An evaluation of secure storage of authentication data

Juanita Blue, Eoghan Furey

School of Computing,
Letterkenny Institute of Technology
Co. Donegal
Ireland
Email: eoghan.furey@lyit.ie

Kevin Curran

School of Computing, Engineering &
Intelligent Systems, Faculty of Computing,
Engineering & Built Environment
Ulster University, Northern Ireland
Email: kj.curran@ulster.ac.uk

Abstract

The issue of secure password storage is of major industry concern, as all databases are at risk of breach, potentially leaving all associated user accounts vulnerable. Key industry documents such as 'IEEE Standard Specifications for Password-Based Public-Key Cryptographic Techniques' and 'RFC 2898: Password-Based Cryptography Specification' outline best practice mechanisms for secure storage of authentication credentials. We examine here authentication and secure storage of authentication data, inclusive of common industry implementation and best practice standards and regulation. The broader topic of authentication is examined to assess 'state-of-the-art' technologies that are currently available. The technologies researched included biometrics, multifactor authentication and smart cards and we show that despite the advancements these technologies have contributed to the area of authentication, it is unrealistic to expect their common implementation soon.

Keywords: Authentication, Computer Security, Password Management, Cryptography

1. Introduction

Cyber theft is simply becoming the fastest growing crime in the world. Gartner reports that this rising tide of cybercrime has pushed cybersecurity spending to more than $80 billion in 2016 [1]. Due to the sensitive nature of attacks and breaches, governments and organisations are reluctant to disclose the details of system vulnerabilities that have facilitated these attacks (disclosure itself exacerbates such vulnerabilities) [2]. For this reason, there is difficulty in identifying an exact figure relating to theft of authentication data. However, it can be stated that exposed authentication data (by way of SQL injection or otherwise) often acts as a gateway to greater breaches including compromised user accounts, privilege escalation, code injection and unauthorised direct disk access [3]. Based on this premise, a substantial portion of the sum could be attributed to insecure authentication data storage. Securely stored authentication data is a paramount element of system security. Passwords are implemented as a security feature, providing authentication, authorisation and accounting [4]. They are used as a proven mechanism that prevents unauthorised access to a system. Organisations often include guidelines in their security policy that define password length, entropy and period of use, but often neglect to define how these passwords should be securely stored. Best practice guidelines advise that at minimum, authentication data should be:

- Salted with a random string of not less than 32 bytes
- Encrypted/hashed using a one-way, proven and un-cracked cryptographic hash function (sha256, sha512)
- The output of this hash function should be re-hashed for no less than 1000 iterations [5, 6].

Despite the readily availability of best practise guidelines relating to authentication data storage, it is clearly often an oversight that results in a significant risk to industry. As stated in IEEE document entitled '*An Authentication and Auditing Architecture for Enhancing Security on eGovernment Services*', "…the lack of security at the backend level hinders every effort to find evidence and investigate events related to credential misuse and data tampering". Control measures are wrongly applied at application level, and should be applied at database level instead [7]. It is unlikely that organisations will openly declare that they do not store authentication data to best practise, as doing so poses a grave threat to their user accounts, customer data and system security, not to mention their reputation. Equally, it is unlikely that all organisations will improve or replace their existing insecure infrastructure in the foreseeable future. If the solution could be developed using existing staple technology, it could invaluable to all facets of industry. The solution could be discretely applied to the existing infrastructure by an administrator, to mitigate risks associated with compromised authentication data at minimal cost, effort and downtime.

2. Authentication

There are three main categories of means by which individuals can authenticate and verify their identity. These categories are defined as 'something you know' (password, pin), 'something you have' (token, key fob) and 'something you are' (fingerprint, iris pattern). Organisations that possess higher data-confidentiality requirements have extended their user-authentication systems beyond the database driven and/or directory driven components, like passwords, traditionally used in web applications. Implementation of strong authentication systems spans from simple extensions of existing architectures to high-tech integrated and interconnected technologies. Some of the latest developments and implementations are as follows.

2.1 Biometrics

Biometrics is a form of authentication where one or more of a person's biological traits are analysed to verify their identity. The physical characteristics analysed can include fingerprints, hand or earlobe geometry, iris or retina patterns, voice waves, hand-dorsa vein recognition, DNA and signatures [8, 9]. To form biometric 'identities', raw biometric data is collected in analogue form and converted to digital data that is computer readable. Advancements in digitalisation of analogue data have allowed personal identification by biometrics to become almost instantaneous. Typically, organisations that have employed the use of biometric authentication are those who require higher levels of security such as military bases, government research facilities and financial institutions with large vaults. However, as single factor authentication is considered more vulnerable, regular companies are now seeking two-factor solutions that incorporate biometrics [8]. To date, iris and retina pattern authentication have been used by some bank ATMs, voice wave recognition has been used for access to databanks in research facilities and hand geometry is used in authentication for access to buildings. Law enforcement organisations have employed the use of facial recognition for identifying individuals in large crowds and earlobe geometry has been used to prove identity in cases of identity theft [10].

Regardless of the type of individual traits that are used for authentication, all digitalized data is stored in databases. Biometric data storage invokes the same basic infrastructure as passwords and is vulnerable to the same threats [8]. The secure storage of biometric authentication data has recently been highlighted by the breach of OPM in 2015. In June of 2015, 5.6 million fingerprints were stolen by attackers. Although the attack was downplayed by the company, security experts maintain that it poses serious implications. Misuse of the fingerprint data may currently be limited, but it could be useful in future exploitation as authentication by biometrics continues to be employed [11]. Unlike passwords or credit card numbers, fingerprints cannot be changed. Each set of fingerprints stolen can realistically be interpreted as a stolen verified identity. As biometric authentication is immutable, it is difficult to recover from credentials that are compromised. Some forms are easier to compromise than others. Iris and retina patterns and other biometric data such as electrophysiological signals are currently difficult to replicate. Fingerprints, however, are left everywhere daily and are easy to forge with use of ballistic gel and similar products [10]). Fingerprints are also not guaranteed too be unique and can possess characteristics that are similar enough to be accepted by biometric authentication systems [11]. Facial features can also be reproduced and voice waves can be recorded. Biometrics do not provide absolute assurance of verified identity and in some cases may be deemed unreliable or fall victim to spoofing [12]. The use of biometrics for authentication poses several key challenges, not limited to the theft, replication and unreliability, but other factors that are more inclined to impact widespread implementation. The technology is complex and expensive to deploy, two major factors that would deter organisations from incorporating it [8]. Biometric data may offer improved authentication features, but the associated data is still stored in structured databases and vulnerable to theft. Consequently, recommended best practice mechanisms must still be invoked to ensure security while the data is in transit and storage.

2.2 Two-Factor and Multi-Factor Authentication

Two-factor authentication requires a user to provide two forms of identification, each from a different category. Usually this is implemented by knowledge, such as a security code or password, and a possession, such as a card or physical token. An ATM card is a prime example of two-factor authentication (using a card and a PIN). Multi-factor authentication operates in the same manner but with more than two factors (that include the third category). Multi-factor applications are available for Apple, Blackberry and Google OSs. Some incorporate the use of biometrics or use GPS to verify location as an additional factor. One-time passwords are also a popular implementation, allowing users to receive a code that enables the phone to satisfy the possession factor. Breach of two or multi-factor authentication generally involves physical access to the target individual and/or their environment. Matthew Prince, CEO of Cloudflare suffered a breach of his email account despite the use of two-factor authentication [13]. This attack was meticulously orchestrated and involved the deception of both AT&T and Google [14]. Despite this example, there are certainly additional obstacles set in place with each factor. Independence from one another means that compromise of one factor does not usually lead to the fall of another (Strom, 2014).

PCI-DSS, HIPAA, The Gramm–Leach–Bliley Act (GLBA), The Sarbanes-Oxley Act (SOX) and The Criminal Justice Information Services Division (CJIS) all recommend the use of two-factor authentication in their compliance documents [15]. NIST also advocates the use of tokens as part of two-factor authentication. However, the possession category of authentication can be bypassed with a boot from BIOS, leaving access restrictions dependent on the strength of the remaining factors [16]. 'Possessions' are also susceptible to loss, theft and damage. Regardless of the second factor, the knowledge category is generally a password, stored in a database and is vulnerable to the attack vectors unless stored to best practice [17].

2.3 Smart Cards

Used as a part of possession credentials, smart cards operate as standard plastic cards embedded with integrated circuits or microprocessors. This form is authentication is deployed as an element of two or multifactor authentication and is used as part of a government-sponsored electronic-ID program in several European countries [11]. In the countries that employ their use, smartcard readers are integrated into standard hardware such as PCs and laptops. Authentication required for access to government-related accounts (taxes, social welfare) is gained by insertion of the smartcard, combined with a password. Mobile phone technology has also become integrated with the smart cards, the SIM often acting as the proof of identity and the phone acting as the reader for one-time passwords [11].

Despite the assurances offered by smart cards, the authentication data they are dependent upon is once again stored in a structured database. Additionally, this information is stored along with several other pieces of sensitive information, all relating to the identity of individuals. Successfully implementing a one-way hash system that protects smart cards from attack vectors including brute-force, dictionary and many more types of attacks has proven challenging [18]. Despite utilizing start of the art technology, the use of smart cards in authentication is still only as secure as the mechanisms employed in the database that protects the associated information. As demonstrated by the discussion of state of the art mechanisms in authentication, all forms of authentication data must be stored in databases to enable their functional use. Regardless of how effective that mechanism itself is, the data remains vulnerable to simple attacks such as an SQL injection. Policy and regulation relating to authentication other than passwords is lacking. PCI-DSS, SOX, GLBA and HIPAA documentation all focus entirely on password storage, disregarding biometric [10] and smartcard technologies. However, if industry were to comply with best practice recommendations, all types of authentication data would be stored with salt, one-way hashing and iterations at minimum.

Furthermore, two and multi-factor authentication are still reliant on the use of passwords as part of the authentication process. This reiterates the importance of best practice storage of the data in question. Passwords that are poorly stored will provide an easy hurdle for an attacker. Given the great expense and complexity of biometric authentication, it is unlikely it will become heavily integrated in our daily lives in the foreseeable future. Additionally, the increased expense of two and multi-factor authentication (including smart cards) and the inherent reliance on databases mean that passwords will remain the main form of authentication for some time. The focus must shift to adhering to best practice password policy and storage; creating strong passwords that aim to be impossible to crack in the case of a database breach.

3. Passwords

Passwords are the most common form of authentication. For passwords to be effective, they must adhere to basic standards to avoid weaknesses that enable password guessing and more sophisticated attacks such as brute force or dictionary attacks. Dictionary attacks are executed with automated tools that attempt known dictionary words repeatedly until access is gained to a system [19]. An example of one such tool is 'John the Ripper'. To comply with several industry regulations, it is necessary for enterprises to have stringent password management requirements. These requirements are defined and documented in the security policy of most organisations [10]. Password management is implemented as a mitigation strategy for risks associated with system authentication but as described in this document, often the policies are outdated or ineffectively implemented.

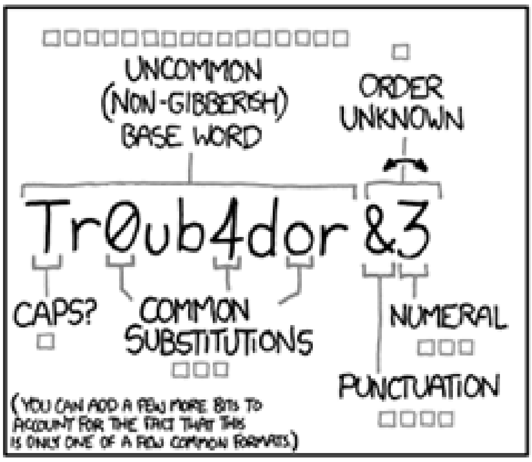

□□□□□□□□□□□□□□□□
UNCOMMON (NON-GIBBERISH) BASE WORD
□
ORDER UNKNOWN
Tr0ub4dor&3
CAPS?
□
COMMON SUBSTITUTIONS
□□□
NUMERAL
□□□
PUNCTUATION
□□□□
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS)

Figure 1: Vulnerabilities of common password policy implementation [20]

Documents containing best practice guidelines relating to password creation are freely available online. These recommendations are produced by security professionals such as Bruce Schneier, Brian Krebs and security authorities like *OWASP*. Unfortunately, consultation of outdated documents and poor interpretation has resulted in security policies that are weak and not tailored meet to best practice advisements. Bruce Schneier identifies the two main issues of password cracking as irresponsible use of cracked hashing mechanisms such as MD5 and non-randomized passwords produced by account holders. As organisations themselves define the security policies that police both these issues, they are ultimately deemed responsible [19]. Good password creation policy has two integral factors: length and entropy. Entropy is largely associated with password length rather than the randomization of the string itself [21]. Common policy implementations have encouraged or enforced users to replace letters with obvious symbols to maintain memorability while creating supposed randomness, as in Figure **1**. This has cultivated an environment where passwords have become difficult for humans to remember and easy for computers to guess, as demonstrated by Figure **2**.

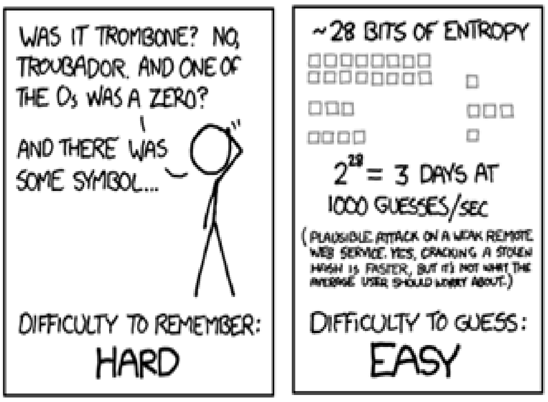

WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER: HARD
~28 BITS OF ENTROPY
□□□□□□□□
□□□□□□□□ □
□□□ □□□
□□□□ □
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS: EASY

Figure 2: Password policy complexity quandary [20]

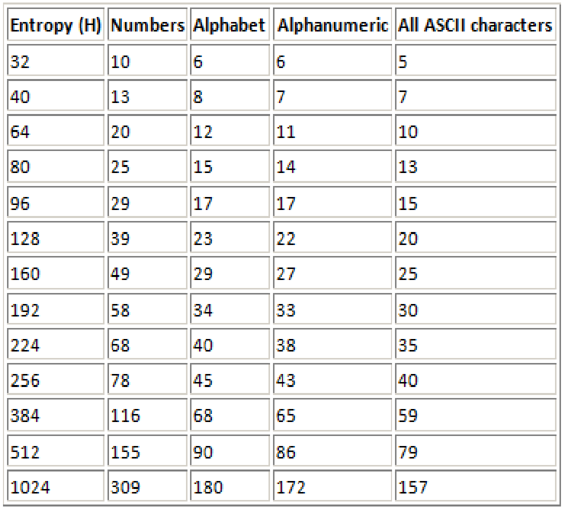
3.1 Password Length

As any system user's personal experience would indicate, minimal length for a new password is eight characters. This commonly-defined length is historically based on legacy Unix passwords from the 1980's with a DES-based crypt() function that were limited to 8 characters (the high bit of each byte was ignored). Eight-character long passwords were deemed the best and safest option that could be hoped for [22]. Currently, experts claim that any low entropy password that is less than 14 characters in length probably has its hash value registered in a rainbow table (a list of pre-computed hashed values for dictionary words and known popular passwords) [23]. Passwords of this length or less are considered extremely vulnerable to a plethora of different attacks. Common password policy enforced by organisations has simply not remained abreast with the advancements in computational processing power and the increasing number of attack vectors. Additionally, as outlined in the following sub-heading, password length has a large impact on password entropy. This acutely increases the importance of password length when attempting to implement a high entropy password. Interestingly, the regulatory and compliance documents, such as PCI-DSS, recommend use of a 'long' password, but do not stipulate the actual length that should be implemented.

3.1.1. Password Entropy

The idea that ‘password complexity trumps length’ is a myth [12]. Entropy offers a representation of the level of randomness contained within a password. This concept can then be translated into a form that conveys how computationally difficult it is to crack a password. Using a long string of words that have no logical relationship with one another such as ‘JapaneseHamburgerDragonCelticFoot’ is far more effective than simply replacing letters with typical symbols in a short password string [23], for example, Manchester becomes M@nch35t3r. It is widely advised that any password implementation that uses names, repeating characters, sequential numbers, well known keyboard entries (qwerty) or personal interests that can be easily enumerated should be avoided. However, these guidelines are seldom displayed on the interfaces that prompt password creation. When implementing a password policy, length and random text should be prioritised over character complexity. This renders the current trend of including uppercase, lowercase, numerals and symbols in every password a fallacy. Computational entropy can be measured on binary bits where: H = The binary bit length of the password, L = The character length of the password and Log_2 = Log function (base 2). N = The number of possible characters/symbols in the password and the formula for measurement of binary bits can be defined as H=L*Log_2(N). The entropy per bit is gained from the length of the password (L), not the possible symbols (N).

As demonstrated by the experimental project *RC5*, conducted by *Distributed.net*, it took volunteers in the organisation 1,726 days (4.5 years) to find a 64-bit key encrypted with RSA. In contrast it only took the group 193 days (6 months) to find a 56-bit key also encrypted with RSA (Distributed.net). Table 1 demonstrates how entropy per bit can be controlled by the selecting password length and the symbols that may be included in the pool. According to Table 1, to achieve entropy per bit of 64 only using numbers, a password would have to be 20 characters long. In contrast, a password length of only 13 is required if the desired entropy per bit is 80 and all the ASCII characters may be used. Currently the recommended entropy per bit stands at a minimum of 72-bits [21]. Based on the table above, this indicates that password length should lie somewhere between 12 and 23 characters, depending on the number of symbols that are included in the pool.



Entropy (H)	Numbers	Alphabet	Alphanumeric	All ASCII characters
32	10	6	6	5
40	13	8	7	7
64	20	12	11	10
80	25	15	14	13
96	29	17	17	15
128	39	23	22	20
160	49	29	27	25
192	58	34	33	30
224	68	40	38	35
256	78	45	43	40
384	116	68	65	59
512	155	90	86	79
1024	309	180	172	157

Table 1: Entropy per bit based on length & symbol pool [21]

3.1.2 Password Lifespan

Although implementation of a password lifespan is generally an enforced element of organisational password policy, research suggests that no security benefits are gained from frequent password changes. A password update is commonly required in organisations once every 30 days. This is based on the legacy mainframe systems that operated without networking. At that time, the biggest authentication risk was password cracking. It was calculated that to try every possible password would take a system at least two months. Hence the expiry date was set to 30 days [24]. Several security experts claim implementation of a password expiry has negative effects as users forget their current password and are more inclined to document it on a post-it or somewhere near their PC [25]. Gene Spafford insists that provided passwords are of acceptable length and entropy, are stored correctly and have not been disclosed, there is no actual security-based reason to regularly update them on a system (Spafford, 2006). Despite this argument, regular updating of passwords is recommended by PCI-DSS and HIPAA but NIST in 2017 issued new guidance to stop this habit [26].

Regardless of their critical role in online authentication, passwords suffer from several in tractable weaknesses (Strahs et al., 2009). Even with policies implemented that dictate password length, randomness and life-span, they remain vulnerable to both cracking and theft. Many organisations have orchestrated policies that focus on password length and ‘randomness’ as a safeguard against brute-force and dictionary attacks, though as demonstrated above, high-entropy is seldom truly achieved. Although increased complexity should be invoked to protect against password guessing (based on enumeration and social engineering), brute-force and dictionary attacks, it provides no defence against password theft. If an authentication database is compromised, every password stored may potentially become a tool that could be employed to commit further crimes. No matter how complex, authentication data must be stored and to be effective, it must be stored to best practice.

3.1.3 Password Attacks

A proxy server is a dedicated computer on a network or virtual system running on a computer that acts as an intermediary between an endpoint device and another server from which a user or client is requesting a service. An advantage of a proxy server is that its cache can serve all users. Proxy servers are used for both legal and illegal purposes. In the enterprise, a proxy server is used to facilitate security, administrative control or caching services, among other purposes. In a personal computing context, proxy servers are used to enable user privacy and anonymous surfing. A man in the middle attack involves an actor, who inserts themselves in between two parties communicating (Figure 3). The actor impersonates both parties and gains access to the information that is being transferred [12]. Man-in-the-middle attacks can be typically executed using ARP poisoning or spoofing where falsified Address Resolution Protocol (ARP) messages are sent over a local area network (LAN). The attack machine is then capable of intercepting data intended for that IP address.

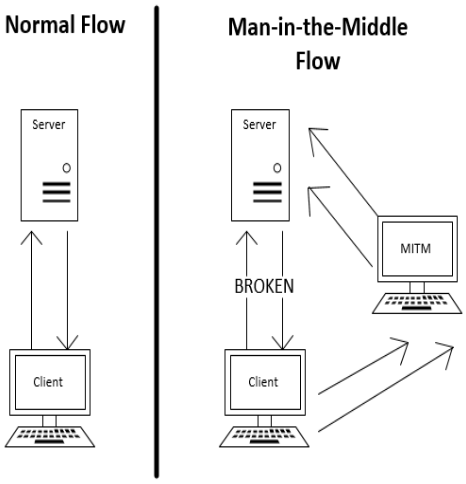


Figure 3: Typical Man-in-the-Middle attack

Cain & Abel is an application that is utilised for password recovery and possesses capacity to execute Arp poisoning, dictionary, brute force and cryptanalysis attacks, in addition to recording VoIP conversations and the possibility to analyse route protocols. It is comprised of two major components, Cain that operates as the front-end application that sniffs and recovers passwords and Abel, a Windows service that scrambles the traffic inside the network, for additional protection.

4. Storage of authentication data

The methods commonly implemented to increase password strength may combat password guessing and dictionary attacks, but they provide no safeguard against password theft. It is considered standard practice in secure systems to keep authentication data private. Compromised credentials can have grave ramifications for the victim organisation and other websites and applications where users have implemented an identical password. Additionally, depending on the type of organisation, there may be legal and regulatory requirements, such as PCI-DSS and SOX. To keep systems safe, secure storage of authentication data is of paramount importance. Several set guidelines for the secure storage of passwords include '*IEEE Standard Specifications for Password-Based Public-Key Cryptographic Techniques*' [5], '*IETF Password-Based Cryptography Specification*' [27] and '*OWASP Password Storage Cheat Sheet'* [28]. They all advocate a minimal standard of Salt, One-way hashing and Iterations. Systems that do not invoke these methods are quite simply non-compliant with PCI-DSS and other compliance document, as they have not implemented 'secure storage'. For optimal security, each of the elements must be implemented as per recommendation and prior to storage in a database. None of these methods are new, each has been utilised for at least two decades. However, they remain the key factors on which industry best practice is based and when implemented correctly prove extremely effective. The implementation variables may have changed in the form of increased number of salt bytes and iterations, or use of improved one-way hash mechanisms, but the core mechanisms recommended remain the same.

4.1 Salt

Salt is the general term for a random string of data that is concatenated to a plaintext password prior to input into a one-way hash function. Salt is used to prevent a system collision where another user has selected an identical password, as each salt string is unique and it is also used to secure against attacks where hash-matching strategies are invoked. These include Rainbow Tables Attacks, Brute Force Attacks and Birthday Attacks. It is recommended that the Java class SecureRandom() is used to produce salt strings. It is designed to be cryptographically secure and purpose built for security purposes such as generating session IDs and encryption keys. Use of weaker random number generators such as Java's Random() class is discouraged as they have known hidden biases [28]. Use of salt is intended to deter attackers from attempting to decrypt an entire database of passwords. For a single password hashed with 12-bit salt, a different rainbow table would have to be created for each possible salt value. An attacker would need to produce 2 to the power of 12 (4096) rainbow tables. For 8 byte salt, this number would jump to 2 to the power of 64 (18,446,744,073,709,551,616). The IETF RFC 2898 documentation of PKCS5 recommends the use of a 64-bit salt string (8 bytes). A more recent evaluation recommends a salt string length of 32 bytes, calculated as 256 bits [28]. Even though the concept of salt has been present for nearly two decades, it is still the most highly recommended [29] and proven method to prevent against attacks that target entire authentication databases.

4.2 One-way Hashing

This is the umbrella term for various algorithms that convert variable-length text to a fixed-length string of digits for data management or security purposes. Once converted, it is almost impossible to derive the original text from the string produced. Identical text inputs will produce identical fixed length string outputs, and a minimal one-bit difference will produce an entirely different output.

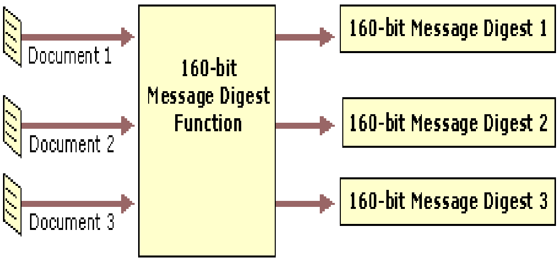


Figure 4: Message digest function (SHA-1)

These algorithms are commonly used with public key technology as an element of authentication. Certain datasets and use of obsolete hashing mechanisms can produce collisions where varying inputs produce an identical output. However, although this has occurred, it is uncommon and takes much computational effort. Encouragingly, for SHA-256 and SHA-512 message digest algorithms it remains mathematically and computationally improbable that a collision shall occur. A good one-way hash function must possess the following properties:

- Preimage resistance: Impossible to compute the output y from the input x (hash(x) = y)
- Second preimage resistance: Impossible to compute input x1 from another input x2 (hash(x1) = hash(x2))
- Collision resistance: Difficult to find two inputs x1 and x2 (hash(x1) = hash(x2)) (OWASP, 2010)

Message digests such as MD1, MD2 and MD5 are now considered cryptographically insecure as documented collisions have occurred (Figure 4). Use of the 160-bit SHA-1 is currently being depreciated by browsers due to a recently discovered susceptibility to collisions and pre-image attacks [30]. Despite loss of Federal Information Processing Standards (FIPS) approval, many of these obsolete and insecure message digest functions are still in common use by organisations globally. Where one-way hashing has been implemented using these algorithms, authentication data is once again vulnerable.

Security authorities now advocate the use of FIPS approved SHA-256 (256-bits) and SHA-512 (512-bits) in authentication, although there are certain compatibility issues that need to be addressed before working implementation can be finalised for digital certificates and the like [31]. Based on the strength of SHA-256 and the increased length of the output (and therefore increased entropy as previously discussed), this one-way hashing mechanism is deemed highly appropriate for use in authentication data protection.

4.3 Iterations

Iterations describe the number of times the output from a one-way hashing function is re-hashed. The use of iterations increases the computational power and time cost of reproducing a key derived from a hashed and salted password. Also known as 'key stretching' and 'slowing', this increases the difficulty of an attack on an authentication database. The best practice recommendation is set at 'not less than 1000' iterations [28]. As this number increases the cost and difficulty of an exhaustive search significantly, yet avoids a noticeable impact on a legitimate user deriving individual keys. The parameter included in the IETF standard was intended to increase over time as CPU power improved. In 2005, a Kerberos standard recommended 4096 iterations. In 2010 Apple incorporated 10,000 iterations for their iTunes authentication database [28], while in 2011, LastPass used 5000 iterations for JavaScript clients and 100,000 iterations for server-side processing.

4.4 PBKDF2

IEEE, IETF, *NIST and OWASP* recommend the JavaScript implementation of the password-based key derivation function 2 (PBKDF2). The function belongs to a family of PBKDs developed to automate secure processing of authentication data. The operation of PBDKF2 is shown in Figure 5. During pre-storage PBKDF2 processing the function intakes five parameters, including original password, salt, salt length, hashing type and iteration count.

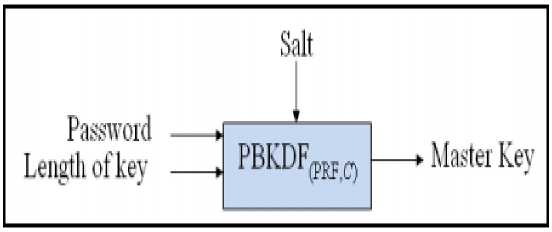


Figure 5: PBDKF2 Function (NIST, 2010)

References to storage of authentication data in *PCI-DSS* documentation, simply states under requirement 8.2, that the data must be stored securely. Compliance documents do not stipulate how achieve items on their checklists. However, failure to implement measures correctly results in non-compliance. Interestingly, the *HIPAA* document makes no reference to storage at all. Despite this over-sight, the mechanisms, implemented as per recommendation are considered best practice in relation to storage of authentication data according to *NIST*, *IEEE*, *IETF* and *OWASP*.

When PBDKF2 is executed with a secure one-way hash function (SHA-256), a salt of recommended length (32 bytes) and an acceptable number of iterations (5000+), this security measure provides an effective safeguard against the useful theft of passwords stored in authentication databases. This is inclusive of passwords that do not meet the basic criteria of best practice password policy (passwords that are short in length with low entropy). Up until mid-2015, SHA-1 was the best practice recommended one-way hash mechanism. Based on recent developments, organisations that previously enjoyed best practice compliance, may have to re-evaluate implementations. The SHA-1 algorithm now poses a serious security risk and implementations must be updated to SHA-256 wherever possible.

5. Evaluation of Authentication Storage Mechanisms

Several ‘Password Management’ applications were identified that offer mechanisms for strengthening passwords. It was established that these applications were essentially pluggable extensions that strengthen singular passwords for various website accounts. Each application invoked variations of best practice methods including one-way hashing, key-stretching (iterations) and salt. The mechanisms were generally applied at the client-side of a specified browser. It was swiftly surmised that these applications were intended for private use by individuals wishing to strengthen authentication for assorted personal accounts. These applications could not operate as valid solution for organisations to improve password storage at the back-end. No commercial applications were discovered that assist administrators in applying best practice security measures to webserver/authentication database infrastructure.

Personal use of password managers is encouraged by industry professionals such as Brue Schneier[1]. Many commercial varieties such as *Dashlane* and *LastPass* are freely available. The applications discussed here encompass the original legacy applications that modern versions are based on. Documentation relating to the internal structure and functionality of modern implementations was scant. This was attributed to the risks involved in documenting the architecture of applications that are intended to improve online account security. The following applications were assessed for functionality and effectiveness:

5.1 Password Agent

This application converts low-entropy passwords used in web account authentication, to a more secure option. It invokes enhanced hashing by way of a salt repository server and a browser plug-in. The salt repository stores a list of salts for each account while the plug-in (Agent) provides a user interface, salt

[1] www.schneier.com/blog/archives/2014/09/security_of_pas.html

retrieval and hashing function. When the user requires a new secure password for a website, they activate the agent and enter a plaintext password into the text field. The web-site specific salt is automatically concatenated to the plaintext password and the string is hashed to generate the stronger, more secure password. Each time the user wishes to access their online account, they enter the plaintext password and the plug-in regenerates the secure string originally produced [32]. This application was designed as a *Mozilla Firefox* extension and implements the salt repository as a Java servlet. The interface is an XML-based REST-style protocol designed for use with XML HTTP Request objects (Javascript). This allows for synchronous HTTPS requests that can translate XML responses into DOM documents. The Agent element is written in Javascript and XML User Interface Language [32].

5.2 Lucent Personalized Web Assistant (LPWA)

This application addresses the use of the same credentials for several web accounts. User ID and password reuse can allow for enumeration across several websites and be dangerous if carried over for login at a user's place of employment. The application serves three main purposes; it generates pseudonymous aliases, filters privacy-sensitive HTTP header fields and provides indirection as the TCP connection between the client node and the website. Traffic is passed through a proxy to prevent tracking of the originating computer. Ultimately the application operates as a HTTP proxy that provides anonymity services to users by creating aliases for usernames, email addresses and passwords. The application does not support HTTPS, instead it must fully trust the proxy server [32]. The proxy remains stateless so it does not retain the information relating to the active browsing session from one request to the next. Instead it introduces a proxy browser that tags each user request with a new user ID and session information.

5.3 PwdHash

This application creates a different secure password for each website [32]. Security is obtained by application of a salt string based on the domain name of the site to each plaintext password created followed by one-way hashing. The application requires no server-side changes. Instead it stores the salt values on the client-side and transmits the hashed and salted password product to the remote website. Each hash output is tailored to meet the website password policy requirements [22]. Use of the domain name as salt does however cause the generated passwords to be vulnerable to dictionary attacks. The application functions by implementing password hashing natively in the *Internet Explorer* browser. The extension listens for when the focus leaves the password field, it then replaces the content of the field with a salted and hashed value. From this point on, anytime the user enters a plaintext password into the field, the password is replaced with a hashed version [22].

5.4 Password Multiplier

This application operates as a browser extension of Mozilla Firefox. Mozilla's cross-platform scripting tools and XUL are invoked to allow for integration. To activate the application, the user authorizes the interface with a user-name and master password. Following the initialization, the application is invoked every time the user double-clicks on a password field. Passwords are strengthened through hashing and the result is cached to disk (so it only needs to be performed once per user, per system). To confirm the hashing, the user must re-enter the master password and domain name of the site [32]. This application requires no changes server-side [33] and supports both HTML form password inputs and standard HTTP authentication prompts [32]. It invokes the use of many iterations to slow attacks and does not incorporate salt to strengthen passwords [33].

6. Conclusion

Authentication data, in its varying forms, is literally the key to vaults that contain the details of our lives. Not only are we dependent on databases to store the information required for our day-to-day existence and prove our very identities; we rely on them to store the data that allows access to this information. As demonstrated by the database breaches executed on a near daily basis, these stores are vulnerable to both online and offline attacks and therefore must be protected. Despite great advancement in the field of authentication, all data used in the authentication process is reliant upon databases for storage. This is inclusive of passwords, biometric data and elements of two and multi-factor authentication. Based on this premise, human kind will be reliant on secure storage within databases to protect their passwords and other types of authentication data for some time to come.

Password policy that enforces length, entropy and lifespan may provide safe guards against password guessing or brute-force attacks, but it cannot be relied upon as a defence against theft via a database breach. Other security mechanisms must be implemented to protect authentication data while it is being stored. Much of industry's authentication data is poorly stored in plaintext, due to the oversights of times gone by. This data is extremely vulnerable, as a simple attack like a SQL injection, could result in the theft of any number of useable passwords, potentially enabling criminals to execute greater and more lucrative crimes. The IEEE and IETF standards may be considered 'dated', based on date alone; but the documents still provide recommendations that are industry best practise. Currently, no other recommendations exist. Furthermore, these recommendations are confirmed and further endorsed in documents by leading security authorities such as *OWASP* and *NIST*. Although PCI-DSS documents do not stipulate how authentication data should be stored, they do state that it should be stored 'securely', offering no contradiction to the recommendations offered by the IEEE and IETF.

Best practice envelops an array of advisements ranging from password length and entropy, to concatenation of salt, one-way hashing and iterations. Despite these recommendations, best practice for both password policy and storage is often implemented incorrectly or not at all. This has resulted in vulnerable databases and vulnerable data. Methods that should be implemented to achieve best practice data are clearly outlined in the IEEE, IETF and NIST documents. Salt, one-way hashing, iterations and use of PBDKF2 should all be utilised with parameters that acknowledge the advancements in computer processing power. FIP approved SHA-256 one-way hashing is an ideal option for one-way hashing as it is un-cracked and possess a length of 256 bits. This supports the theory of increased entropy with increased length. Password manager applications have been developed that are intended strengthen passwords for various web accounts. Unfortunately, these applications are geared toward private individual users and do not provide for administrators who wish to apply corrective measures to existing authentication databases. Any remedial implementations on back-end infrastructure will most likely be at large expense regarding both time and money. For 'The Internet of Things' is to continue to develop, and eventually transform our lives, the issue of improper storage of authentication data must be addressed.

References

1. Gartner (2016) Gartner Says Worldwide Information Security Spending Will Grow 7.9 Percent to Reach $81.6 Billion in 2016. Gartner.com, August 9th 2016. https://www.gartner.com/newsroom/id/3404817

2. NCA (2016) Cyber Crime Assessment 2016. NCA Strategic Cyber Industry Group, http://www.nationalcrimeagency.gov.uk/publications/709-cyber-crime-assessment-2016/file

3. Ferrara, A. (2012) Properly salting passwords, the case against pepper http://blog.ircmaxell.com/2012/04/properly-salting-passwords-case-against.html

4. Miller,S., Curran, K., Lunney, T. (2016) Traffic Classification for the Detection of Anonymous Web Proxy Routing. *International Journal for Information Security Research*, Vol. 5, No. 1, March 2016, pp: 538 - 545, DOI: 10.20533/ijisr.2042.4639.2015.0061, ISSN: 2042-4639

5.IEEE (2008) IEEE Standard Specifications for Password-Based Public-Key Cryptographic Techniques. http://ieeexplore.ieee.org/document/4773330/

6. OWASP. (2015) Password Storage Cheat sheet, OWASP, https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet

7. Flores, D. (2014) An authentication and auditing architecture for enhancing security on eGovernment services. First International Conference on eDemocracy & eGovernment (ICEDEG), NY, USA, 24-25 April 2014 DOI**:** 10.1109/ICEDEG.2014.6819952,

8. Jain, A. (2005) Biometrics: Personal Identification in Networked Society (online), http://www.kennys.ie/biometrics-15.html

9. Tang, Y., Huang, D., Wang, Y. (2012) ID Proof on the Go: Development of a Mobile EEG-Based Biometric Authentication System, 21st International Conference on Pattern Recognition (ICPR), Tsukuba, Japan, pp: 45-54, 11-15 Nov 2012

10. Dubin, J. (2007) Complex password compliance requirements made simple (online), http://searchsecurity.techtarget.com/tip/Complex-password-compliance-requirements-made-simple

11. Ksiazak, P., Farrelly, W. & Curran, K. (2015) "Lightweight Authentication Protocol for Secure Communications between Resource-Limited Devices and Wireless Sensor Networks." *International Journal of Information Security and Privacy*, Vol. 8, No. 4, pp: 62-102, October 2015, DOI: 10.4018/IJISP.2014100104

12. Choi Y, Lee Y, Moon J, Won D (2017) Security enhanced multi-factor biometric authentication scheme

using bio-hash function. PLoS ONE12(5): e0176250. https://doi.org/10.1371/journal.pone.0176250

13. Jensen, B. (2015) 5 Myths of Password Security (online), https://stormpath.com/blog/5-myths-password-security/

14. Prince, M. (2012) The Four Critical Security Flaws that Resulted in Last Friday's Hack. Cloudflare, April 11 2016, https://blog.cloudflare.com/the-four-critical-security-flaws-that-resulte/

15. Rizzo, T. (2013) The Most Recent Password Security Compliance Guidelines http://insights.scorpionsoft.com/bid/329695/The-Most-Recent-Password-Security-Compliance-Guidelines

16. NIST (2016) NIST'S POLICY ON HASH FUNCTIONS, *National Institute of Standards and Technology*, available: http://csrc.nist.gov/groups/ST/hash/policy.html

17. Strom, D. (2014) Introduction to multifactor authentication methods in the enterprise. Search Security, July 2014 http://searchsecurity.techtarget.com/feature/The-fundamentals-of-MFA-Multifactor-authentication-in-the-enterprise

18. Abhishek, K., Roshan, S., Prabhat and, K., Rajeev, R. (2012) A Comprehensive Study on Multifactor Authentication Schemes. Proceedings of the Second International Conference on Advances in Computing and Information Technology (ACITY) July 13-15, 2012, Chennai, India, pp: 561--568

19. Schneier, B. (2013) A Really Good Article on How Easy it is to Crack Passwords. Schneier Blog, July 2013, https://www.schneier.com/blog/archives/2013/06/a_really_good_a.html

20. XKCD.com. (2013) Password Strength (online), http://xkcd.com/936/

21. Toponce, A. (2011) Strong Passwords NEED Entropy, ptree.org, 3rd july 2011 https://pthree.org/2011/03/07/ strong-passwords-need-entropy/

22. Pornin, T. (2013) Where did common minimum password length guidelines originate (online), http://security.stackexchange.com/questions/47098/where-did-common-minimum-password-length-guidelines-originate

23. Ross, B. et al. (2005) Stronger Password Authentication Using Browser Extensions. Proceedings of the 14th conference on USENIX Security Symposium, Volume 14, Baltimore, MD. July 31 - August 2005 pp: 22-32

24. Spafford, G. (2006) Security Myths & Passwords. Cerias Blog, August 2006 http://www.cerias.purdue.edu/site/blog/post/password-change-myths/

25. Halamka, J. (2012) A Really Good Article on How Easy it Is to Crack Passwords (online), http://geekdoctor.blogspot.ie/2012/12/the-quest-for-perfect-password.html

26. Barker, E. (2016) NIST Special Publication 800-57 Part 1 Revision 4 - Recommendation for Key Management, NIST, http://dx.doi.org/10.6028/NIST.SP.800-57pt1r4

27. IETF. (2000) PKCS #5: Password-Based Cryptography Specification Version 2.0, https://tools.ietf.org/html/rfc2898

28. OWASP. (2010) Hashing Java. OSWAP, https://www.owasp.org/index.php/Hashing_Java

29. Thavalengal, S., Andorko, I., Drimbarean, A., Bigioi, P. & Corcoran, P. (2015) Proof-of-concept and evaluation of a dual function visible/NIR camera for iris authentication in smartphones. IEEE Transactions on Consumer Electronics, Vol: 61, Issue: 2, May 2015, DOI**:** 10.1109/TCE.2015.7150566

30. Ristic, I. (2014) SHA-1 Depreciation: What you need to know (online), https://community.qualys.com/blogs/securitylabs/2014/09/09/sha1-deprecation-what-you-need-to-know

31. Beattie, D. (2014) Everything you need to know about the move to SHA-1 (online), https://www.globalsign.com/en/blog/everything-you-need-to-know-about-the-move-to-sha-256/

32. Strahs, B., Yue, C., Wang, H. (2009) Secure Passwords Through Enhanced Hashing. Proceedings of the 23rd conference on Large installation system administration, LISA'09, Baltimore, USA, Nov 1-6th 2009, pp: 7-14

33. Halderman, J., Waters, B., Felten, E. (2005) A Convenient Method for Securely Managing Passwords, Proceedings of the 14th international conference on World Wide Web (WWW 2005), Chiba, Japan, May 10-14 2005