Spring 2015

Analysis of Password Cracking Methods & Applications

John A. Chester
The University Of Akron, jac177@zips.uakron.edu

Analysis of Password Cracking Methods & Applications

John A. Chester

The University of Akron

Abstract -- This project examines the nature of password cracking and modern applications. Several applications for different platforms are studied. Different methods of cracking are explained, including dictionary attack, brute force, and rainbow tables. Password cracking across different mediums is examined. Hashing and how it affects password cracking is discussed. An implementation of two hash-based password cracking algorithms is developed, along with experimental results of their efficiency.

I. Introduction

Password cracking is the process of either guessing or recovering a password from stored locations or from a data transmission system [1]. Since the introduction of a computer password, hackers have tried to crack passwords but it has only became popular and practical within the last ten years [2].

The typical way password cracking works is to get a file containing user hashed passwords and then run a cracker against the file to try to get matches for all of the hashes, thus revealing all of the passwords in the file. While the latter part is typically uncomfortably fast, the first can be very difficult and many approaches may need to be taken to penetrate a system's security to obtain a password file. However, using simple, targeted Google searches it has become easier to gather unprotected hashes of users.

II. Hashing

When passwords are stored on a system they are done so by first going through a hashing algorithm, then that hashed string is stored into a file. Hashing algorithms are one-way functions that turn a string of data into a fixed-length "fingerprint" that cannot be reversed [4]. A hash for the word “computer” would be “df53ca268240ca7667c8566ee54568a,” using the popular hashing algorithm, MD5. Only cryptographic hash functions are used to implement password hashing [4]. Some common examples are MD5, SHA-1, LM, NTLM, and Whirlpool [3].

Since typically a password to be checked against the password file has to run through the same hash algorithm, a way to make hashes more secure is to use a hashing algorithm that takes a longer amount of computational time, like SHA-512. That way it will take longer for each password guess, in an automated password cracking program, to be run through the hash algorithm to be verified. A common password cracking technique is to generate all of the hashes to be verified ahead of time. That way all the cracker has to do is compare all of the hashes in the password file with the ones it has already generated. Since the hashes are already computed, the time it takes to run a password through the hash is irrelevant and makes the strength of the hash alone useless. This table of pre-computed hashes is called a rainbow table [4].

A common way to circumvent a rainbow table attack is to use something called a *salt*. A salt is a randomized string appended to the end of a user's password. This is then hashed together. In addition to providing a more secure password, salts also ensure that the hash generated for two users with same password will be different. For example a password, like "passwd" could be hashed along with "QxLUF1bgIAdeQX" concatenated with it for one user

and "bv5PehSMfV11Cd" concatenated for another. That would give the MD5 hashes: 7bc4372cb5ca16d37bf8d688d82a19b1 and 9436b51ce8857e7487eedb9998b4ff50 respectively. In order for a password to be verified by a system the salt would be needed, so it is typically stored in a user account database or as part of the hash string itself [4].

A problem with salts is that they may be either reused or too short. If it is reused an attacker can simply apply the salt to each password guess before they hash it. If it is too short the salt can be brute-forced and the possibilities added to the rainbow table. To solve this problem it is recommended to generate a good salt using a Cryptographically Secure Pseudo-Random Number Generator (CSPRNG). It is similar to a pseudo-random number generator except that it is cryptographically secure and much more random [4].

III. Password File & System Penetration

One of the greatest challenges of password cracking is the act of obtaining the file which contains all of the hashed passwords. After this step has been taken all an attacker needs to do is run a password cracking tool on the file until its cracked, and usually it doesn't take all that long. For example, it is possible for Hashcat to crack MD5 hashes at a rate of 92672M h/s, measured in “hashes” per second [16].

The easiest and quickest way for attackers to obtain passwords is through specialized Google search strings called dorks. These are strange, yet well structured Google searches that can find information on a website that would otherwise not be easily accessible. The most important thing hackers are interested in is the password file. Some good Google dorks they would use to do this are

intext:phpMyAdmin SQL Dump filetype:sql intext:INSERT INTO `admin` (`id`, `user`, `password`) VALUES – github

This shows results of SQL dump files which might contain user names and passwords

“Index of /” + password.txt

intitle:”index of” passwd

These dorks search for Unix based password files. There are a large amount of these allowing attackers to try many different searches to find the right information. All it takes is a bit of time [6].

More recently hackers have been able to obtain passwords on Unix based systems through the Shellshock vulnerability. By passing in a malicious shell command like () {:;}; /bin/cat /etc/passwd, attackers are able to have the password file dumped to the console window. This allows hackers to obtain the hashed passwords on some web servers [5].

Otherwise attackers can use local attacks on the password file. Windows operating systems store their users hashed passwords in the SAM hive in the registry. Windows XP uses an outdated and weak hash, LM, to hash the user's password. Windows Vista and other newer versions of windows use NTLM which is a 128-bit hash and is much more secure than LM but certainly still breakable under the right conditions. The password file is not accessible while the operating system is running however. In order for an attacker to recover these hashed passwords one would need to access the files offline, using something like a Live CD [8].

IV. Password Cracking Applications

The way password cracking usually works is either to pull passwords out of dictionary files or generate them, then run them through a hash algorithm or to lookup the hashes in a rainbow table and compare them. This section aims to cover some of the more popular password cracking tools that are used today both by attackers and system administrators to test the security of their system.

JohnTheRipper:

This is a fast and popular local password cracking application from Openwall. It was created with the intention of detecting weak UNIX passwords but can also easily crack weak Windows LM hashes [7].

RainbowCrack:

This extremely popular password cracking application uses rainbow tables generated from the application itself to find a matching password hash. It can crack a wide variety of hashes including LM, NTLM, MD5, and SHA-1 [9]. It generates rainbow chains which are stored in a special format in the binary rainbow table files, and then checks those files against the password file in order to find a match and then resolve the password using a lookup method [10]. This method is much faster than brute force as the hashes are computed before the crack saving a lot of computational time especially over billions of password attempts. The obvious downside is that the rainbow table files take up a significant amount of disk space, usually in the 100s of gigabytes.

Cain and Abel:

This Windows password cracking tool is very versatile in that it can not only crack passwords locally but it can act as a sniffer on the network, record VoIP conversations, and analyze routing protocols. As far as the cracking itself it can perform brute force attacks, cryptanalysis attacks, and uncover cached passwords. It was developed with network administrators and penetration testers in mind [1].

LOphtCrack:

This is a versatile Windows tool that can attack Windows Workstations and Windows Servers. The application attacks by using dictionary, brute force, and pre-computed hashes. It is known to be extremely fast and has an easy to use user interface [11].

Aircrack-NG:

This is a unique tool in that it is designed to crack a different type of password, Wi-Fi passwords. It can crack WEP or WPA passwords [12]. It analyses the encrypted packets captured over a wireless network and attempts to crack the password with its cracking algorithm [12]. The application uses the FMS (Fluhrer, Mantin, and Shamir) attack which is a special type of cryptanalysis attack. FMS allows the attacker to recover the key after a large amount of packets have been sent [13].

Hashcat:

A very powerful free cracking tool for Windows, OSX, and Linux. It features a rule based attack system and multi-threading for very fast performance. Hashcat supports a long

range of hashes and can perform the attack in numerous ways such as dictionary and brute-force [3].

V. Prevention

The most important thing to take away from this paper is how one can prevent these attacks from being successful. One of the best techniques for stronger passwords is to use a passphrase. For example: IOnlyEatPieOnDaysThatEndWithY. Mixing in different cases and adding a symbol or two enhances the security greatly but it is the enormous length that truly keeps the password secure. Each character added increases the number of possible permutations of the password and makes it significantly hard with each one. Another great benefit is that these are usually easy to remember as they rhyme or are an easily memorable quote or phrase. A longstanding rule of thumb for password length is 8 characters but as computers get more powerful this often won't be long enough to avert a successful attack, especially if the password is easy to guess or worse, a dictionary word. Other good techniques include changing passwords regularly (about every 3 months), not using the same password on multiple sites, and avoiding dictionary words and names.

For system administrators security is a little different as they have more control over the security of their system. One of the best ways they can test their system is to try to attack it themselves. This is one type of system penetration testing. A good way to do this is to use a password cracking tool like JohnTheRipper to check the integrity of their password file. Also they should be sure to use strong salts on their password hashes to make them much more secure, and enforce password expiration policies.

After system penetration testing, administrators should use the results to implement changes to some of the system configurations to prevent successful attacks. One of the first things they should do is disable the outdated Windows LM hashes. These passwords are typically easily cracked. Luckily they also can be easily disabled. Either by changing the registry or requiring a password longer than 14 characters, which should be done anyway for a secure system. Another recommendation is to change the name of highly privileged accounts. Changing the name of Administrator to something else will render many attacker's tools useless [14].

Administrators should also consider the physical safety of their machines on the network. An attacker can easily get a local password file by booting to a USB drive or LiveCD. To prevent that and other local attacks like it, administrators should change their BIOS settings so that the system cannot boot to anything but the local hard disk, then protect the BIOS with a password [14].

VI. Program Implementation Details

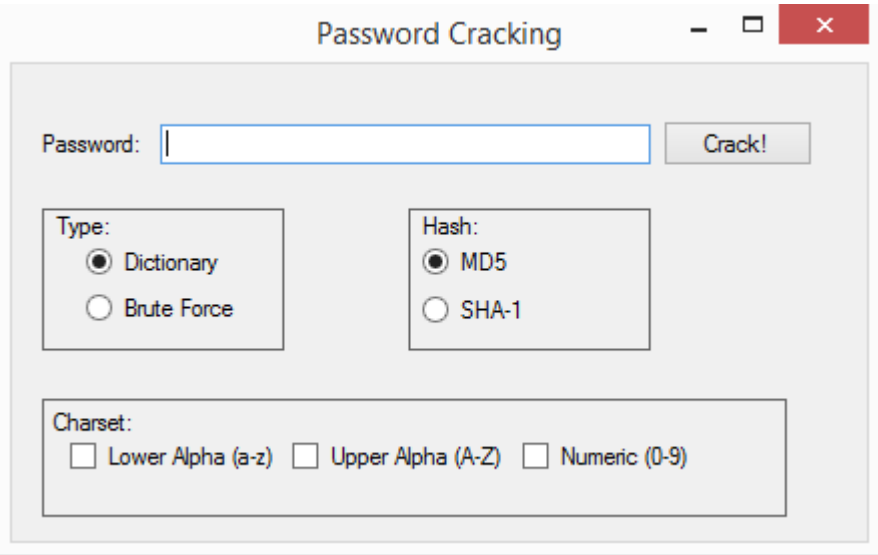

Figure 1.

The program written for this project demonstrates password cracking. It performs two different types of attacks and supports two different hashes. The application is written in C# and has a GUI which lets a user enter a password, choose an attack, hashing algorithm, and character set. Then the user clicks the crack button to initiate the attack on the single password. See Figure 1. Results are displayed on how many attempts were made to crack the password successfully and how long the attack took.

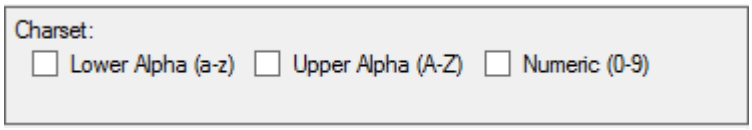

Figure 2.

In the application there is a checkbox section (see above) which allows the user to choose which characters are in the character set that they want to use to try to crack the password, shown in Figure 2. The options are lowercase alphabet characters (a-z), uppercase alphabet characters (A-Z), and numeric characters (0-9). Any combination of them may be chosen for consideration. This section will only affect the time of the brute force attacks. The dictionary attack is run

linearly and will check every password in the dictionary file until it finds a match or searches through the entire file.

The program supports two hashing algorithms: MD5 and SHA-1. The user can choose which is to be used during the crack by choosing the appropriate radio button. Once the crack button has been clicked a function pointer, or delegate, will point to the chosen hash function during the crack execution. The password entered in the text box is hashed first and saved as a variable then the program checks every word in the dictionary against the hashed password until there is a match or it runs out of words. For each attempt, the word being considered is hashed right before it is compared with the hashed password. This makes the execution time similar to a real password crack.

The two different attacks supported are: Dictionary and Brute Force. The two hashing algorithms that can be used are: MD5 and SHA-1. The user can also choose which characters the attack should try when using brute force attacks in the char set section.

For the dictionary attack a dictionary file is read in at the start of the program then every word in that dictionary is checked against the user's password. The dictionary file from WinEDT that is used in the program contains 118, 619 words [17]. Free dictionary files are abundant on the web. Outpost9 [18] and CyberWarzone [19] are examples of websites that list dictionary files with common words, names, and even different languages. Depending on which hash is chosen the word being evaluated will run through the hash and be checked against the user's entered password that has already been hashed. This is to simulate a real dictionary attack is it works in a similar manner. This uses a single *foreach* loop running through all of the words in the dictionary and thus runs in time $O(n)$ where n is the number of words in the dictionary. This

crack should finish within a second on most platforms. It is also an effective crack as dictionary words are often used as user's passwords [15]. In addition, password length is irrelevant for dictionary attacks as the whole word is checked at once against the password and won't affect the time complexity of this algorithm. Randomly generated passwords like "uQSHxgE5" are not found in the dictionary, rendering this type of attack useless.

The brute force attack works by trying every possible combination of the user's selected char set up to 8 characters. Clearly this is a much slower approach than the dictionary attack but a more thorough one as it checks every possible permutation of the user's selected char set. The time complexity of the algorithm I used is $O(m^n)$ where m is the number of characters in the char set and n is the length of the password. From the time complexity it is easy to see that the running time increases dramatically for every character that the length is increased, much more so than increasing the size of the char set. It is difficult to avoid this worst case time complexity for any implementation but fortunately it is easy to optimize with parallelization, in particular with help from the GPU. In addition, rule-based attacks can greatly improve the chances of the password being found quicker. Those are attacks where the brute force attempts are prioritized in the order of likelihood of success. This crack is immune to randomly generated passwords and will succeed given enough time and resources. It is particularly effective on shorter length passwords.

VII. Experimentation

Brute Force Data				
Password	**Length**	**Attempts/Hashes**	**MD5 Time (Seconds)**	**SHA-1 Time (Seconds)**
a	1	2	0.001	0.001
z	1	27	0.001	0.001
an	2	380	0.003	0.003
or	2	502	0.004	0.005
a1	2	1038	0.011	0.021
z9	2	1359	0.012	0.017
aA1	3	216029	1.705	1.714
zZ9	3	249381	2.12	1.966
aZ16	4	14970377	114.995	119.043
16Za	4	460207	3.617	4.257
abcd	4	80975	0.663	0.676
9999	4	14641	0.109	0.125
fast	4	407545	3.138	3.283
Slow	4	3466988	25.294	29.198
apple	5	2905499	22.914	22.781
zebra	5	887355	6.493	7.301
fast1	5	53515623	394.668	440.193
Slow9	5	982495000	7310.119	8498.922
abcde	5	2738180	20.529	23.8
quick	5	5912046	45.004	49.023
pass1	5	53464980	419.319	454.406
abcdef	6	88831622	672.138	747.115
aaaaaa	6	14900789	114.985	127.562
passwd	6	70006643	539.175	616.408
	Total:	1295527188	9697.017	11147.821
	Average(hashes/second):		**133600.5895**	**116213.4903**

Table 1.

Dictionary Data				
Password	**Length**	**Attempts/Hashes**	**MD5 Time (Seconds)**	**SHA-1 Time (Seconds)**
a	1	2	0.001	0.001
I	1	1	0.001	0.001
an	2	7	0.001	0.001
or	2	65	0.001	0.001
and	3	124	0.001	0.001
try	3	763	0.008	0.006
test	4	3471	0.026	0.028
pass	4	2791	0.021	0.024
apple	5	4122	0.031	0.034
zebra	5	9967	0.073	0.078
kitten	6	14711	0.108	0.115
hacker	6	13901	0.1	0.136
balloon	7	21030	0.152	0.164
puppies	7	300018	0.214	0.263
password	8	44857	0.321	0.341
computer	8	37618	0.303	0.29
	Total:	453448	1.362	1.484
	Average(hashes/second):		**332928.047**	**305557.9515**

Table 2.

In order to check the efficiency of the program, tests cases were run to see how fast a user's password could be cracked. Both methods, dictionary and brute-force, were used against several different passwords of varying length and complexity. Both of the hashing algorithms were used with each password cracking method. The tests were run on a Windows 8.1 desktop machine with an Intel i7 3770k processor and a Nvidia 760GTX video card, although the GPU is not used and all calculations are done in the CPU.

As expected, MD5 hashes were generated faster than SHA-1 hashes. They were generated about 15% faster when applied using a brute-force attack. Although the hashes were computed slowly overall with 133,600.5895 h/s (hashes per second) for MD5 and 116213.4903 h/s for SHA-1. Hashcat, for example, hashes MD5 at a rate of 2,753 Mh/s (millions of hashes

per second) and SHA-1 at a rate of 655 Mh/s on a Windows 7 machine [16]. This is mainly due to Hashcat's use of multi-threading, rule based attacks, and utilization of the GPU.

When the dictionary attack was used things changed a bit but the data above is more accurate as the time was drawn out over more attempts allowing for a better average. MD5 only performed 9% better than SHA-1 when a password was found using the dictionary attack. Tries per second was much higher, at a rate of 332,928 h/s for MD5 and 305,557 h/s for SHA-1. The reason for the speed increase is due to the implementation. All of the dictionary words are loaded into memory upon the application's launch. Therefore the attempts can be hashed much quicker than a word that has to first be generated first, as this operation takes time. While length did seem to affect the time of a successful crack this is only because longer length words are arranged toward the end of the particular dictionary file used. Dictionary files can be arranged in different ways and no correlation may exist at all between length and the time span of the crack.

VIII. Conclusion

Password cracking is a very real threat. There are numerous methods that can be used in an attack that have been described. Ways to prevent successful attacks by hackers for use by both users and administrators have been discussed. An implementation of two hashing algorithms was developed to be tested for efficiency and their results have been reported.

References

[1] Infosecinstitute, http://resources.infosecinstitute.com/10-popular-password-cracking-tools/, last accessed on 9 March 2015.

[2] Infosecinstitute, http://resources.infosecinstitute.com/password-cracking-evolution/, last accessed on 9 March 2015.

[3] Hashcat, https://hashcat.net/hashcat, last accessed on 9 March 2015.

[4] CrackStation, https://crackstation.net/hashing-security.htm, last accessed on 9 March 2015.

[5] CloudFlare, http://blog.cloudflare.com/inside-shellshock/, last accessed on 9 March 2015.

[6] Start Hacking, http://www.starthacking.net/google-dorks/, last accessed on 9 March 2015.

[7] Openwall, http://www.openwall.com/john/, last accessed on 9 March 2015.

[8] Speedtools, http://www.speedtools.org/registry-security-1.php, last accessed on 9 March 2015.

[9] RainbowCrack Project, http://project-rainbowcrack.com/, last accessed on 9 March 2015.

[10] RainbowCrack Project, http://project-rainbowcrack.com/generate.htm, last accessed on 9 March 2015.

[11] L0phtCrack, http://www.l0phtcrack.com/learn.html, last accessed on 9 March 2015.

[12] Aircrack-NG, http://www.aircrack-ng.org/, last accessed on 9 March 2015.

[13] EMC, http://www.emc.com/emc-plus/rsa-labs/historical/rsa-security-response-weaknesses-algorithm-rc4.htm, last accessed on 9 March 2015.

[14] WindowsITPro, http://windowsitpro.com/security/prevent-password-cracking, last accessed on 9 March 2015.

[15] Gizmodo, http://gizmodo.com/the-25-most-popular-passwords-of-2014-were-all-doomed-1680596951, last accessed on 9 March 2015.

[16] Hashcat, http://hashcat.net/oclhashcat/, last accessed on 17 March 2015.

[17] WinEdt, http://www.winedt.org/Dict/, last accessed on 20 March 2015.

[18] Outpost9, http://www.outpost9.com/files/WordLists.html, last accessed on 22 March 2015.

[19] CyberWarzone, http://cyberwarzone.com/massive-collection-password-wordlists-recover-lost-password/, last accessed on 22 March 2015