Global Information Assurance Certification Paper

The Dangers of Weak Hashes

GIAC GWEB Gold Certification

Author: Kelly Brown, kbrown@aboutweb.com
Advisor: Robert Vandenbrink

Accepted: November 15, 2013

Abstract

There have been several high publicity password leaks over the past year including LinkedIn, Yahoo, and eHarmony. While you never want to have vulnerabilities that allow hackers to get access to your password hashes, you also want to make sure that if the hashes are compromised it is not easy for hackers to generate passwords from the hashes. As these leaks have demonstrated, large companies are using weak hashing mechanisms that make it easy to crack user passwords. In this paper I will discuss the basics of password hashing, look at password cracking software and hardware, and discuss best practices for using hashes securely.

1. Introduction

In June of 2012 a hacker posted more than 8 million passwords to the internet belonging to LinkedIn and eHarmony (Goodin, 2012). Within hours, over two million of the passwords were cracked and posted on-line. Within a week, 99% of the passwords had been cracked. The LinkedIn passwords were using the SHA-1 algorithm without a salt making them particularly easy to crack. The eHarmony passwords were also stored using poor cryptographic practices as unsalted MD5 hashes. A month later, 450,000 were leaked from Yahoo (Gross, 2012). In this case, passwords were stored in clear text. These incidents are just a sampling a of the poor password storage techniques currently in place on major Web sites. While the security flaws that allowed these passwords to be leaked in the first place are a serious security concern, the lack of best practices in password storage exasperated the situation.

Implementing a few best practices in password storage will minimize the potential damage caused by a password leak. The following technicques should be used:

- Store passwords as hashes using strong encryption algorithms
- Salt the hashes
- Employ key stretching or slow algorithms to increase password cracking time
- Encrypt the password hashes

Kelly Brown, kbrown@aboutweb.com

2. Hashing

2.1. Understanding Hashes

The commonly used term "hash" refers to a one-way hash function which is a mathematical formula that takes an arbitrary length message and returns a fixed length value. A hash formula is represented as h=H(M), where h is the hash, H is the hashing function, and M is the message. The hash formula has several desirable mathematical characteristics:

> Given M, it is easy to compute h.
> Given h, it is hard to compute M such that H(M)=h.
> Given M, it is hard to find another message, M', such that H(M)=H(M')
> (Schneier, 1996)

We want a hashing function where it is easy to compute the hash value, where it is hard to reverse the computed hash values back into the original message, and it is hard to find two inputs that generate the same hash value. As an example consider two values generated by the MD5 hash. The hash value for "password" is "5f4dcc3b5aa765d61d8327deb882cf99" and the hash value for "passwore" is "a826176c6495c5116189db91770e20ce". The computer representation of these two strings is very similar, there is only a one bit difference, yet the hash values are significantly different. There is also no way to take the computed hash value and determine the word we started with other than randomly guessing words until you find one that produces the same hash value.

Hash functions typically genenerate a hash value of a specific bit length. This is usally 160 bits or more. A hash value that is too short will have a high collision rate where many messages result in the same hash value, so hash sizes should be large enough to accommodate a large number of values. Hash collisions are unavoidable. Very large messages, sometimes entire documents, are reduced to a smaller value of a fixed size so there are times when different messages will produce the same value. However, a good hashing algorithm makes it difficult to forcibly generate two messages that have the same hash value.

While the sample MD5 hash values above appear to have little in common, MD5 is no longer considered to be a good hash algorithm because weaknesses have been found in the algorithm. Researchers have been able to manipulate messages so that two different messages

result in the same hash. These weaknesses were first reported in 1996 and security experts started suggesting that MD5 be replaced with stronger algorithms (Dobbertin, 1996).

So how do you know if an hash algorithm is good? The National Institute of Standards and Technology (NIST) publishes official standards for hash functions called the *Secure Hash Standard* (SHS). The current version of the this document is FIPS PUB 180-4 (National Institute of Standards and Technology, 2012). This publication defines several hash algorithms including: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256. The hashs after SHA-1 are collectively known as SHA-2 and are part of a newer specification. These algorithms undergo rigorous testing to ensure they are cyptographically strong. NIST Special Publication 800-107, Revision 1, *Recommendation for Applications Using Approved Hash Algorithms* (National Institute of Standards and Technology, 2012) recommends phasing out the use SHA-1 because of recent weaknesses found in the algorithm. Weakness may one day be found in the SHA-2 algorithms; SHA-3 is already in the works but has not been finalized yet. Cryptography is a constantly advancing science. As computing power increases and weakness are found in the current algorithms, a new generation of stronger algorithms takes their place and the cycle repeats.

The NIST standards are not the end all be all of hashing algorithms. There are several other hasing algorithms such as RIPEMD, Tiger, and SWIFFT that have been extensivly tested by security researchers. However the NIST algorithms are generally considered to be secure and are required for government related work.

2.1.1. The Psychology of Password Selection

In December of 2009 a social gaming site RockYou.com was hacked and 32 million passwords were exposed (Signler, 2009). An analysis of the passwords (Imperva, 2010) revealed several trends how users select passwords. People tend to use short passwords; 30% of the passwords were six characters or less and over 50% where eight characters or less. People tend to use a limited set of characters for passwords; 40% of people choose passwords consisting only of lower case letters, 16% of people used only numeric characters in their passwords, and less than 4% of people used special characters. People use common words for their passwords; 50% of people choose slang, dictionary words, or trivial passwords consisting of adjacent letters,

numbers, or simple patterns for their passwords such as a word followed by a one or more numbers. The most common password was "123456". Of the 32 million accounts leaked, there were only 14.5 million unique passwords meaning there was a lot of duplication of passwords. The "123456" password for example, was used by over 290,000 different accounts.

The RockYou password leak was a critical turning in password cracking. Not only did this provide an extensive list of commonly used passwords, it provided an insight in the psychology of password selection and allowed password crackers to limit the scope of their attacks which decreased the time and increased the likely hood of cracking passwords. This trend continues today, each new password leak is added to word lists and reveals more about the psychology of password selection.

2.2. Password Cracking

While there are many attacks against password protection systems, password cracking refers to the process of extracting passwords from data. This data is typically the hash values of passwords. As we have seen, this data is often leaked by web sites but can also be obtained by gaining physical access to an operating system.

The most straightforward attack is called the brute-force attack. This is simply trying every possible combination of characters until you find a matching hash value. This can become very time consuming particularly with long passwords. The difficulty also increases when more characters are allowed in passwords. Table 1 shows the number of possible combinations using the length of the password and different character combinations. The numbers grow quite large as the length and complexity of the password increases and this would seem to make password cracking impossible. While it's true that the length of time to brute-force passwords increases with complexity, there are several other techniques that crackers can use to expose these passwords.

Characters in Password	Lower Case Letters	Lower and Upper Case	Full ASCII
1	26	52	256
3	17,576	140,608	16,777,216

5	11,881,376	$3.8*10^{8}$	$1.1*10^{12}$
8	$2*10^{11}$	$5.3*10^{13}$	$1.8*10^{19}$
10	$1.4*10^{14}$	$1.4*10^{17}$	$1.2*10^{24}$
15	$1.7*10^{17}$	$5.4*10^{25}$	$1.3*10^{36}$
20	$2.0*10^{28}$	$2.1*10^{34}$	$1.4*10^{48}$

Table 1: Number of Combinations based on Password Length and Scope

Simple brute force attacks become inefficient for long complex passwords, because every possible combination of letters and characters must be tried. However, most people do not use random passwords. They use words and names they are familiar with and this is where dictionary attacks come into play. Instead of trying random series of characters, a dictionary attack uses a list of words to match hashes. Originally this was a list of all the words in the dictionary, but these lists have been expanded over time. The RockYou attack in particular revealed millions of commonly used passwords and has become part of the standard dictionary used to crack passwords.

2.2.1. Rainbow Tables

In 1980 a paper called "A Cryptanalytic Time – Memory Trade-Off" was published in the IEEE Transaction on Information Theory (Hellman, 1980). This paper introduced the concept of precomputing cyptologic function values and saving them so they can be used to quickly look up values. This technique works particuarlly well for hashes and lists of precomputed hashes became known as "rainbow tables."

There are many rainbow tables available on-line and most password cracking tools come with a set of rainbow tables. Any password that has ever been leaked already exists in these rainbow tables which is what makes password reuse particularly dangerous. Rainbow tables allow near instaneous cracking of a password since the hash value has already been calcualted and simply needs to be looked up. For example, using the sample rainbow table in Table 2 and a hash value of "a61a78e492ee60c63ed8f2bb3a6a0072" you can quickly calculate the corresponding password is "pa$$word". Real hash tables consist of a large number of precalculated values.

5f4dcc3b5aa765d61d8327deb882cf99	password
bed128365216c019988915ed3add75fb	passw0rd
90f2c9c53f66540e67349e0ab83d8cd0	p@ssword
a61a78e492ee60c63ed8f2bb3a6a0072	pa$$word
b7463760284fd06773ac2a48e29b0acf	p@$$w0rd
482c811da5d5b4bc6d497ffa98491e38	password123

Table 2: Sample MD5 Rainbow Table

Ideally, all rainbow tables would be complete, that is they would have a password for every possible hash value. In practice, rainbow tables are rarely complete especially for large hash values. The SHA-256 algorithm for instance has a 256 bit hash length which means there are $1.16*10^{77}$ hash values. Not only would a complete hash table be extremely large, but generating such a table would take a very long time based on current computation speeds. Just generating the hashes for that many values would take a long time, but because of hash collisions there would be gaps in the table and due to the nature of the hashing function, there is no way to reverse engineer the gaps except to keep trying new combinations until every gap is filled. However, nearly complete rainbow tables with over 99% of possible hash values are available for NTLM, MD5, and SHA1 on-line (RainbowCrack Project, 2013). There are also complete rainbow tables for hashing algorithms such as the LM hashes used by older NT versions of windows including Windows 2000.

While there are many rainbow tables available online, one of the most interesting and most over looked ones is Google. Because rainbow tables are typically stored as plain text and made available on Web sites, they are indexed by Google. For example the MD5 hash for "password" is "5f4dcc3b5aa765d61d8327deb882cf99". As can be seen in Figure 1, searching for this hash value through Google returns over 19 thousand results.

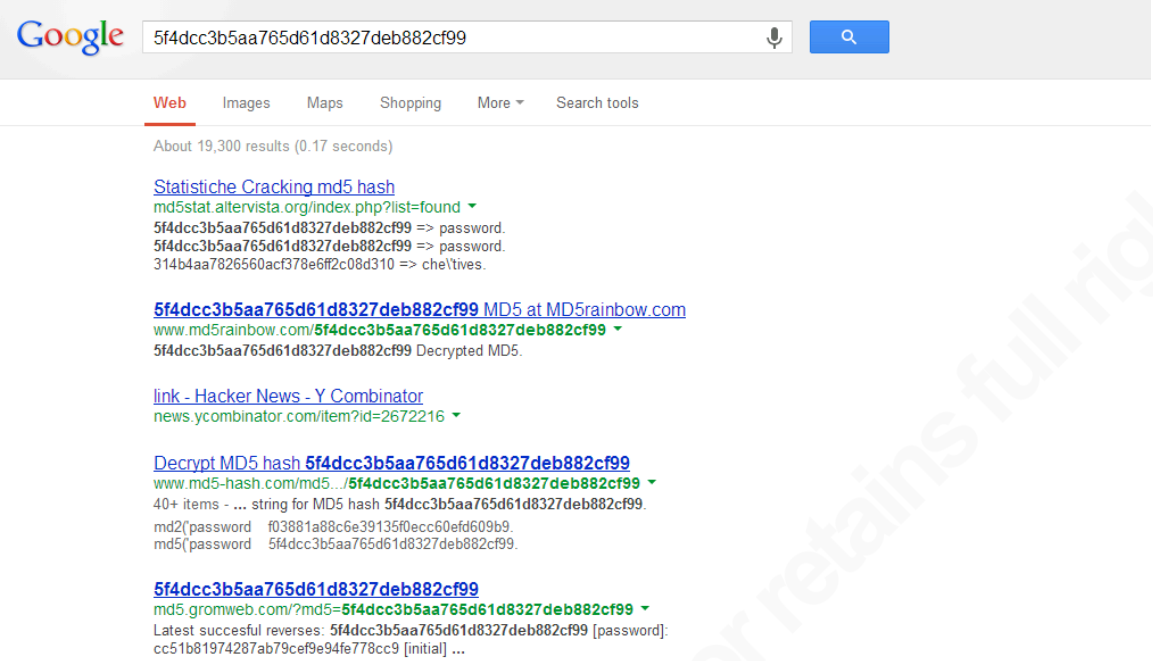

Figure 1: Google search for MD4 Hash of "password"

2.2.2. Cracking Software

There are many software applications that have the ability to crack passwords. Many of these are open source and often used by both white hat hackers (to identify security weaknesses in passwords) and by actual hackers who are looking to break into systems. A few of the more popular software packages are:

Cain and Abel (Oxid.it, 2013) is a closed source application that promotes itself is a password recovery tool for the Microsoft Operating System. It has password cracking capability, but also has many options for extracting passwords from the file system and network traffic.

John the Ripper (Openwall, 2013) is open source password cracker that was originally developed to detect weak UNIX passwords. However it has expanded over time and now has support for many types of UNIX and Windows based passwords.

Hashcat (Hashcat, 2013) is one devoted solely to cracking hashes. It claims to be the fastest md5, phpass, mscash2, and WPA/WPA2 cracker and performance tests back up these claims.

All full-featured password crackers support a common set of functionality. They can brute force passwords using a specified set of characters. For instance, if a site only allows

upper case, lower case, and numerical values in passwords there is no point in including special characters. The goal is to minimize the scope of the combinations that need to be tested. Password crackers allow you to specify a minimum and maximum password length. If you know a site requires at least 8 characters in the password there is no point in trying shorter values. The maximum value is there to provide an upper limit to reduce the time spent trying to crack the password. This value is somewhat dependent on the speed of the cracking software and hardware and the maximum time you are willing to allow the cracker to run.

Password crackers also support dictionary attacks which are much faster than brute force attacks. They allow you to specify a file of words to compare the password to. A popular word list is the RockYou word list, though there are larger wordlists that have compiled from all known leaked passwords. Most advanced password word crackers also support the concept of rules. This is a combination attack using a word list with a limited brute force attack that makes modifications to the words in the list based on a set of rules. For instance, "password" has consistently shown up as one of the most popular passwords, but variations of it are also popular. A rule may specify that the letter "a" in the word list should be replaced with a "@" sign and the letter "s" by a "$" sign. Using this rule a password list containing the word "password" would also match "p@ssword", "pa$$word", "p@$$word", etc. This is a very powerful technique, but rules also increase the number of the combinations that are checked. Too many rules can become almost as slow as a brute force approach. Most password crackers come with a set of sample rules and there are also rule sets available on-line based on the analysis of the many password leaks that have occurred.

Password cracking is typically done in stages of efficiency removing identified hashes from the list after the corresponding password is found. This reduces the size of the list, so the more complex and time consuming approaches are run against fewer hashes. Running a brute force attack against a list of passwords is very time consuming, so the first pass at a password list is typically a straight dictionary attack. This is usually quite effective (unfortunately) and can usually identify a significant number of passwords in a list. The next pass is typically a dictionary attack with a rule set which is then followed by a limited brute force attack. At this point the list is usually a small fraction of the original list and more intensive techniques can be tried such as a dictionary attack with a larger rule set or a more comprehensive brute force attack.

Cracking software is being continuously improved and optimized for performance. In December 2012 at the Passsword^12 conference, Jen Steube announced an improvement to the SHA1 hashing calculation algorithm that increased the speed by 21% (Steube, 2012). Jen Steube is the developer of the Hashcat and this improvement has been incorporated into the Hashcat application. As these algorithms are optimized, it allows cracking to happen faster and provides the opportunity for more complex attacks.

Figure 2 shows a sample Hashcat attack. In this example I used the following command: *cudaHashcat-plus64.exe -m 0 7kmd5.txt rockyou.txt*. The command line executable program is cudaHashcat-plus64.exe. The cuda stands for the CUDA (NVIDIA, 2013) parallel computing platform used by NVIDIA graphic cards that allows Hashcat to use the GPU. The "-m 0" command line options specifies the MD5 hash. The "7kmd5.txt" file is source file of hashes and the "rockyou.txt" file is a dictionary file. This was a straight dictionary attack and, as can be seen in the output, 28.85% of the 6871 hashes were cracked in a total of 7 seconds.

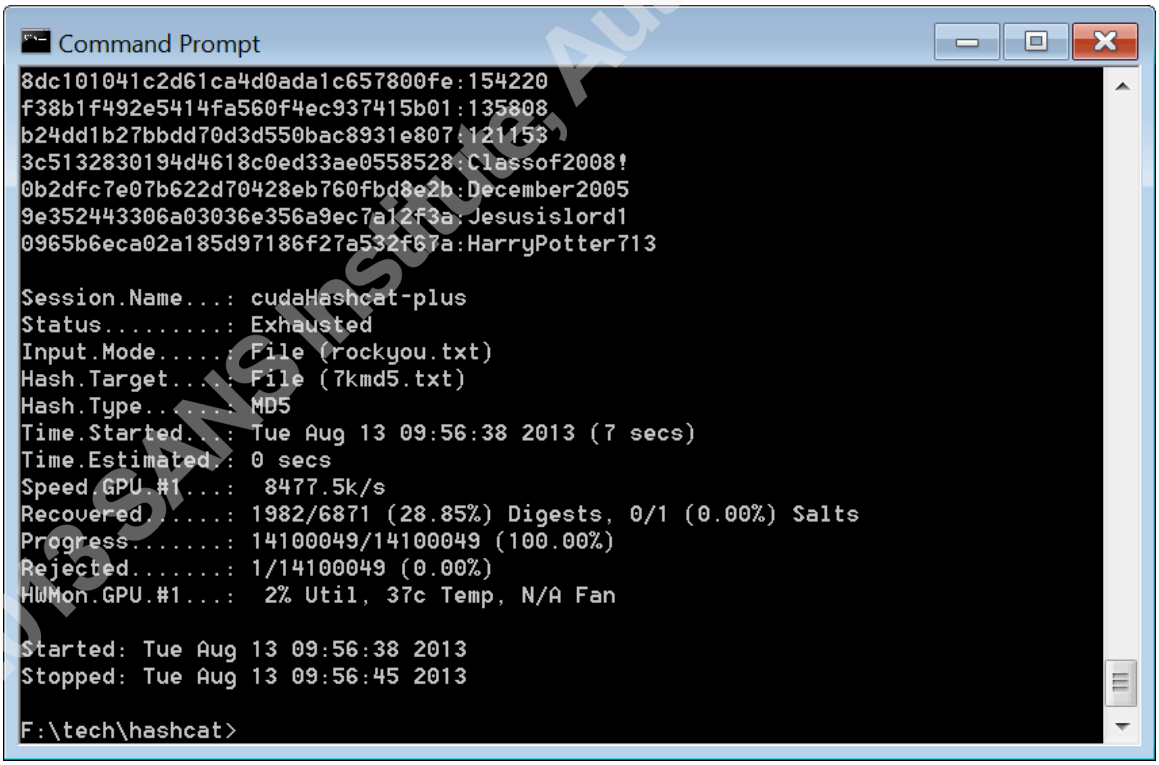

Figure 2: Sample Hashcat Attack

Figure 3 shows the same crack run again, but using a rule set file. As you can see in the figure, nearly twice as many passwords (44.74%) were recovered using a rule file as opposed to a straight dictionary attack. The second attempt also took twice as long to run due to the increased complexity of using a rule set. However, the rules actually generated 64 times as many hash codes. You might expect it take 64 times as long to run, but the second run makes extensive use of the GPU, 60% versus only 2% in the first run. The much faster GPU of the graphic card is used by the Hashcat program when you start applying different rules to the word list.

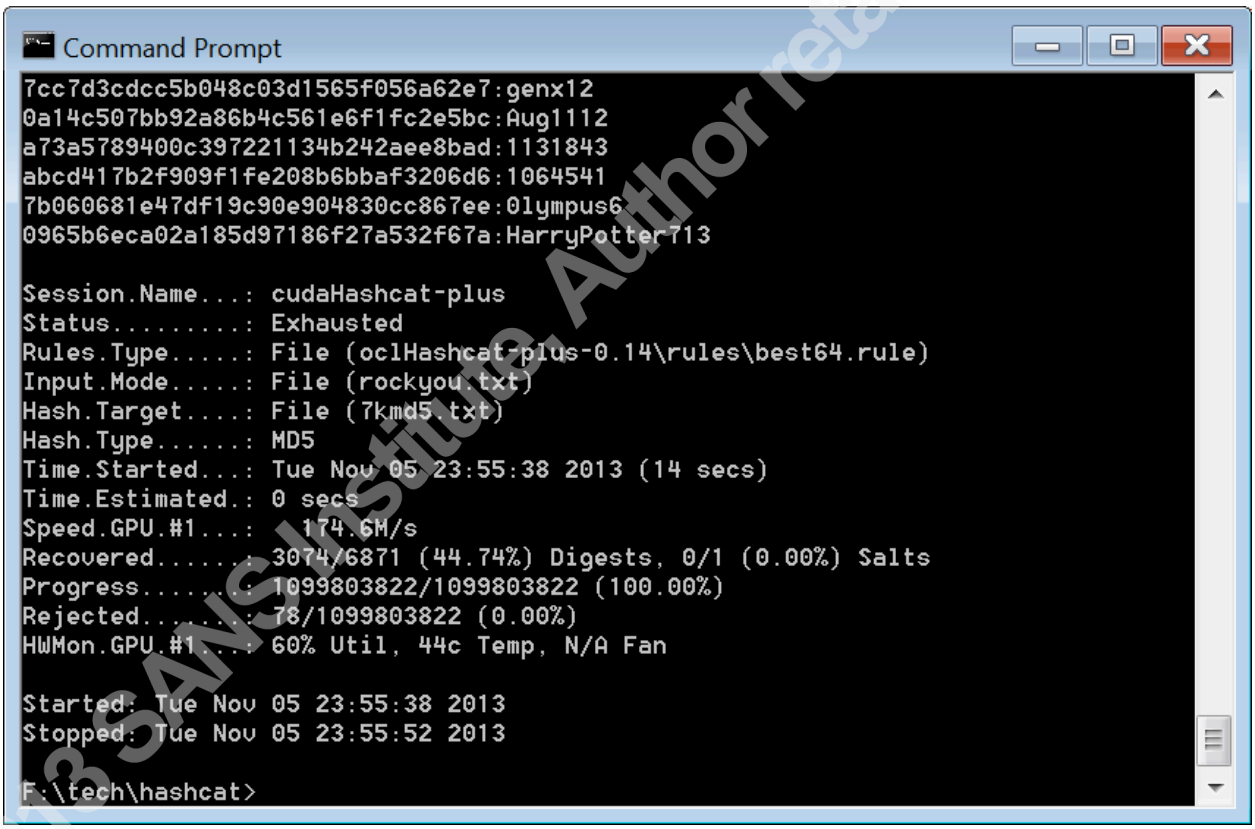

Figure 3: Hashcat Using Rule Set

The specific rule set used in this scenario comes with Hashcat and is called the best64 rule set. It is a list of rules developed by the Hashcat community that have been found to be effective in cracking passwords. An excerpt from the file of the first few rules looks like this:

```
## nothing, reverse, case... base stuff
:
r
```

Kelly Brown, kbrown@aboutweb.com

```
u
T0

## simple number append
$0
$1
$2
$3
$4
$5
$6
$7
$8
$9
```

Understanding what these rules do will give you some insight into the types of rules password crackers use. The first command ";" means do nothing or just use each word as it appears. The second command "r" stands for reverse and tries the word backwards. The third command "u" converts the word to upper case. The fourth command "T0" toggles the case of the first character. The next series of commands "$0" through "$9" tries appending numbers to the word. Applying these rules to the word "password" generates the following sequence of words: password, drowssap, PASSWORD, Password, password0, password1, password2, password3, password4, password5, password6, password7, password8, password9.

2.2.3. Hardware

While specialized hardware has been built for security purposes it is generally an expensive proposition limited to those with large budgets such as governments. This changed in the 2007 when NVIDIA released the CUDA development kit with their G80 processor (Li, 2010). The CUDA development kit allowed developers to harness the power of the Graphics Processing Unit (GPU) for applications other than simply graphics. Unlike the Central Processing Unit (CPU) which is the core of the general purpose computer, GPUs have limited capabilities. They are designed to process graphic commands which are highly parallelized and follow regular patterns. While the capabilities of GPUs are limited, they can process Cryptographic functions like hashes quickly and efficiently, making them useful tools for cracking passwords.

The new version of Hashcat known as oclHashcat-plus (Hashcat, 2013) has added support to use the graphic cards to increase the performance of hashing calculations. As an

example of the performance boost, my laptop running a 2.30 GHz Intel i7-3610QM core, can compute 55.2 million hashes per second using the regular version of Hashcat. The OCL version of Hashcat using my GeForce GTX 670M graphic card GPU can generate 547.2 million hashes per second, nearly a tenfold increase in performance. It's also important to note that this is a laptop with a mobile graphics card --desktop computers with full graphic cards see even better performance.

At the Password^12 conference in 2012, Jeremi Gosney presented a high performance clustered password cracker (Gosney, 2012). Figure 3 shows the configuration of one of five servers that made up the cluster which included a total of 25 graphic cards. The cluster used oclHashcat-plus to crack passwords. Using this cluster he was able to process 180 billion MD5 passwords per second. If we compare this to total combinations I calculated back in Table 1, this device is capable of brute forcing all 8 character lower case passwords in just over a second.

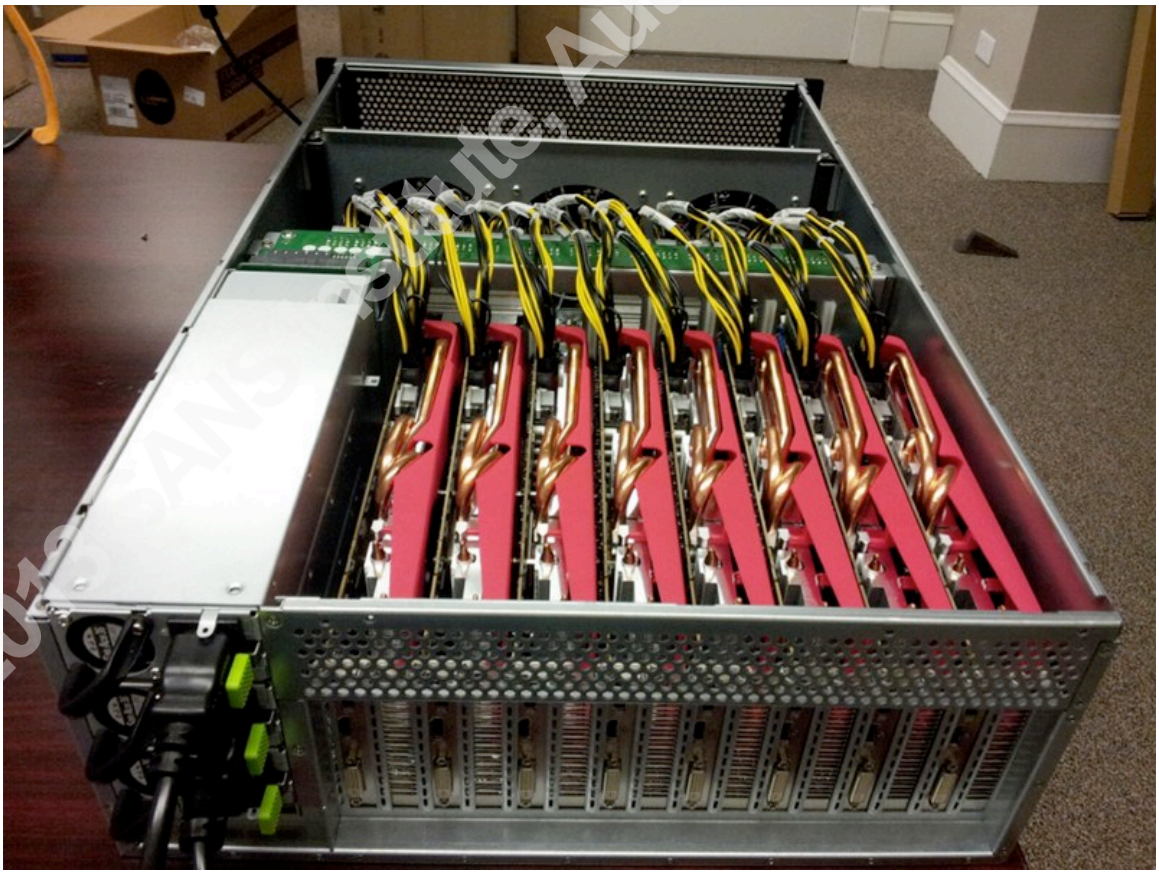

Figure 4: Password Cracking Machine (Gosney, 2012)

While Jeremi Gosney's machine is very fast, it is also a rather expensive proposition to the average person. However, thanks to the power of the cloud, this expense can be mitigated. According to Info Security (Info Security, 2010), a German hacker used an Amazon cloud instance to crack all six-character combinations of the SHA-1 hashing algorithm in 49 minutes for $2.10. The cloud gives a tremendous amount of power to users at very little cost putting high performance password cracking into the hands of the average user and reinforcing the need to ensure hashes are used securely.

2.2.4. Social Cracking

The LinkedIn password leak demonstrates the social aspect of password cracking. The passwords that were originally leaked were not the full set of passwords, but the ones the original hacker couldn't crack (Goodin, 2012). Less than two and half hours after the original list of 1.5 million hashes was posted on the InsidePro cracker forum, a user responded with 1.2 million of them cracked. This process continued with password crackers taking the remaining list and narrowing it down over time. This process highlights the social aspect of password cracking as the collective resources of thousands of users are applied to crack password lists. This represents a tremendous amount of collaborative computing resource and password cracking expertise which was able to crack the majority of the difficult LinkedIn passwords in a matter of hours.

2.3. Hashing Best Practices

Given the different types of attacks and vulnerabilities of hashes we've discussed, they may not seem like they are worth using. However, the alternatives are limited when passwords are used and it's certainly more secure than storing passwords as clear text. There are three primary techniques that are used to improve hash security: adding salt to the hash (which increases the time needed to compute the hash), and employing encryption on the hash values.

2.3.1. Salting Hashes

The concept of adding salt to a hash is pretty straight forward; an additional value is appended or prepended to the password before it is hashed. This prevents the use of rainbow tables and saving computed values to crack the passwords. While the implementation is straight forward, salt still needs to be implemented properly in order to be beneficial.

Use unique salt values. A different salt should be created for each password/user. If the same salt were used for all the passwords in your system a rainbow table could be created for the set of passwords and a cracker would only have to try each hash value once and compare to all the passwords used in the system. Using a unique salt for each password forces the attacker to run the cracking process for each password separately.

Use a large salt value. Most hash implementations support 32 or 64 bit salts. A small value such as one character/byte, limits the scope of the salt to only 256 values and would allow a cracker to generate a lookup table for each of the 256 values.

Randomly Generate salt values. Ensure the salt value is randomly generated across the range of values. A salt that is duplicated for many passwords suffers from the lookup table problem.

Salt values do not need to be hidden. The purpose of the salt is to add entropy to the hashed value. Attempting to hide the salt is considered security through obscurity which has been proven time and again to be a poor security practice.

2.3.2. Slowing Cracking

Because hash values cannot be calculated in reverse as designed, the general approach used in cracking passwords is to try a lot values until you find a match. Most hashing algorithms are quite fast and millions of comparisons can be computed per second. If it takes one second to compute a hash, it is not really noticeable to a user logging into a system, but to a password cracker one second to calculate each hash value can make password cracking impossible to accomplish in a reasonable amount of time.

Password-Based Key Derivation Function (PBKDF2) was originally proposed by RSA Laboratories as part of the Internet Engineering Task Force's RFC 2898 (RSA Laboratories, 2000). The basic idea is to iterate over a hashing function using the output of each iteration as the input for the next, this is also known as key stretching. So instead of simply using SHA1 to compute the hash once, you use it thousands of time. This concept is designed to scale as computers become faster; if the computing speed of a hash doubles you can simply double the number of iterations.

One of the earlier implementations of this technique is the bcrypt algorithm presented at the 1999 Usenix conference (Provos & Mazières, 1999). It uses a modified version of the Blowfish which is computationally intensive and uses configurable iterations to produce a computationally expensive and scalable hashing algorithm. Despite being introduced relatively recently, the computational intensive portion of the algorithm has been mitigated by GPU implementations.

A more recent algorithm is scrypt which was introduced at BSDCan in 2009 (Percival C., 2009). It is specifically designed to be difficult to implement in GPUs by using large amounts of randomly accessed memory. This algorithm is currently in draft status as in IETF standard (Percival & Josefsson, 2013), but is widely supported by the security community.

2.3.3. Encrypting Hashes

Once you have salted your passwords and used a slow hashing algorithm, the next step is to introduce encryption. The most straight forward approach is simply to encrypt the passwords using a strong encryption algorithm such as AES. This makes the hashes uncrackable without the encryption key. However, this also introduces problems with key management which is a difficult issue in itself. If a system is fully compromised the encryption will likely be compromised. However in many cases, such as the LinkedIn attack, the passwords are leaked through a SQL injection attack and the server was never fully compromised. Since encryption keys are stored outside the database, this provides an extra level of protection.

To provide an extra level of security, a hardware security module (HSM) can be employed. An HSM moves cryptographic functions or keys off the server into a separate device. This type of security is very difficult to defeat because the keys cannot be stolen; they exist within the HSM hardware and are not retrievable.

3. Conclusion

Password leaks are becoming a common occurrence on the internet with several large scale leaks happing every year. These leaks have revealed the poor practice many companies employ when storing their passwords. The widely available lists of common passwords, an expanding knowledgebase on how user select passwords, and advances in password cracking technologies have made basic hashes more vulnerable than ever. However there are several security measures that can be put in place to increase the security password hashes:

- Use strong hashing algorithms
  - Don't try to create your own hashing algorithm
  - Don't use outdated algorithms (such as MD5 or SHA1)
  - Use SHA2 or similar strength algorithm
- Salt Hashes
  - Use a different salt for each hash value
  - Use long salt values of at least 32 or 64 bits
  - Ensure hash values are randomly generated
  - Don't go out of your way to try to hide hash values
- Employ techniques to slow password cracking
  - Use a key stretching algorithm
  - Scrypt is currently the preferred algorithm
- Encrypt the password hashes
  - Encrypt hash value
  - Employ strong key management for encryption keys

Kelly Brown, kbrown@aboutweb.com

4. References

Dobbertin, H. (1996, Summer). The Status of MD5 After a Recent Attack. *Crypto Bytes* .

Goodin, D. (2012, June 6). *8 million leaked passwords connected to LinkedIn, dating website*. Retrieved August 3, 2013, from Arstechnica: http://arstechnica.com/security/2012/06/8-million-leaked-passwords-connected-to-linkedin/

Gosney, J. M. (2012). Password Cracking HPC. *Passwords^12.* Oslo, Norway.

Gross, D. (2012, July 13). *Yahoo hacked, 450,000 passwords posted online*. Retrieved August 4, 2013, from CNN: http://www.cnn.com/2012/07/12/tech/web/yahoo-users-hacked

Hashcat. (2013, August 12). *Hashcat*. Retrieved August 12, 2013, from Hashcat: http://hashcat.net

Hashcat. (2013, August 12). *oclHashcat-plus*. Retrieved August 12, 2013, from Hashcat: http://hashcat.net/oclhashcat-plus/

Hellman, M. E. (1980). A Cryptanalytic Time - Memory Trade-Off. *IEEE Transactions on Information Theory* , 401-406.

Imperva. (2010). *Consumer Password Worst Practices.* Retrieved November 4, 2013, from Imperva: http://www.imperva.com/docs/WP_Consumer_Password_Worst_Practices.pdf

Info Security. (2010, November 18). *SHA-1 crypto protocol cracked using Amazon cloud computing resources*. Retrieved August 13, 2013, from Info Security: http://www.infosecurity-magazine.com/view/14059/sha1-crypto-protocol-cracked-using-amazon-cloud-computing-resources/

Li, C.-T. (2010). *Handbook of research on computational forensics, digital crime, and investigation: methods and solutions.* IGI Global.

National Institute of Standards and Technology. (2012, August). *Recommendation for Applications Using Approved Hash Algorithms.* Retrieved August 7, 2013, from

Information Technology Laboratory: http://csrc.nist.gov/publications/nistpubs/800-107-rev1/sp800-107-rev1.pdf

National Institute of Standards and Technology. (2012, March). *Secure Hash Standard.* Retrieved August 7, 2013, from Information Technology Laboratory: http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf

NVIDIA. (2013, August 13). *What is CUDA*. Retrieved August 13, 2013, from NVIDIA: https://developer.nvidia.com/what-cuda

Openwall. (2013, August 12). *John the Ripper*. Retrieved August 12, 2013, from Openwall: http://www.openwall.com/john/

Oxid.it. (2013, August 12). *Cain & Abel*. Retrieved August 13, 2013, from Oxid.it: http://www.oxid.it/cain.html

Percival, C. (2009). Stronger Key Derivation Via Sequential Memory-Hard Functions. *BSDCan.* Ottawa, Canada.

Percival, C., & Josefsson, S. (2013, September 24). *The scrypt Password-Based Key Derivation Function (draft).* Retrieved August 13, 2013, from Internet Engineering Task Force: http://tools.ietf.org/html/draft-josefsson-scrypt-kdf-01

Provos, N., & Mazières, D. (1999). A Future-Adaptable Password Scheme. *USENIX ATC '99.* Montgomery, CA: Usenix Association.

RainbowCrack Project. (2013, August 10). *List of Rainbow Tables*. Retrieved August 10, 2013, from RainbowCrack Project: http://project-rainbowcrack.com/table.htm

RSA Laboratories. (2000, September). *PKCS #5: Password-Based Cryptography Specification*. Retrieved August 13, 2013, from Internet Engineering Task Force: https://tools.ietf.org/html/rfc2898

Schneier, B. (1996). *Applied Cryptography, Second Edition: Protocols, Algorithms, and Soure Code in C.* New York: John Wiley & Sons, Inc.

Signler, M. (2009, December 14). *One Of The 32 Million With A RockYou Account? You May Want To Change All Your Passwords. Like Now.* Retrieved August 3, 2013, from TechCrunch: http://techcrunch.com/2009/12/14/rockyou-hacked/

Steube, J. (2012). Exploiting a SHA1 Weakness in Password Cracking. *Password^12.* Oslo, Norway.

Kelly Brown, kbrown@aboutweb.com