott, B. (2020, August 17). *Learning password security jargon: Brute Force Attack.* Nordpass. https://nordpass.com/blog/brute-force-attack/