Interested in learning more about security?

SANS Institute InfoSec Reading Room

Password Security-- Thirty-Five Years Later

User passwords were first introduced to allow time sharing on large mainframes. Mainframes at the time were large systems within a controlled environment. Passwords were stored in plain text file with no perceived need to secure user passwords. The first documented password breach occurred when a user wanted to gain additional time on a mainframe than the allotted four hours limit. The user simply printed the password file giving birth to the first documented password breach. Developers are continuing to develop new wa...

Password Security—Thirty-Five Years Later

GIAC (GSEC) Gold Certification

George Khalil, George@GeorgeKhalil.com

Advisor: Chris Walker

Accepted:

(Date your final draft is accepted by your advisor)

Abstract

User passwords were first introduced to allow time sharing on large mainframes. Mainframes at the time were large systems within a controlled environment. Passwords were stored in plain text file with no perceived need to secure user passwords. The first documented password breach occurred when a user wanted to gain additional time on a mainframe than the allotted four hours limit. The user simply printed the password file giving birth to the first documented password breach. Developers are continuing to develop new ways to protect user passwords. Initially, Unix encrypted the passwords and stored them in passwd files along with user names. Microsoft developed their own LANMAN hashing protocol which was proven to be weak and easily cracked. Vendors keep on increasing their encryption strength to slow down attackers as they increase their brute force attacking speeds using specialized hardware. One-way Hashing was a ground breaking method to protect passwords through removing the capability to reverse the encryption. Attackers diverted their efforts to the time-memory trade off approach. Attackers adapting to the increasing adoption of one way hashing focused on pre-computing the hash values in advanced. An attacker with access to large computing power can pre-calculate large number of hashes and crack hashes within a fraction of the time compared to brute forcing.

Vendors responded to the time-memory trade off threat by increasing the number of password hashing re-iterations and introduced salting to create unique hashes for each password. Attackers developed tools to taking advantage of the vast multi-threading capabilities of GPUs as well as utilizing scalable cloud computing power. Bitcoin technology revolves around distributed hashing to secure transactions. Bitcoin-mining paid users to provide the computing power to perform these calculations in return of a fee. The financial incentive provided the right environment for high powered ASICS specializing in unprecedented hash functions speeds. The similarity between the ASICS used to break the AES protocol, and the BitCoin mining hardware could one day permit the use of Bitcoin mining hardware to calculate speeds far beyond what today's current GPU speeds can offer.

1. History of password use

Computer historians trace the first use of a computer password back to Massachusetts Institute of Technology in the 1960s (McMillan, 2012). MIT's time-sharing computer, called Compatible Time-Sharing System (CTSS), was designed to accommodate multiple users on many terminals. MIT implemented a solution to this challenge by creating the first known password that identifies individual users and grants them access to the system. The system stored data using punch cards and allocated four hours per week to each user. Despite the advantages of the innovation, however, the researcher that pioneered the use of passwords on CTSS did not incorporate password security into the system architecture. In the spring of 1962, Allan Scherr, a Ph.D. researcher at MIT, wanted to increase the time allotted to him on CTSS so that he could complete his performance simulation. He addressed this dilemma simply by printing out the password file—a strategy that afforded him access to other users' accounts. Scherr's solution was the first documented computer password breach. In 1966, a software bug

caused the password file to be displayed in the welcome message that appeared upon login. This breach was possible because the passwords were stored in plain text.

The earliest documented attempt to encrypt user passwords dates back to the early 1970s over the Multics User Control Subsystem Mainframe (MULTICS) used by the US Air Force. Multics converted user passwords through one-way transformation using a square root ANDed each user password with a mask and discard some bits (Vleck, 2014). Paul Karger and his team attempted to break the passwords on the Multics system to determine how many weak passwords existed. The team achieved a 90 percent success rate because numerous users never changed their assigned default passwords (Multics.org, 2014). The Air Force Tiger team performed a security assessment of Multics in 1972–1974 and successfully breached the system. The findings prompted the developers and administrators to establish new and stronger methods for password protection (Vleck, 2014).

During the 1970s, UNIX developers realized the importance of password security. UNIX operating system's 6th edition implements a password cipher that simulates the M-209 cipher machine used in World War II. The cipher uses the password as the encryption key rather than using the password as the protected plain text. The inverted design of using passwords as an encryption cipher key was implemented to counteract the known weakness of the M-209 cipher. That is; an attacker successfully broke the cipher and recovered the protected plain text. Recovering the cipher key was a considerably more difficult matter (Morris, 1978). In his 1978 study on possible attacks against the M-209 cipher key, Robert Morris calculated that on the basis of the average password length used at the time, a key brute force dictionary attack against 62 alphanumerical characters would take 318 hours. The research resulted in the development of

brute force attack dictionary, which was created using user passwords that were collected over time.

UNIX 7th edition operating system enables fast encryption by implementing the newly available DES encryption and reiterating it twenty-five times. In the design, the password module was modified to prevent users from choosing predictable passwords. The introduction of one way hashing functionality revolutionized password security. Rather than using reversible encryption to secure the password, hashing is a one way operations to secure plain text data. The validation is done by comparison rather than decryption operations. Twelve-bit password salting for randomizing one way hashes was introduced into the operating system to increase the difficulty with which an attacker pre-computes a hashed password. The use of passwords with a unique salt system makes it nearly impossible for an attacker to determine quickly whether a user has the same password across different systems (Morris, 1978). At the time, DES decryption capabilities in software were limited and excessively time-consuming. DES hardware encrypting chips were developed to offload encryption to hardware and significantly increase the speed. In the 1980s, DES software crypt modules increased in speed by 10- to 20-fold, but systems that used DES remained at great risk because password length was limited to eight characters. Meanwhile, brute force attackers were increasing in speed. Attackers brute force hashes; encryption keys and passwords by performing these calculations depends on the speed. The increase of complexity through DES key increase, Hashing, multiple re-iterations as well as salting focuses on slowing down the attacker brute force.

UNIX usernames and hashed passwords were initially stored in a single file that can be read by all users. With the release of System V and BSD 4.3 in 1988, developers introduced password shadowing, which separates user names from passwords. The introduction of the

shadow file enabled the application of restrictions on standard users' access to password hashes. By the 1990s, password auditing tools, such as COPS, Crack, Cracker Jack, npasswd for password strength testing and validation, became available. In the early 1990s, Microsoft introduced LANMAN hashing followed by NTLM for Windows NT. LANMAN hashes use DES encryption while weakening a password by reducing its length and lack of case support. NTLM uses better MD4-based hashes to protect user passwords, but in contrast to UNIX, it does not use salting or iterations.

By the early to mid-1990s, BSD extended DES-BASED crypt(3) to properly support long passwords and permit configurable iteration counts of up to 725 iterations, along with 24-bit salting (Peslyak, 2012). In 1994, FreeBSD introduced the MD5-based crypt(3) library, which enables the use of long passwords and 1000 iterations of MD5 with up to 48-bit salting. Most Linux distributions adopted the FreeBSD MD5 library by the late 1990s. Early rainbow table attack tools, such as Qcrack and BitSlice, were released around 1997, thereby allowing fast pre-computations of DES hashes while supporting salting designed to attack a dictionary with 4096 possible salts (Peslyak, 2012). As computers and users became connected and networked in the late 1990s, password attacks extended to networks. During this period, non-switched Ethernet was a commonly used technology. Passwords were secured at rest but usually transmitted across networks during operation. In the early 1990s (Peslyak, 2012), attackers targeted passwords in transit by sniffing compromised servers, thereby intercepting passwords transmitted by all the users in a local segment.

Developers adapted to the networked computing movement and implemented challenge-response pairs, Kerberos, S/Key, and SSH to provide network authentication without exposing protected keys or passwords. With the expansion of the World Wide Web, web application

authentication became widely adopted. This innovation also extended the allowable password length and highlighted the need to secure passwords. In the late 1990s and early 2000s, numerous web developers built application password security around the PHP MD5 module. However, the adopted PHP MD5 hashing function lacks password salting and reiteration (Peslyak, 2012).

Microsoft's first password hashing attempt was the LAN Manager hash. LANMAN was the only available password hashing algorithm until Microsoft introduced windows NTLM hashing in Windows NT4, 2000, and newer operating systems. LANMAN uses DES encryption to hash a password. This password, however, is converted into all-capital letters, split into two groups of seven characters, and padded to 14 characters. This feature weakens the complexity of passwords. The DES encryption was cracked by the Electronic Frontier Foundation in 1998 in about 23 hours (Sanders, 2010). LANMAN support was maintained for backward compatibility through Windows 7. Given the weaknesses of DES and LM hashing, Microsoft still included it on Windows 7 and Vista, but these features are disabled by default.

NTLM improved security starting with Windows NT, 2000, and newer operating systems by allowing the use of long passwords and allowing the distinction between upper and lowercase letters. Although NTLM increases the complexity of a derived hash to a greater extent than do LM hashes, it remains vulnerable to rainbow table attacks because of the absence of password salting. Increasing length and case sensitivity necessitates rainbow tables that are significantly larger than LM tables. LANMAN and NTLM hashed passwords are stored in a SAM file. Microsoft also encrypts a SAM file by using SYSKey encryption to protect the SAM file that hosts hashed passwords. Although SYSKey provides an additional layer of protection, tools such as SAMDUMP/SAMDUMP2 can recover the SYSKey boot key from the system hive by

using Bkhive to dump the SAM file. Such recovery provides an attacker a copy of the user's hashed password. SYSKey does not protect passwords while they are loaded in memory. To solve this problem, Microsoft incorporated the MD5 digest and AES key encryption into Windows Vista, 7, 8, 2008, 2008R2, and Server 2012 (Microsoft, n.d.). Passwords in a networked domain environment are centrally managed through active directory and stored in the NTDS.DIT database.

Authentication needs are not limited to local systems; operating systems are networked and offer users remote services that require authentication. Microsoft provided several iterations of challenge-response authentication protocols. Protocol names are similar to the local password hashing algorithms used by Microsoft and to hashing algorithms, except when it comes to the remote challenge-response component. Microsoft's LANMAN authentication was the first challenge-response authentication available on the Windows platforms introduced prior to Windows NT (Pilkington, 2012). An LM authentication client's response is based on already compromised LM hashes; the derived handshake hash is inherently weak due to the dependency on the LM hashing algorithm. To improve security, Microsoft introduced NTLM challenge-response authentication with Windows NT. NTLM challenge-response adds a second response from a client on the basis of the NT hash and an LM hash. If a password exceeds the limits imposed on LM hash specifications, response is returned only on the basis of the NT hash (Pilkington, 2012). Via Windows NT 4 SP4, multiple security features were introduced, most notable of which was NTLMv2. The latest challenge-response protocol provides client and server mutual authentication that exclusively relies on an NT hash (Pilkington, 2012).

The challenge that confronts password hashing has always been usability versus security. Hashing algorithms are constantly limited by existing CPU capabilities. The operating system

should be able to perform appropriate hashing calculations to validate the user's password without negatively affecting the user's login experience. Moore's law predicts CPU speed to double every 24 months (Strickland, 2014). This growth has deterred the effective functioning of older hashing algorithms because attackers can compute hashes at speeds significantly faster than the rate needed to authenticate a user. Older password protocols from UNIX and LM also hashing limit password length. Attackers began using distributed computing to increase available computing power. As estimated by the SANS password calculator, a 10-character password composed of upper and lowercase letters and numbers can be cracked in less than a day using a cluster of GPU equipped computers running oclhashcat-plus (Bolson, 2009). Security researcher Jeremi Gosney presented a 25-GPU password cracking unit in 2012 password conference in Norway. Gosney's password cracking server achieves 348 billion NTLM hashes per second. His system is capable of brute forcing a 14-character Windows XP LM password in only six minutes (Roberts, 2012) and brute forcing an 8-character NTLM password in 5.5 hours. The system also achieves 180 billion MD5 hashes per second, 63 billion SHA1 per second, 71,000 bcrypt hashes per second, and 363,000 sha512crypt hashes per second. Gosney and Prof. Amnon Barak indicated that they were working on extending hashcat implementation using virtual open clusters to support up to 128 GPUs. The system was tested against the 6.4-million LinkedIn password hash that was leaked online, and it achieved 90 to 95 percent success rate in brute forcing the hashes (Roberts, 2012).

Although multi-GPU systems offer significant power to a password tester, a tester with a budget of under US$200 can test the acquired hashes against pre-computed hashes through cloud services, such as CloudCracker.com. For US$17, CloudCracker.com offers network handshake hash testing against a 300 million-word dictionary through its cloud platform. Testing is

accomplished in 20 minutes. The company also offers multiple massive dictionaries and on-demand CPU capacity without the need to invest in custom hardware and spend significant time pre-computing hash dictionaries. The dictionaries available range from 385 billion LM/NTLM hashes to 72 trillion hashes, which encompass the entire DES key space (CloudCracker.com, 2014). North Carolina State University published a whitepaper called "Abusing cloud-based browsers for fun and profit" in 2012. The researchers explored the use of available free cloud-based browsing services as scalable cloud-cracking engines. The concept exploits the capability of cloud browsers to execute JavaScript, and the browser map-reduce architecture was specifically tested on cloud browsers, such as Pufin, Amazon Silk, Opera Mini, and Cloud Browser. The researchers, who conducted limited testing, achieved 24,096 hashes per second, or 200 million hashes per job (Tendulkar, 2012). Although Amazon EC2 instances are available for as low as ¢8 per hour (with options to add GPU powered instances), the exploitation of much larger scales of free cloud-based resources can be used by malicious users who seek anonymity while taking advantage of cloud scalability.

2. Password attacks and defenses

2.1. Proprietary hashing

With the first hashing implementation of password encryption, Microsoft opted for self-developed LANMAN hashing. The hashing algorithm does not feature case sensitivity or leverage passwords longer than seven characters. Attackers then focused on brute forcing passwords. As CPU speeds increased, attackers were able to achieve significant progress in cracking LM hashes because of the aforementioned password limitation. The SANS password calculator estimates that a single GPU-powered computer can calculate 50 percent of all

alphanumeric characters while preserving case for 8-character passwords in .088 days. Attackers can pre-calculate hashes and use rainbow tables or freely available rainbow tables. The time versus memory tradeoff significantly improves the speed. A 14-character password composed of numbers, case-sensitive letters, and special characters hashed by LM hashing can be brute forced in 6.09e+12 years with the use of the CPU. The same hash can be decrypted in 28 minutes by using rainbow tables (Rahul Sharma, 2010). Unfortunately, the protocol design of Microsoft's hashing algorithm is flawed because of the seven-character password segmentation. Administrators and security professionals are encouraged to disable LM hashes in their environment. Although Microsoft disabled LM password hash storage in Windows 7 and higher versions, researchers discovered that a Windows 7 SP1 desktop calculates the LM hash for passwords and stores it in memory. SANS researcher Mike Pikington tested a Windows 7 workstation in a domain configuration. The domain was configured using GPOs to disable the LM hash storage, per Microsoft's recommendation, and deny weak challenge-response protocols. The researcher tested passwords of less than 15 characters; the hashes were loaded in memory, regardless of the security setting that uses LM hashes, which enable a tester to crack the password with an online LM dictionary in a matter of seconds (Pilkington, 2012). Research confirmed that passwords under 15 characters are available in memory in the form of the LM hash, regardless of what domain security configurations are in place. Passwords exceeding 15 characters are available in memory only in the form of a more secure NT hash.

2.2. Increasingly complex public encryption algorithms

After the LM proprietary encryption design flaw had been identified, the community adopted community-tested encryption algorithms. UNIX 7th edition has DES encryption, which protects stored passwords. Given the advent of increasingly faster processors, developers

reiterated the hash twenty-five times to increase the difficulty level against attacks. In 1972, the National Institute of Standards and Technology (NIST) set DES as the standard encryption. The protocol was developed in collaboration between the NSA and IBM. After a two-year analysis, NIST reduced encryption key length from 128 bits to 56 bits. The decision caused widespread allegation of mistrust and skepticism regarding government eavesdropping and backdoor encryption allegations (Zande, 2001). After DES had been implemented, the security was re-evaluated every five years up to the 1999 RSA challenge when DES was cracked using a global network of 100,000 PCs in 22 hours and 15 minutes. Since then, the DES breach and the explosive growth of CPU, GPU, and cloud computing power have rendered DES an insecure standard. The industry turned to triple DES to increase key length from 56 bits to 128 bits. Triple DES encryption applies three rounds of DES encryption by using different keys that enable a 128-bit key length; it also provides up to 16 bits for parity, thereby effectively leaving an 112-bit key. Although triple DES has not been cracked, it is only meant as a stopgap measure to extend the use of the DES algorithm until additional ciphers are vetted out by the community. AES was adopted by NIST in 2001 and set as a standard by the US government in 2002 as a replacement for DES. Although the LANMAN hash is based on a proprietary implementation of DES-style encryption, vendors opted to use one-way hashing rather than key-based hashing tools, such as 3DES or AES, to protect passwords.

2.3. One-way hashing

In the early 1990s, BSD OS abandoned password encryption in favor of one-way hashing, which is implemented via the MD5 hashing of passwords. Encryption supports a reverse path in decrypting cipher text. One-way hashes cannot convert hashed data back into original plain text. Operating systems perform the one-way hashing function when a password is

created and store the output in an appropriate storage location. As a user enters a password to authenticate, the hashing algorithm calculates and compares the result with the stored hash. If the results match, the user is granted access but if the results do not correspond, the user is denied access. Windows implemented MD4-based hashing for active directories and NT hashed passwords. UNIX continues to increase security by increasing the number of hashing iterations to elevate the computing power required to brute force password hashes. More complex algorithms were introduced; these included the SHA hashing of passwords, with password stretching achieved by increasing the number of reiterations. The complexity of hashing algorithms continues to grow because of the increasing power available to attackers. According to the latest GPU brute forcing benchmarks, a single ATI Radeon HD69xx can achieve 10.3 billion single-round MD5 hashes per second or 2.6 billion single-round SHA1 hashes per second. One Radeon R9 295x2 GPU can brute force 23.2 billion single-round MD5 hashes per second or 5.97 billion single-round SHA1 hashes per second (Golubev, 2014). The cost of the initial electronic frontier foundation DES cracker machine, Deep Crack, which broke DES encryption, was an estimated US$250,000. Deep Crack can test 90 billion keys per second. The cost of a Radeon R9 295x2 is US$1,500, and it can test 10.3 billion MD5 hashes per second. CloudCracker.com offers MSCHAPv2 cracking to the public; it can brute force MSCHAPv2 protocol keys in under 24 hours. SciEngine offers a single hardware-accelerated server using 128 FPGAs capable of brute forcing 292 billion keys per second (Sciengines.com, 2009).

Bitcoin mining is an interesting field that has accelerated the development of custom ASICS and FPGAs. Mining hardware is developed to process Bitcoin hashing through the use of only double SHA-256. Given the financial incentive involved in Bitcoin mining, vendors compete in rolling out the fastest miner in the market. Minerscube 15 TH/s currently advertises

15 trillion double SHA-256 operations per second at a cost of approximately US$10,000. Although no reports of Bitcoin mining hardware being modified for password hash brute forcing have been published, FPGAs and ASICS have been used to break encryption protocols via the Deep Crack and SciEngine machines. GPUs are presently one of the fastest off-the-shelf hardware used for brute forcing and Bitcoin mining. However, current Bitcoin miners offer up to 1,000-fold increases in performance over the fastest GPUs available in the market (Bitcoin.it, 2014).

2.4. Rainbow tables

CPU, GPU, and processing power is continuously increasing, and attackers are seeking near-real-time cracking capabilities. In 1980, cryptography researcher Martin Hellman published "A cryptanalytic time-memory tradeoff," which discusses the pre-calculation of cipher text for every possible key at the cryptanalysts leisure and the storage of keys in tables sorted by the resultant cipher text (Hellman, 1980). The creation, storage, and usability of a database that contains every possible hash for every possible key are improbable for large key spaces. A password chain solution was devised, along with hash reduction functionality. When calculating rainbow tables, the system identifies a beginning and ending hash for each chain. After the identification, a random key is selected. When calculating passwords, the system identifies the chain that best matches the range that contains the hash. Once the chain is identified, the system with a known starting and ending plain text reiterates through that range until a match is found. A rainbow table combines pre-calculated hashing with brute forcing to achieve speed and reduction in storage that cannot be derived by a pure hash table or speed that is impossible to achieve with brute forcing alone (kuliukas.com, 2014).

George Khalil, George@GeorgeKhalil.com

Rainbow tables require significant memory and disk IO to load massive data into memory for searching. Researchers compared various passwords and the efficiency with which such passwords are cracked using a dictionary, brute forcing and rainbow tables. The results are shown in Table 1.

Table 1. Results for the comparison of passwords and cracking efficiency.

Hash	Passoword	Brute force	Wordlist	Rainbow table
d8345c89b946376a7851a65f869badeb	JUBSQUAD08	35 days*	Not found	Not found
a4e98a338bdd9fe9a468942b1ada10da	CHUMPADZ	3 min.	Not found	Not found
bf29e956e892a58066873de557927389	naomi1	2 sec.	3 sec.	7 min.
a62af261a00404b5c4465bbe4d5b9234	r124026243	35 days*	3 sec.	Not found
f5dba72177a903feec9af0ee9ff5108e	scarlett	2 min. 20 sec.	3 sec.	4 min.
d4ed17ec4bca7d7d87efb949a3293d68	rocker	2 sec.	3 sec.	19 min.
87691c22664559a76b255e01f4a22382	doraines1	23 hours*	3 sec.	13 min.
5f4dcc3b5aa765d61d8327deb882cf99	password	28 sec.	3 sec.	1 min.
4e562c4d962d7f0ecd3076a696deb864	193644	3 sec.	Not found	2 min.
d5f9762acb42de03a05fa69243eb43f0	ericson	4 sec.	3 sec.	2 min.
100416b93d34d3482c47a7f06ca50f29	12345678901234	23 hours 15 min.*	Not found	30 sec.
b2974bde01dccc073d9d186300ebdcc8	AsD12 !!	72 days 13 hours.*	Not found	37 min.
5d69dd95ac183c9643780ed7027d128a	password9	1 day 3 hours.*	Not found	46 sec.

Source: Genbox (2012)

The above-mentioned test demonstrates that some rainbow tables more slowly crack simple passwords than does a word list or brute force attack. As a response to this problem, the security community and private companies invested in computing power or outsourced table generation to distributed computing volunteers. Various rainbow tables boast of different success rates; an example is the freerainbowtable project initiated by the security community. The concept that underlies the project is the use of distributed computing power to generate free rainbow tables for the community. After the community's rainbow table client software is installed, the computer then operates in the community's distributed computing hashing system. Currently, the freerainbowtable project offers the following rainbow tables:

1. LM rainbow tables (416 GB)

2. MD5 rainbow tables (3,889 GB)

3. MYSQLSHA1 rainbow tables (1,480 GB)

4. NTLM rainbow tables (3,957 GB)

Rainbow tables effectively break single-round password hashes by pre-computing the hashes in advance and running them through the reduction functions for results. Key space significantly increases as the number of character's increase, thereby resulting in larger tables. These tables, in return, require considerably longer time for computations, faster IO, more memory, and larger storage.

2.5. Password complexity enforcement

The advances leading to the development of rainbow tables effectively compromised the simple passwords protected by one-way hashes. Excellent technologies have been created for passwords as long as ten characters, for which tables are broken down into lowercase and uppercase letters, numbers, and symbols. Rainbow tables for eight characters or more are significantly larger than one- to eight-character passwords. Developers recognized this flaw as soon as the early UNIX systems were released and added additional modules to authentication systems. Defenses had been in place long before rainbow tables became mainstream tools. Early UNIX systems are characterized by password complexity requirements, password change requirements, aging, and password history. These features prevent users from creating simple passwords that are easily attacked by brute force. The same techniques protect hashes from rainbow table attacks by increasing the complexity that characterizes the advance pre-calculation of hashes. Early systems impose a maximum password length of eight characters—a feature that is no longer an issue with modern operating systems, which support significantly longer passwords.

George Khalil, George@GeorgeKhalil.com

Password complexity enforcement on modern operating systems continues to be weakened by users. Many users share the same passwords across multiple systems; some systems use stronger hashing algorithms than others. If a system with a weak algorithm is compromised, an attacker can test plain text passwords against the other systems to which a user has access. Password change policies are designed to compel users to modify their passwords in a period shorter than the time it takes to crack such passwords, but many users tend to use dictionary words and add a number sequence and special character. These elements counteract complexity requirements and weaken an organization' security. Word lists can be generated on the basis of dictionaries with specified morphing factors to add the common two digits and special characters that most users employ. This method enables attackers to apply brute force at a more rapid rate than that achievable by iterative attack or rainbow tables. Security awareness and technical password complexity enforcement are crucial in educating users to use passphrases rather than passwords and unique passwords for each system.

2.6. Password salting

Increasing password length and applying password stretching techniques via re-iterative hashing increase the costs incurred by an attacker that applies brute force and rainbow tables. Nevertheless, hashes remain vulnerable to time–memory tradeoff attack if an attacker has access to faster systems and more storage than those currently available to users. Developers are seeking to create an encryption algorithm that can keep pace with consistently increasing CPU power. Security researchers now use dollar-hour calculation rather than CPU cycles. This new measure enables security architects to measure the cost that an attacker bears as it applies brute force to password hashes. This strategy is preferable to the use of fixed CPU cycles. The goal is to ensure that attackers incur high costs when they apply brute forcing to make password brute

forcing financially unsolvable. The initial attempt at increasing an attacker's CPU cost was the randomization of passwords. Password hashing creates a unique hash for each password, and attackers that invest CPU dollars can identify the same hash across multiple systems, thereby maximizing the value of their efforts.

Security architects introduced password salting to combat pre-computation attacks. In this strategy, architects add a unique random value to plain text passwords prior to hashing it. The random value generated by the system is unique for each account, thereby resulting in a unique hash value for that system. This added complexity prevents large-scale rainbow table re-use across multiple systems. Prior to password salting, a single rainbow table can be used to crack password hashes collected from multiple systems. With the addition of a random string to each user's account, an attacker is forced to resort to real-time brute forcing after obtaining the hash and its unique salt string. The increased dollar per hour increase due to the addition of salt to the hashing is not limited to attacks on a single user's password. An attacker cannot re-use a hash across multiple systems or determine whether a user is re-using the same passwords across multiple systems.

2.7. Next-generation password encryption

Next-generation password hashing focuses on increasing the cost of pre-computing hashes and preventing the re-use of hashes. Hashing algorithms that require many computing cycles result in large costs for attackers. Memory-hard functions are gaining popularity as hashing algorithms because of the larger computing power required processing these hashes. Table 2 shows a comparison of one year's worth of hardware costs incurred by password brute forcing using various hashes versus the costs incurred with the use of PBKDF2, bcrypt, and scrypt memory-hard functions.

Table 2. Cost comparison.

KDF	6 letters	8 letters	8 chars	10 chars	40-char text	80-char text
DES CRYPT	< $1	< $1	< $1	< $1	< $1	< $1
MD5	< $1	< $1	< $1	$1.1k	$1	$1.5T
MD5 CRYPT	< $1	< $1	$130	$1.1M	$1.4k	$1.5 × 10^{15}
PBKDF2 (100 ms)	< $1	< $1	$18k	$160M	$200k	$2.2 × 10^{17}
bcrypt (95 ms)	< $1	$4	$130k	$1.2B	$1.5M	$48B
scrypt (64 ms)	< $1	$150	$4.8M	$43B	$52M	$6 × 10^{19}
PBKDF2 (5.0 s)	< $1	$29	$920k	$8.3B	$10M	$11 × 10^{18}
bcrypt (3.0 s)	< $1	$130	$4.3M	$39B	$47M	$1.5T
scrypt (3.8 s)	$900	$610k	$19B	$175T	$210B	$2.3 × 10^{23}

Source: Percival (2009)

Although bcrypt is currently available in Java, PHP, ASP.NET, and other similar platforms, it has not been incorporated into UNIX or Windows authentication systems. Bcrypt increases the costs borne by an attacker, but it suffers from a built-in weakness of a 72-character password aggregate length for a password and salt (PHP.net, 2014). Scrypt, on the other hand, does not impose character limitations and has been implemented as a file encryption protocol in tarsnap and several crypto currencies, such as lite coin and dog coin, as well as in several programming languages, including C sharp, Java, PHP, Ruby, Python, and C. Because of the scrypt encryption adoption for crypto currencies, several vendors developed coin mining machines that run custom ASICS designed specifically to calculate scrypt encryption. Scrypt offers users options to reduce the amount of memory used to calculate a hash. Emerging protocols increase security by introducing the memory-hard concept. The protocols require large amounts of RAM and, therefore, make parallel processing more difficult given memory limitations. Scrypt enables users to reduce the amount of memory, thereby diminishing its effectiveness.

George Khalil, George@GeorgeKhalil.com

3. Conclusion

The battle to secure user passwords will continue as computing power grows. As demonstrated in the past 40 years, secure encryption protocols and hashing algorithms should keep pace with the changes engendered by ever-increasing computing power. Researchers are now seeking protocols that are specific to operating environments. Future protocols will require more resources to run but facilitate user performance. Protocols should prefer CPUs rather than GPUs. Servers will unlikely possess GPU parallel computing power because their availability in enterprise servers is also highly unlikely (Peslyak, 2012). Protocols should seek to differentiate factors and apply performance bias that determines which security measures excellently perform on enterprise hardware and underperform on standard desktop hardware. Servers are unlikely to have the GPUs, ASICS, and FPGAs that attackers are using. Currently, salting hashes protect systems against the re-use of pre-computed rainbow tables across multiple systems. However, as long as a salt or local parameter is stored on the same system or file, as is the case in shadow files, an attacker can attack hashes by brute force. The challenge is to remove local parameters or salts from the system without compromising security. The system will continue to require access to salt to validate users' passwords. Removing the salt will prevent an attacker from brute forcing a hash. Nevertheless, providing a mechanism for the system to continue processing password hashing functions while prohibiting an attacker from accessing local parameters constitutes an operational and protocol challenge. Solar Designer discussed the possibility of storing local parameters in hardware via a security module or a dedicated server. Moving data to hardware separates the salt and hash that provides access to the authentication system and protects the salt from theft. These measures render hashes useless to an attacker.

George Khalil, George@GeorgeKhalil.com

Implementing technical encryption and hashing algorithms does not release the user from the responsibility of protecting a password or provide extensive defense to an organization. It is merely another layer of protection characterized by security awareness, network security, logging, auditing, and separation of duties, as well as the creation of protected enclaves. Users remain the largest source of vulnerability in organizations through password re-use, simple password usage, and social engineering. Dual Factor has been proposed as solutions to password problems, but this approach is being challenged by the community. Dual Factor is not mandatory across all authentication systems (Grajek, 2014) and is still vulnerable to theft, such as the RSA token seed theft (Goodin, 2011). Fingerprints, certificates, and smartcards are different forms of passwords that also require their own mechanisms for protecting confidential components from theft. These passwords have their own set of vulnerability issues, such as the seed file example for tokens. No single solution to securing user authentication exists. Protocols and algorithms have to evolve continuously and increase re-iterations, length, memory, and cost because attackers constantly increase their power and develop custom hardware. Passwords should be protected but should remain a tier of a broad-ranging defense scheme for defending, protecting, detecting, and remediating breaches.

4. References

Bitcoin.it. (2014). *Mining Hardware Comparison*. Retrieved from Bitcoin.it: https://en.bitcoin.it/wiki/Mining_hardware_comparison

bolson. (2009, 06 12). *How Long To Crack A Password Spreadsheet*. Retrieved from SANS.org: http://cyber-defense.sans.org/blog/2009/06/12/how-long-to-crack-a-password-spreadsheet#

CloudCracker.com. (2014, 06 15). *CloudCracker Dictionaries*. Retrieved from CloudCracker.com: https://www.cloudcracker.com/dictionaries.html

Genbox. (2012, 01 24). *The Power of Rainbow tables*. Retrieved from IQ Security - Security that makes sense: http://iqsecur.blogspot.com/2012/01/power-of-rainbow-tables.html

George Khalil, George@GeorgeKhalil.com

Golubev, I. (2014, 04 30). *GPU speed estimations for MD5/SHA1/Office 2007/WPA/WinZip/SL3*. Retrieved from golubev.com: http://golubev.com/gpuest.htm

Goodin, D. (2011, 03 18). *RSA Breach leaks data for hacking SecurID tokens*. Retrieved from theregister.com: http://www.theregister.co.uk/2011/03/18/rsa_breach_leaks_securid_data/

Grajek, G. (2014, 02 04). *The Problem with Two-Factor Authentication*. Retrieved from darkreading.com: http://www.darkreading.com/identity-and-access-management/the-problem-with-two-factor-authentication/d/d-id/1113697

Hellman, M. (1980, 07 04). *A Cryptanalytic Time-Memory Tradeoff*. Retrieved from cs.miami.edu: http://www.cs.miami.edu/~burt/learning/Csc609.122/doc/36.pdf

kuliukas.com. (2014, 06 28). *How do Rainbow Tables work*. Retrieved from kuliukas.com: http://kestas.kuliukas.com/RainbowTables/

McMillan, R. (2012, 02 27). *The World's First Computer Password? It Was Useless Too*. Retrieved from Wired.com: http://www.wired.com/2012/01/computer-password/

Microsoft. (n.d.). *Defending against Pass-the-Hash Attacks*. Retrieved from Microsoft.com: http://www.microsoft.com/security/sir/strategy/default.aspx#!password_hashes

Morris, R. (1978, 04 03). *Password Security: A Case History*. Retrieved from Bell-Labs: http://cm.bell-labs.com/who/dmr/passwd.ps

Multics.org. (2014, 05 31). *Site History: RADC*. Retrieved from Multics: http://www.multicians.org/site-radc.html

Percival, C. (2009). *Stronger Key Derivation via Sequential memory-hard functions*. Retrieved from UC Santa Barbara - Scrypt paper: http://www.cs.ucsb.edu/~rich/class/cs290-cloud/papers/scrypt.pdf

Peslyak, S. D.-A. (2012, 06 07). *The History of Password Security*. Retrieved from throwingFire: http://throwingfire.com/the-history-of-password-security/

PHP.net. (2014, 07 02). *password_hash*. Retrieved from PHP.net: http://www.php.net/manual/en/function.password-hash.php

Pilkington, M. (2012, 02 29). *Protecting Privileged Domain Accounts: LM Hashes -- The Good, the Bad, and the Ugly*. Retrieved from SANS DFIR: http://digital-forensics.sans.org/blog/2012/02/29/protecting-privileged-domain-accounts-lm-hashes-the-good-the-bad-and-the-ugly

Pilkington, M. (2012, 09 18). *Protecting Privileged Domain Accounts: Network Authentication In-Depth*. Retrieved from sans.org: http://digital-forensics.sans.org/blog/2012/09/18/protecting-privileged-domain-accounts-network-authentication-in-depth

Rahul Sharma. (2010, 04 27). *Cracking using Rainbow tables*. Retrieved from slideshare.net: http://www.slideshare.net/fullscreen/rahulsharmaait/cracking-using-rainbow-tables/1

Roberts, P. (2012, 12 04). *Update: New 25 GPU Monster Devours Passwords In Seconds*. Retrieved from SecurityLedger.com: https://securityledger.com/2012/12/new-25-gpu-monster-devours-passwords-in-seconds/

Sanders, C. (2010, 01 20). *How i Cracked your windows password (Part 1)*. Retrieved from WindowsSecurity.com: http://www.windowsecurity.com/articles-

tutorials/authentication_and_encryption/How-Cracked-Windows-Password-Part1.html

Sciengines.com. (2009). *Break DES in less than a single day*. Retrieved from sciengines.com: http://www.sciengines.com/company/news-a-events/74-des-in-1-day.html?eab1dd0ce8f296f6302f76f8761818c0=0b59c5b91984fe01986ef3bbc6de9871

Strickland, J. (2014, 04 14). *How Moore's Law Works*. Retrieved from HowStuffWorks.com: http://computer.howstuffworks.com/moores-law.htm

Vasant Tendulkar, J. P. (2012). *Abusing Cloud-Based Browsers for Fun and Profit.* Retrieved from North Carolina University: http://adl.csie.ncu.edu.tw/~jhhe/doc/bmr.pdf

Vleck, T. V. (2014, 05 31). *How the Air Force cracked Multics Security*. Retrieved from multics.org: http://www.multicians.org/security.html

Zande, P. V. (2001, 07 22). *The Day DES Died*. Retrieved from SANS.org: http://www.sans.org/reading-room/whitepapers/vpns/day-des-died-722

George Khalil, George@GeorgeKhalil.com

Last Updated: July 23rd, 2017

Upcoming SANS Training

Click Here for a full list of all Upcoming SANS Events by Location

SANS San Antonio 2017	San Antonio, TXUS	Aug 06, 2017 - Aug 11, 2017	Live Event
SANS Boston 2017	Boston, MAUS	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Hyderabad 2017	Hyderabad, IN	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Prague 2017	Prague, CZ	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS New York City 2017	New York City, NYUS	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS Salt Lake City 2017	Salt Lake City, UTUS	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS Chicago 2017	Chicago, ILUS	Aug 21, 2017 - Aug 26, 2017	Live Event
SANS Adelaide 2017	Adelaide, AU	Aug 21, 2017 - Aug 26, 2017	Live Event
SANS Virginia Beach 2017	Virginia Beach, VAUS	Aug 21, 2017 - Sep 01, 2017	Live Event
SANS San Francisco Fall 2017	San Francisco, CAUS	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS Tampa - Clearwater 2017	Clearwater, FLUS	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS Network Security 2017	Las Vegas, NVUS	Sep 10, 2017 - Sep 17, 2017	Live Event
SANS Dublin 2017	Dublin, IE	Sep 11, 2017 - Sep 16, 2017	Live Event
Data Breach Summit & Training	Chicago, ILUS	Sep 25, 2017 - Oct 02, 2017	Live Event
Rocky Mountain Fall 2017	Denver, COUS	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS SEC504 at Cyber Security Week 2017	The Hague, NL	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Baltimore Fall 2017	Baltimore, MDUS	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, GB	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Copenhagen 2017	Copenhagen, DK	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS DFIR Prague 2017	Prague, CZ	Oct 02, 2017 - Oct 08, 2017	Live Event
SANS Oslo Autumn 2017	Oslo, NO	Oct 02, 2017 - Oct 07, 2017	Live Event
SANS October Singapore 2017	Singapore, SG	Oct 09, 2017 - Oct 28, 2017	Live Event
SANS Phoenix-Mesa 2017	Mesa, AZUS	Oct 09, 2017 - Oct 14, 2017	Live Event
SANS AUD507 (GSNA) @ Canberra 2017	Canberra, AU	Oct 09, 2017 - Oct 14, 2017	Live Event
Secure DevOps Summit & Training	Denver, COUS	Oct 10, 2017 - Oct 17, 2017	Live Event
SANS Tysons Corner Fall 2017	McLean, VAUS	Oct 14, 2017 - Oct 21, 2017	Live Event
SANS Brussels Autumn 2017	Brussels, BE	Oct 16, 2017 - Oct 21, 2017	Live Event
SANS Tokyo Autumn 2017	Tokyo, JP	Oct 16, 2017 - Oct 28, 2017	Live Event
Security Awareness Summit & Training 2017	OnlineTNUS	Jul 31, 2017 - Aug 09, 2017	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced