Password Security - When Passwords are there for the World to see

Eleanore Young
Offense Department, scip AG
elyo@scip.ch
https://www.scip.ch

Marc Ruef (Editor)
Research Department, scip AG
maru@scip.ch
https://www.scip.ch

Keywords: Bitcoin, Exchange, GitHub, Hashcat, Leak, OWASP, Password, Policy, Rapid, Storage

1. Preface

This paper was written in 2017 as part of a research project at scip AG, Switzerland. It was initially published online at *https://www.scip.ch/en/?labs.20170112* and is available in English and German. Providing our clients with innovative research for the information technology of the future is an essential part of our company culture.

2. Introduction

The year 2016 has seen many reveals of successful attacks on user account databases; the most notable cases being the attacks on *Yahoo* [1] and *Dropbox* [2]. Thanks to recent advances not only in graphics processing hardware (GPUs), but also in password cracking software, it has become dangerously cheap to determine the actual passwords from such password databases. We will explore the setup of the appropriate hardware, common strategies used for cracking passwords, and how to make it harder for attackers to determine your passwords.

3. A Little Background

Due to the fact that the past 15 years or so have seen many successful attacks on password databases, software developers have learned not to store the user account passwords in plain text, but to only store so called *digests*, or *hashes*, of the passwords.

Hashes are the output of one-way algorithms, which means that it is computationally *very hard* to determine the input from the output. These methods allow developers to verify the authenticity of a password without actually storing the password itself. Commonly used hashing algorithms are: MD5, SHA-1, SHA-256, SHA-512, NTLM, or combinations thereof. The Wikipedia articles on *cryptographic hash functions* [3] and *key derivation functions* [4] offer a good overview over the subject, but I particularly like the introductions by *Zulfikar Ramzan* [5] and *Srinivas Devadas* [6].

All is not well though, as not all hashing algorithms were designed for securely hashing passwords, and many have vulnerabilities such as frequent collisions (when two different input values have the same hash), or more efficient ways to calculate the hashes, in other words, shortcuts. Modern password cracking software use these vulnerabilities to very rapidly determine the original password from a hash without having to attempt a reversal of the hashing algorithm.

Furthermore, if passwords are fed through hashing algorithms as is, two persons who happen to use the *same password*, will also have the *same hash value*. As a countermeasure, developers have started adding random user-specific values (the salt) to the password before calculating the hash. The salt will then be stored alongside the password hash in the user account database. As such, even if two persons use the same password, their resulting hash value will be different due to the added salt.

Modern GPU architectures are designed for large scale parallelism. Currently, a decent consumer-grade graphics card is capable of performing on the order of 1000 calculations simultaneously. For example, when using two recent top-range consumer graphics cards to perform an attack on NTLM hashes, this would result in approximately 27 Trillion hashing operations per second! In 2012, the developer team *MOSIX* [7] first published GPU clustering software which allows for practically unlimited scaling of the underlying hardware with minimal effort. Thus, it has become possible to reverse even password hashes that employ a salt at within reasonable time frames, assuming the hashing algorithms are *amenable to parallelization* [8].

Attackers commonly use a combination of multiple password cracking programs, most of them freely available. As of the writing of this article, free and open source programs that are capable of using graphics cards are: *John the Ripper* [9] and *Hashcat* [10]. Others are *Ophcrack* [11], and *Cain and Abel* [12]. There exist more, but usually for specific areas such as computer passwords or WiFi passwords.

4. Strategies of Attackers

People who profit from cracking passwords have steadily progressed towards increasingly professionalized work. The strategies presented herein present only a small part of the effort undertaken to determine user credentials, and are by no means complete.

As far as hardware is concerned, attackers will adapt it to their use case, ranging from anything between a mid-range laptop to a 128 GPU cluster. Due to the fact that modern GPUs are now reasonably cheap, even one person can easily come up with the money to build a powerful cracking computer.

For the sake of brevity, we will focus on a powerful desktop computer with a recent release of Hashcat. The program Hashcat provides various attack methods and supports a large number of hashing algorithms, always automatically making use of the available CPUs and GPUs. I believe that it manages to illustrate how effectively passwords can be cracked nowadays.

For the examples in the following sections, we will try to crack the passwords of six persons. Persons "user1" and "user2" use two of the most common passwords, "user3" and "user4" use two short randomly generated passwords, and "user5" and "user6" use a very common easy-to-remember format. The hashing algorithm used is SHA-1 without a salt.

1. *user1* has the password *hallo1* and the corresponding hash of `d819b82566e9b601d87e168d0dffe31cee1a9229`.
2. *user2* has the password *1234567890* and the corresponding hash of `01b307acba4f54f55aafc33bb06bbbf6ca803e9a`.
3. *user3* has the password *Xkkxer* and the corresponding hash of `a357d8269ef8f96973a98c820b2fb47251f11696`.
4. *user4* has the password *Lf!lyASz* and the corresponding hash of `338cb709d331e5bb6361300f59fcea03bec56111`.
5. *user5* has the password *Martin90?* and the corresponding hash of `1c15c4f9a0da91443205f58f5731a3c850c34b6d`.
6. *user6* has the password *Bergsteiger2016!* and the corresponding hash of `72bf66db2315e75dfb569ba3d3a09203e906c111`.

Hashcat parses the above list from the text file `hashes.txt` in the format `username:hash-value`.

5. Dictionary Attacks

Dictionary attacks rely on word lists and rule sets to derive candidate passwords, and are usually the fastest to deliver results. Based on our experience, dictionary attacks tend to recover between 10% and 30% of the available passwords. They work, because people prefer to use common words with simple variations for passwords.

The attacker's all-time favorite word list is based on the public *Rockyou* [13] user account database leak from 2009, because it contains a very large number of real, primarily English-based passwords. However, we found that we get better coverage when including German-language dictionaries, or targeted word lists for dictionary attacks.

Dictionary rules are used to modify each word in the dictionary to capture password variations such as *hallo90!* or *Hallo*. The *GitHub* [14] source repository of Hashcat already provides a good set of dictionary rules.

The following command manages to recover the passwords of *user1*, *user2*, and *user5* within the first couple of minutes, but fails to recover any of the other user's passwords.

```
$ hashcat -a 0 -m 100 -w 3 –username -r
InsidePro-PasswordsPro.rule hashes.txt
wordlists-dir/
```

6. Mask Attacks

Despite all that has happened with regards to password security over the past years, a large percentage of users still use passwords that consist of a capitalized word, a few digits and a special character, in that order. They are usually *between 8 and 10 characters long* [15]. Coincidentally, these passwords fulfill the minimum password requirements commonly used in Windows domains.

User-chosen passwords will follow a pattern. In Hashcat, these patterns can be expressed as so-called masks, for example `?u?l?l?l?l?d?d?s`. In a mask attack, Hashcat will perform a brute force attack (i.e. try out every possible value) based on the supplied masks, which is more efficient than a blunt brute force attack alone.

The GitHub source repository of Hashcat already provides good masks, but for non-English language or corporate environments, it is often helpful to create masks from statistical information about passwords or from a known password policy, such as the default Windows domain policy. The *PACK* [16] utilities were expressly developed for this purpose and are of great help in these scenarios. In certain cases it is helpful to evaluate existing sets of masks against a password policy to further reduce the number of brute force tries Hashcat has to make, so I created package *Policymatch* [17] to help with that.

The following command additionally recovers the password of *user3* within a minute.

```
$ hashcat -a 3 -m 100 -w 3 –username
hashes.txt rockyou-1-60.hcmask
```

Recovering the password of *user4* took a bit longer, about 72 hours:

```
$ hashcat -a 3 -m 100 -w 3 –username
hashes.txt 8char-1l-1u-1d-1s-compliant.hcmask
```

7. Hybrid Attacks

Hashcat allows us to combine the two previous attack methods with the hybrid attack method. The following command finally manages to recover the last password, that of *user6* in about 15 minutes. This specific attack works well, because users often include a date or a year in the password.

```
$ hashcat -a 6 -m 100 -w 3 –username
hashes.txt wordlists-dir/ "?d?d?d?d?s"
```

8. Closing the Blinds on Prying Eyes

As a form of proof of *who you are*, passwords belong to the group of *what you alone know*. But because passwords are so hard to remember, they do not actually serve as a secure form of authentication. We see from the above examples that even passwords that are commonly considered more secure, can be recovered with little effort. A very scary realization.

From the user perspective, the best approach to passwords as a form of authentication is to use password managers, such as *KeePassX* [18], *LastPass* [19], or *1Password* [20]. The most important aspect of password managers is that they allow users to generate very complex passwords that never have to be remembered or entered by hand. In my opinion, the talks by *Pierre-Antoine Haidar-Bachminska* [21] and *David Jaeger* [22] offers a perfect introduction to better password strategies for those inclined.

From the developer perspective, it is crucial that passwords are stored and handled in a secure fashion using algorithms specifically designed for authentication data. For the basics, read the OWASP cheat sheets on *password storage* [23], *forgot password* [24], *authentication* [25] and *pinning* [26]. If you're interested in knowing more about the subject, watch the talk by *Per Thorsheim* [27].

9. External Links

[1] https://www.wired.com/2016/12/yahoo-hack-billion-users/
[2] https://dropbox.thecthulhu.com/
[3] https://en.wikipedia.org/wiki/Cryptographic_hash_function
[4] https://en.wikipedia.org/wiki/Key_derivation_function
[5] https://www.khanacademy.org/economics-finance-domain/core-finance/money-and-banking/bitcoin/v/bitcoin-cryptographic-hash-function
[6] https://www.youtube-nocookie.com/watch?v=KqqOXndnvic
[7] http://www.mosix.cs.huji.ac.il/txt_vcl.html
[8] http://security.stackexchange.com/questions/4781/do-any-security-experts-recommend-bcrypt-for-password-storage#6415
[9] http://www.openwall.com/john/
[10] https://hashcat.net/hashcat/
[11] http://ophcrack.sourceforge.net/
[12] http://www.oxid.it/cain.html
[13] http://downloads.skullsecurity.org/passwords/rockyou.txt.bz2
[14] https://github.com/hashcat/hashcat
[15] https://www.scip.ch/en/?labs.20100709
[16] https://thesprawl.org/projects/pack/
[17] https://github.com/youngec/policymatch
[18] https://www.keepassx.org/
[19] https://www.lastpass.com/
[20] https://1password.com/
[21] https://www.youtube-nocookie.com/watch?v=KabAryo6tIA
[22] https://www.youtube-nocookie.com/watch?v=qdycF4j3Ux0
[23] https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet
[24] https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet
[25] https://www.owasp.org/index.php/Authentication_Cheat_Sheet
[26] https://www.owasp.org/index.php/Pinning_Cheat_Sheet
[27] https://www.youtube-nocookie.com/watch?v=dc-bF2CU0Xo