A Novel Approach for Secure Identity Authentication in Legacy Database Systems

Juanita Blue [1, 2], Eoghan Furey [1], Joan Condell [2]

[1] Department of Computing, Letterkenny Institute of Technology, Letterkenny, County Donegal, Ireland
[2] Intelligent Systems Research Centre, University of Ulster, Derry, Northern Ireland, UK

Abstract— **Information systems in the digital age have become increasingly dependent on databases to store a multitude of fundamental data. A key function of structured databases is to house authentication credentials that verify identity and allow users to access more salient personal data. Authentication databases are frequently a target of attack as they potentially provide an avenue to commit further, more lucrative crimes. Despite the provision of industry standard best practice recommendations from organisations such as Open Web Application Security Project (OWASP), Payment Card Industry Security Standards Council (PCI-SSC), Internet Engineering Task Force (IETF) and Institute of Electrical and Electronics Engineers (IEEE), often practical security implementations within industry flounder. Lacking or substandard implementations have cultivated an environment where authentication databases and the data stored therein are insecure.**

This was demonstrated in the 2016 exposure of a breach experienced by Yahoo where approximately one billion user credentials were stolen. The global technology company was found to be using obsolete security mechanisms to protect user passwords. Dated implementations such as these pose serious threat as they render authentication data highly vulnerable to theft and potential misuse.

This paper offers a novel solution for securing authentication databases on non-compliant Apache servers. The method applies the recommended best practice mechanisms in the form of salt, one-way encryption (hashing) and iterations to both pre-existing and newly created passwords that are stored on insecure systems. The proposed solution can be implemented server-side, with little alteration to the existing infrastructure, unbeknownst to the user. It possesses the potential to improve system security, aid compliance, preserve privacy and protect users.

Keywords— passwords; salt; encryption, authentication; user-credentials

I. Introduction

According to Intel Security and the Centre for Strategic & International Studies (CSIS), "Cybercrime is a growth industry where the returns are great and the risks are low" In a report produced in 2014, the group estimated the global economic cost at more than 400 billion US dollars annually [1]. This figure was approximated based on documented personal loss, organisational theft, damage to professional reputation and also global economic effect.

Due to the sensitive nature of breaches, governments and organisations are reluctant to disclose the details of system vulnerabilities that have been key in each breach. Hence, there is difficulty in identifying an exact cost figure relating to authentication data theft. However, it can be stated that exposed authentication data often acts as a gateway to greater breaches [2]. Based on this premise, a substantial portion of the aforementioned sum could be attributed to insecure authentication data storage.

Usernames and passwords currently remain the most commonly implemented and widely accepted form of identity verification [3]. Prior to the present focus on mitigating risk of cybercrime, typical web application authentication provisions were central repositories now referred to as 'legacy databases'. Traditionally, the risk of breach was not contemplated and therefore passwords were frequently stored in plaintext [4]. It is claimed that a large number of these legacy databases remain in common use and still store authentication credentials in plaintext or by obsolete and substandard means [5].

Frequent and successful compromises of authentication databases such as eHarmony and LinkedIn [6] have indicated an urgent need for a server-side solution capable of rectifying the security shortfalls of existing legacy databases, with an objective of protecting any pre-existing and future passwords written to the database. Such a solution should provide a simple, convenient and affordable option that aids administrators in the mitigation of risks associated with compromised authentication data and user accounts.

II. Background

In seeking a solution to secure noncompliant legacy databases, various associated areas were explored. These included large scale legacy system breaches, best practice standards, state of the art authentication technologies, commonly implemented password policies and existing password strengthening applications.

A. Legacy Databases, Breaches & Best Practice Standards

An increase in incidents of cybercrime has forced industry to adapt and develop methods by which cybersecurity may be improved [7]. Expert groups such as OWASP have identified the protection of authentication credentials as chief in the prevention of additional crime [8]. Initial breaches occur largely by way of 'injections', an attack vector that tops the OWASP list of the 'Most Critical Web Application Security Risks', where attackers gain access to the content of databases [9]. Recent years have seen a plethora of breaches suffered by global corporations such as LinkedIn, Target, Ashley Madison and TalkTalk [10][7][11][12]. The end of 2016 alone saw the culmination of breaches experienced by large web-based companies such as Friend Finder, Yahoo and Hello Kitty. The web accounts linked to these compromised credentials all contained private personal data belonging to both adults and minors [13]14][15]. The associated web accounts stored personal data that included financial and residential details, leaving the registered users directly susceptible to additional crime.

Breaches such as these have highlighted the importance of securing authentication data, forcing a re-evaluation of the defences in place [6][7]. Adherence to standard guidelines such as the 'OWASP Password Storage Cheat sheet', the 'IEEE Standard Specifications for Password-Based Public-Key Cryptographic Techniques' and 'RFC 2898: Password-Based Cryptography Specification' is recommended to prevent theft and misuse of passwords. These documents offer guidance on the security mechanisms that must be invoked to protect stored authentication data. As depicted in Figure 1, in order to maintain the confidentiality and integrity of credentials, best practice guidelines advise that all passwords should be:

- Salted with a random string of not less than 32 bytes [16]
- Encrypted/hashed using a one-way, proven and un-cracked cryptographic hash function such as SHA-256 or SHA-512 [8][17]
- The output of this hash function should be re-hashed for no less than 1,000 iterations [18]

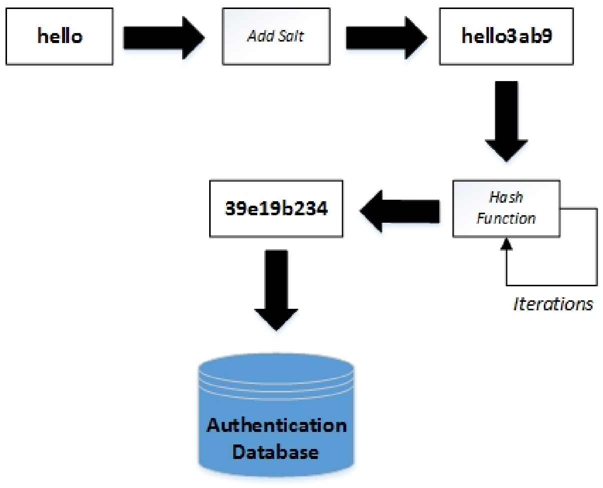

Fig. 1. Application of salt, hashing & iterations to password

The Payment Card Industry Data Security Standard (PCI-DSS), produced by the PCI-SSC, insists that at minimum, the above criteria must be met in order for organisations to be considered compliant [19]. However, despite a multitude of available information relating to best practice implementation, best practice and compliance still evade many organisations due to budget constraints, time factors and a reluctance to replace existing infrastructure [7][20][21].

Often, even organisations who attempt to adopt protective measures fail to implement best practice correctly [22]. Subsequent investigation into password breaches has revealed that several of the companies maintained substandard implementations [5]. Yahoo, Friend Finder and Hello Kitty's compromised databases were found to be storing passwords in plaintext or hashing with depreciated one-way encryption functions such as MD5 and SHA-128. Furthermore, all had omitted the use of salt and iterations within their password protection mechanisms [23][13][15]. The targeted organisations had not adhered to basic best practice recommendations and were therefore non-compliant. Had the recommended basic best practice mechanisms been implemented, the user credentials may have been protected from potential misuse.

It is improbable that organisations will openly declare that they maintain non-compliant implementations, as doing so compromises customer data, system security and company reputation. Based on cost alone it is unlikely that organisations globally will improve or replace their entire existing infrastructure in a bid to mitigate risk [10][20]. An increase in successful breaches and industry's failure to improve current infrastructure vociferates for a solution which utilizes an existing staple technology that can be discretely applied server-side by an administrator, incurring minimal cost, effort and downtime.

B. *State of the Art Technologies*

State of the art technologies such as biometric and multi-factor authentication were assessed for applicability. It was surmised that despite a considerable amount of current research and new developments, in view of the complexity, expense and unreliability of these more recent technologies and their hardware components, it is improbable that these solutions will be deployed for common practical use in the near future. Reliance upon passwords will remain [24].

C. *Common Implementation of Password Policy*

An examination of commonly implemented password security policies was conducted to establish if they increased security of user accounts. The implementation of policies that dictate length, entropy and lifetime of passwords are successful in mitigating risk of unauthorized access to singular accounts via brute force or a dictionary attack. However, passwords at rest in an authentication database remain vulnerable to exposure via an injection attack.

D. *Password Strengthening Applications*

Several password management applications including Password Agent, Lucent Personalized Web Assistant (LPWA), PwdHash and Password Multiplier were assessed for functionality[25][26][27][28]. Operating essentially as pluggable web browser extensions, these solutions offer password strengthening and management features that aid in securing user accounts. Despite implementing best practice recommendations, each solution operated client-side, therefore it was surmised that these applications were intended for private use by individuals wishing to strengthen authentication for assorted personal web accounts. These applications could not provide a valid solution for organisations to improve the security of password storage server-side. No commercial applications were discovered that assist administrators in applying best practice security measures to webserver/authentication database infrastructure.

III. METHODOLOGY

The following describes key methods and technologies that were invoked to develop, implement and test the proposed solution.

A. *Environment*

A suitable virtual Windows test environment was constructed that included an Apache web server. The server was combined with a MySQL database to store authentication data and PHP authentication files to render web interfaces and perform basic user account registration and login authentication functions, as shown in Figure 2. According to the Netcraft Web Server Survey of February 2016, open source Apache servers hold 32.80% of the market share of top servers on the internet [29]. This platform was selected based on popularity and common utilisation with online accounts.

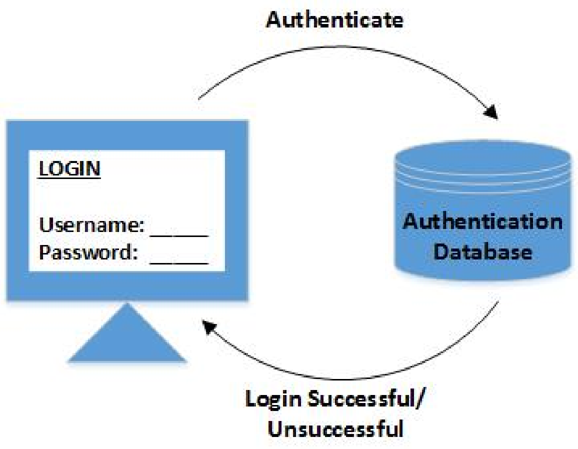

Fig. 2. Authentication via login interface verified via database

The basic environment emulated that of a typical legacy system. It contained no best practice mechanisms and no repository to store salt strings or protected passwords. The database simply contained a 'Users' table that stored account usernames and plaintext passwords.

B. Apache Rewrites

Apache mod_rewrite is a module that allows server-side manipulation of requested URLs. Incoming URLs are checked against a series of rules that contain a regular expression to detect a certain set of conditions. If the conditions are found to be true in the URL, the desired portion of the URL is replaced with a provided substitution string or action. This process continues until there are no more rules or the process is explicitly told to stop.

The *mod_rewrite* module can be utilized to manipulate URLs whilst they are being accessed. It has the capacity to translate complex URLs into a more human-readable format still understandable by a server. It can also be adapted to seamlessly redirect to other pre-constructed files that contain alternate functions; it was this capability that provoked its use as a key component in constructing a server-side authentication solution.

C. PHP Authentication Files

To implement a solution utilizing the features of the Apache *mod_rewrite* module, the server had to be configured to render the web interfaces from the original authentication files and then internally redirect all requests for authentication functions to alternate files. This was achieved by creating new authentication files that contained functions to salt, hash and iterate passwords for user accounts that were newly created and also pre-existing accounts where passwords were stored in plaintext. Upon rendering the web interface created in the original 'Login Page' file, instead of calling the functions from this file to conduct the authentication process, the system would redirect to a new file that would perform the functions to protect the password and then redirect back to the original file. The new authentication files were stored alongside the original files, in the HTTP folder on the web server.

To force the server to redirect to alternate files and execute newly implemented functions, the web server configuration file was altered to include a set of *mod_rewrite* rules that dictated the conditions under which a redirect should occur. The rules stipulated that when an attempt was made to access the login page, the web page should be rendered from the original file and then a redirect to the new protective file should occur. The redirect would allow for the salt, hash and iteration functions from the new file to be executed. Following the application of protective mechanisms and storage in the database, control would be passed back to the original file to finalise the authentication process, as shown in Figure 3. The switch to the new files was executed seamlessly, without the browser URL displaying any indication of a redirect.

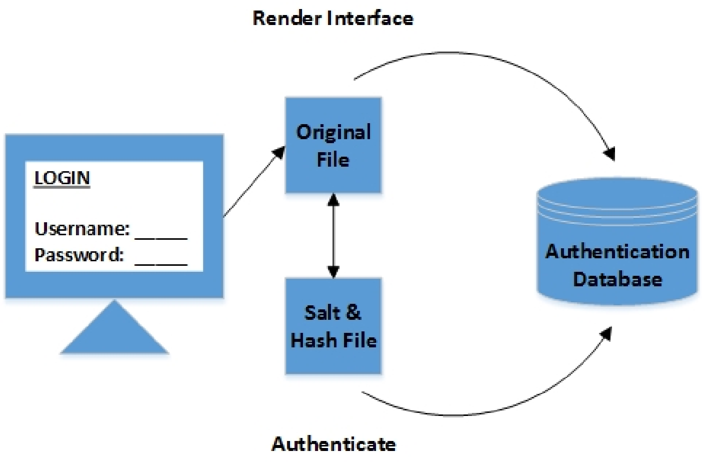

Fig. 3. Authentication with original and new PHP files

D. MySQL Database

The database present in the original insecure environment contained a single table to store usernames and plaintext passwords, with no column present to store randomly generated salt strings. The newly created authentication files were configured to assess if a salt table was already present in the database. If no table was present, one was created with columns to store the username, salt string and protected password for each user account. This process was triggered by any attempt to log in or register a new account. Upon creation of the salt table, the user's original plaintext password, stored in the pre-existing Users table was replaced with a randomly generated string of characters. Figure 4 depicts the new distribution of data.

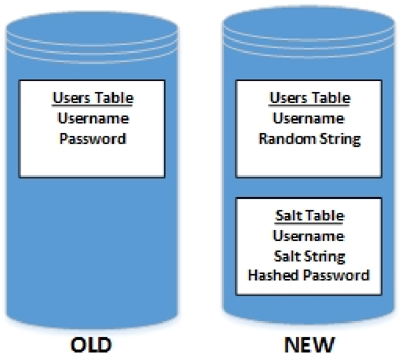

Fig. 4. Pre-existing database and new altered database with Salt table

Following the creation of the salt table, users who were logging in for the first time since the changes were invoked would have the aforementioned protective mechanisms applied to their credentials. Users who were conducting subsequent login attempts were authenticated initially based on their hashed password. When control was transferred back to the original authentication file, users were authenticated once again based on the random string present in the pre-existing user table. All newly created accounts automatically stored the username, salt string and hashed password in the Salt table and stored the username and randomly generated string of characters in the pre-existing Users table. Authentication was then conducted in the aforementioned manner. The purpose of the insertion of a random string in the password field of the original Users table was to remove the plaintext password and to confuse a would-be attacker with two potential passwords. The second password is authenticated automatically by the system without a user prompt, however it is important to note that authentication must be successfully executed using both passwords to be granted further access.

From this point forward, any subsequent successful authentication attempts resulted in the automatic protection of passwords stored in plaintext. At no point was the user prompted to actively change their password.

IV. Results

On invoking the use of Apache re-writes, the system had been successfully manipulated to:

- Create a salt table (if none existed) to store the username, salt and protected password
- Salt, hash and iterate all passwords for newly registered user accounts
- Salt, hash and iterate each plaintext password for pre-existing user accounts when users successfully logged in
- Successfully authenticate for both new and old accounts

A test environment was constructed whereby the rendered webpages were accessed remotely from a client machine on the same network. The success of this implementation was explored through results relating to functionality, viability, reliability and security.

A. Functionality

The functionality of the implemented solution was assessed via login attempts from a remote client on the same network. While authenticating, the *mod_rewrite* module was activated and successfully redirected to new authentication files and applied best practice mechanisms to passwords without any changes to the URL in the browser bar. Essentially, this indicated that the redirects had occurred unbeknownst to the client/user. Redirects were further verified by the presence of the username, salt string and protected password within the Salt table.

The redirects were not found to raise any session alerts or flag any suspicious activity on the server or client systems. This was due to the fact that the redirects were conducted server-side, before the client received any data relating to GET and POST requests. From the server's perspective, the *mod_rewrite* module was simply meeting the conditions and rules stipulated in the server configuration file and carrying out the functions as defined in the new authentication files. From the client's perspective, the system was simply rendering the pages received following GET requests and returning user input to the server in the form of POST requests. This indicated that the solution could be implemented without any negative ramifications on the client or server systems. Table I depicts the impact of redirects on invoked security devices and software applications, the solution could be implemented without experiencing any interference from infrastructural security implementations.

Table I. Redirect Alerts by Security Devices & Software

Device/Software Type	*Alert Raised*	*Redirect Successful*
Web Browser Security Features	✗	✓
Intrusion Detection System (IDS)	✗	✓
Intrusion Prevention System (IPS)	✗	✓
Firewall	✗	✓
Antivirus Software	✗	✓

The solution was also trialed with several popular web browsers including Google Chrome, Microsoft Internet Explorer and Mozilla Firefox. Redirects were found to be successful with all the browsers tested, thus, the solution was proven to be browser independent.

Overall these assessments indicated that the solution could potentially be implemented without any negative impact on the functionality of the client or server systems. It was also noted that the original authentication process carried out by the legacy system was neither truncated nor discarded. The system still conducted the same sequence of steps that were present prior to the new implementation; However, the solution had added additional steps to the authentication process that allowed for new and old passwords to be protected. Not only were passwords being protected to best practice recommendations, but two successful authentication sequences were subsequently required to access user accounts.

B. Viability

The implemented solution was found to be a viable server-side option for administrators seeking to apply best practice recommendations to legacy authentication databases. The solution successfully implemented security mechanisms in the form of salt, hashing and iterations to protect new and old passwords, thus securing user accounts.

This was achieved without incurring any additional cost of overhauling the system or installing new software. The results indicated that the method invoked is a viable, cost effective solution that could be introduced to secure existing systems that store passwords by substandard means. Furthermore, the solution could be implemented with relatively little downtime. Table II below outlines the infrastructural changes required to implement the solution.

Table II. Infrastructural Changes for Implementation

Component	*Change Required*
PHP	Two new files created
HTTP	Activate *mod_rewrite* module Add code to configuration file
MySQL	Generation of Salt table (Automatic)

It must be noted that the solution requires implementation by an individual who is competent in system configuration, PHP and MySQL. This is based on the premise that currently the solution must be implemented manually. There is potential for the necessary alterations to be executed by a pre-defined script; however this requires the parameter and variable names to be included in the script. Creation of the script would entail knowledge of the system in place, or temporary introduction of monitoring software that could identify and communicate the data required to automate the process.

This solution is currently only suitable for Apache web servers as the primary technology invoked is the Apache *mod_rewrite* module. It is possible that the solution could be adapted for alternative platforms, provided the given server possesses a similar URL rewrite function.

C. Reliability

The implemented solution was found to be completely reliable and thus, the method could be permanently invoked. The experiments conducted demonstrated that redirects and protective PHP functions would continue to be executed as long as the rewrite conditions and rules were present in the web server configuration file.

D. Security

To test the security of the implemented solution, an attack in the form of an SQL injection was executed on the system. This attack was designed to extract usernames and passwords from the database so that they may theoretically be used for other purposes.

To facilitate the attack, a dynamic SQL statement had been deliberately placed within the authentication files. This was included as a feature of the original 'insecure environment'. It is typically considered bad practice to include dynamic statements, as they can increase the attack surface of a system, rendering it vulnerable to injections. Despite the recommended use of prepared statements, this secure coding practice is often omitted due to poor programming skills, technical requirements or repeated re-working of code. Use of prepared statements reduces risk but does not entirely mitigate injections as an attack vector.

The SQL injection was successfully executed and resulted in the extraction of all the information contained within the Users and Salt tables.

Successful penetration of the database unequivocally demonstrated the following:

1. The legacy test environment was insecure and vulnerable to attack.
2. The unprotected pre-existing plaintext passwords stored in the Users table were vulnerable and could easily be compromised and potentially misused.

The SQL injection extracted all usernames and passwords present in both tables. Whilst the plaintext passwords required no decipher process and could be immediately utilised, the new hashed passwords would be considered virtually worthless. Significant processing would be required to restore the passwords to a usable form, necessitating a rainbow tables attack to un-iterate, un-hash and un-salt each individual password string. Given the inclusion of the cryptographically secure SHA-256 hashing mechanism, this process would be extremely complex and time consuming, if even possible at all. Table III outlines the potential attack vectors where the solution has mitigated risk.

Table III. Attack Vectors Mitigated by Implemented Solution

Attack Vector	***Risk Mitigated***
Injection	✓
Security Misconfiguration	✓
Sensitive Data Exposure	✓
Using Known Vulnerable Components	✓
Misuse of Authentication Credentials	✓
Brute Force Attack	✓
Rainbow Tables Attack	✓

Despite National Institute of Standards and Technology (NIST) depreciation of hashing methods such as SHA-128 and MD5, the mechanisms remain commonly implemented. In February 2017, Google discovered the first official SHA-128 collision, conclusively identifying the function as insecure [30]. In systems where SHA-128 is utilized, the proposed solution may be invoked to over-ride the noncompliant implementation and aid migration to a Federal Information Processing Standard (FIPS) approved secure hash standard such as SHA-256 or SHA-512.

V. CONCLUSION

Overall, the protective mechanisms introduced by the implemented solution were found to greatly increase the security of the system. As demonstrated by the SQL injection, unprotected passwords stored in the Users table were easily extracted and thus were potentially available for nefarious use. The passwords that had undergone the salt, hash and iteration process maintained integrity and confidentiality. These passwords would require reverse engineering to be converted back to a usable plaintext state.

In a case where passwords have been successfully hashed, but salt has been omitted, a rainbow tables attack could still potentially decipher the plaintext password. Concatenation of salt and iterated hashing are crucial in placing large demands on the resources of an attack system and complicating the decipher process.

The retention of the original authentication function contributed to the improvement of the system's overall security. Although the enhanced system authenticated based on the hashed password stored in the Salt table, the original authentication sequence also had to be executed for a user session to be initiated.

The concept of adding randomly generated hexadecimal strings to the password column in the original Users table served several purposes. These included the continued execution of the pre-existing authentication process, the removal of the plaintext password from the database and it also sought to confuse a would-be attacker. If the contents of both tables were obtained, the random hexadecimal strings would appear to be hashed passwords. In theory the attacker could spend lengthy periods trying to un-hash these individual strings by way of a rainbow tables attack; but this would be impossible, as the inserted strings were randomly generated.

The attacker would also be unaware that the system authenticates twice, initially based on the hashed password present in the Salt table and then on the random string present in the Users table. Without the plaintext password matching the hashed version stored in the Salt table, the random password stored in the Users table is rendered useless. An understanding of this nested process would require familiarization with the content of the authentication files.

The rewrite implements security mechanisms that comply with best practice recommendations, including 32 bytes of salt, one-way hashing using SHA-256 and no less than 1,000 iterations [16][17][18]. Inclusion of these best practice mechanisms meets compliance recommendations outlined by organisations such as the IEEE, IETF, PCI-SSC and OWASP, in compliance with documents such as PCI-DSS and HIPAA (Health Insurance Portability and Accountability Act). These recommendations are considered basic requirements of secure authentication data storage, yet all recommended mechanisms can potentially be improved. Stronger measures could be easily incorporated, however enhanced security may result in latency when the server is experiencing high demand. Increased salt bytes, longer hashing functions and additional iterations all place higher demands on system resources.

Overall, the solution does not appear to have introduced any additional security risks. Vulnerabilities present in the system are inherent to Apache, PHP and MySQL, and would be present regardless of the implemented solution.

Breaches like that experienced by Yahoo not only compromised Yahoo account holders, but customers of large corporations such as British Telecom, AT&T, SBCGlobal, Verizon.net and BellSouth, who all relied on Yahoo for their customer email service. Had Yahoo implemented a solution such as this to over-ride their use of the depreciated MD5 hash function, the compromise of approximately one billion user credentials could have been prevented.

The application of this solution has been successful in improving security on a highly vulnerable system, with few negative ramifications. This implementation has proven successful in encouraging a shift toward compliance with several standards including PCI-DSS, HIPAA and Data Protection Act requirements, improving system security, aiding compliance, preserving privacy and protecting users.

REFERENCES

[1] Centre for Strategic & International Studies, “Net Losses: Estimating the global cost of cybercrime: Economic impact of cyber crime II”, McAfee. Santa Clara, CA,vol. 61162, pp. 2-3, June 2014.

[2] R. Hasan and W. Yurcik, “Beyond Media Hype: Empirical Analysis of Disclosed Privacy Breaches 2005-2006 and a DataSet/Database Foundation for Future Work”, National Centre for Supercomputing Applications (NCSA), University of Illinois, Urbana, IL, 2007

[3] B. Ur, J. Bees, S. Segreti, L. Bauer, N. Christin and L. Cranor, “Do Users’ Perceptions of Password Security Match Reality?”, Carnegie Mellon University, Pittsburg, PA, Association for Computing Machinery (ACM), Santa Clara, CA 2016

[4] J. Graham, J. Hieb and J. Naber, “Improving Cybersecurity for Industrial Control Systems”, True Secure Scada LLC, J.B Speed School of Engineering, Louisville, KY, IEEE 25th International Symposium on Industrial Electronics, IEEE, Santa Clara, CA, 2016

[5] S. Karod, N. Sharma and A. Sharma , “An Improved Hashing Based Password Security Scheme Using Salting and Differential Masking”, 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO), IEEE, Noida, India, December 2015

[6] D. Mirante and C. Justin, “Understanding Password Database Compromises”, Dept. of Computer Science and Engineering Polytechnic Inst. of NYU, Tech. Rep. TR-CSE-2013-02, 2013

[7] The OWASP Foundation, “Password Storage Cheat Sheet”, https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet#Introduction, August 2016

[8] N. Manworrena, J. Letwatb and Olivia Dailyc, , “Why you should care about the Target data breach”, Business Horizons, Kelly School of Business, Indiana , Elsevier, Vol. 59, Issue 3, pp. 257-266, 2015

[9] The OWASP Foundation, “Top 10 – 2013: The ten most critical web application security risks”, https://www.owasp.org/images/f/f8/OWASP_Top_10_-_2013.pdf, 2013

[10] I. Erguler, “Achieving Flatness: Selecting the Honeywords from Existing User Passwords”, IEEE Transactions on Dependable and Secure Computing, IEEE, Vol.13, Issue 2, 2016

[11] D. Horn and S. Nair, “The Prom Problem: Fair and Privacy-Enhanced Matchmaking with Identity Linked Wishes, IEEE, Orlando, 2016

[12] R. Kozik, M. Choraś, W. Hołubowicz and R.Renk, “Extreme Learning Machines for Web Layer Anomaly Detection”, Image Processing and Communications Challenges 8. IP&C 2016. Advances in Intelligent Systems and Computing, vol 525. Springer, Cham, October 2016

[13] H. Graupner, D. Jaeger, F. Cheng and C. Meinel, “Automated parsing and interpretation of identity leaks”, Hasso Plattner Institute, University of Potsdam, Germany, ACM, May 2016

[14] L. Trautman, and P. Ormerod, “Corporate Directors’ and Officers’ Cybersecurity Standard of Care: The Yahoo Data Breach”, American University Law Review, Vol. 66, February 2017

[15] T. Spring, “Hello Kitty database of 3.3 million breached credentials surfaces”, Threat Post, https://threatpost.com/hello-kitty-database-of-3-3-million-breached-credentials-surfaces/122932/, January 2017

[16] N. Ferguson, B. Schneier and T. Kohno, “Cryptography Engineering: Design Principles and Practical Applications”, Wiley Publishing Inc., Indianapolis, IN, Chapter 21, pp 304, 2010

[17] Information Technology Laboratory, “Secure Hash Standard (SHS)”, Federal Information Processing Standards, Gaithersburg, MD, FIPS PUB 180-4, August 2015

[18] B. Kaliski, “PKCS #5: Password-Based Cryptography Specification Version 2.0”, IETF, Network Working Group, RFC 2898, 2000

[19] Payment Card Industry Security Standard Ccouncil, “Payment Card Industry (PCI) Data Security Standard”, Version 3.2, April 2016

[20] A. Schneider, “When companies become prisoners of legacy systems”, The Wall Street Journal, http://deloitte.wsj.com/cio/2013/10/01/when-companies-become-prisoners-of-legacy-systems/, October 2013

[21] Ron Condon, “Is it time to replace passwords with more complex authentication tools”, Computer Weekly, pp026-30, September 2010

[22] K. Brown, “The Dangers of Weak Hashes, “SANS Institute InfoSec Reading Room, November, 2013

[23] B. Krebs, “My Yahoo account was hacked! Now what?”, Krebs On Security, https://krebsonsecurity.com/2016/12/my-yahoo-account-was-hacked-now-what/, December 2016

[24] P. Grassi, M. Garcia, J. Fenton, “DRAFT NIST Special Publication 800-63-3 Digital Identity Guidelines”, National Institute of Standards and Technology, Los Altos, CA, February 2017

[25] B. Strahs, C. Yue and H. Wang, “Secure Passwords Through Enhanced Hashing”, The College of William and Mary, VA, 2009

[26] D. Kristol, E. Gabber, P. Gibbons, Y. Matias and A. Mayer, “Design and Implementation of the Lucent Personalized Web Assistant LPWA”, Information Sciences Research Center, Bell Laboratories, Lucent Technologies, Murray Hill, NJ, June 1998

[27] B. Ross, C. Jackson, N. Maiyake, D. Boneh, J. Mitchell, “Stronger Password Authentication Using Browser Extensions” Dept. of Computer Science, Stanford University, Stanford, CA, 2005

[28] J. Halderman, B. Waters and E. Felten, ”A Convenient Method for Securely Managing Passwords”, Princeton & Stanford Uni, 2005

[29] Netcraft, “February 2016 web server survey” (online), https://news.netcraft.com/archives/2016/02/22/february-2016-web-server-survey.html, February 2016

[30] M. Stevens, E. Bursztein, P. Karpman1, A. Albertini and Yarik Markov, “The first collision for full SHA-1”, CWI Amsterdam, Google Research, February 2017