IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 18, Issue 4, Ver. I (Jul.-Aug. 2016), PP 54-58
www.iosrjournals.org

An Enhanced Password-Username Authentication System Using Cryptographic Hashing and Recognition Based Graphical Password

Tivkaa, M.L.[1]; Choji, D. N.[2]; Agaji, I.[3]; Atsa'am, D.[4]
[1, 2, 3, 4]*(Department of Mathematics/Statistics/Computer Science, University of Agriculture, Makurdi, Nigeria)*

Abstract: *Password-username authentication is a critical component of today's web application technology that is commonly used to control access to restricted resources. However, poor design, coding flaws and weak user login credentials exposes this functionality to Sequel Query Language Injection (SQLI) and online password guessing attacks. Current techniques advanced by researchers to address authentication attacks only focus on either one of them, thus failing to envisage a scenario where the login form can be used to launch both SQLI and online password guessing attacks. To address this challenge, this paper presents an authentication solution that addresses the issue of SQLI and online password guessing attacks on login form as implemented in generic web applications. The solution combines the use of plain text credentials that are cryptographically hashed at runtime with recognition based graphical login credentials. The goal is to always guarantee access to a user account even when such account is under attack while at the same time ensuring convenient and secure login experience by legitimate users. This is achieved by blocking the Internet Protocol (IP) addresses from which there are unsuccessful login attempts. Security test shows that the solution is not vulnerable to SQLI and online password guessing attacks.*

Keywords: *Authentication, Password Guessing, Graphical Password, SQL Injection, Web Application Security*

I. Introduction

Web applications have become popular internet services and are critical to the survival of many enterprises that rely on them. To restrict access to privilege information stored on the web, various authentication schemes such as biometrics, unregistered user requirements, public key cryptography, keystrokes dynamics, click pattern, graphical passwords, one-time password, digital signatures, authentication panel, zero-knowledge proof as well as password/username are implemented in various web applications [1][2]. However, the most commonly used method of authentication is the password/username combination which is implemented using traditional HTML login form with input fields that allow users to enter their username and text based password [1][3].

Though prominent, the vulnerabilities associated with the implementation of Password/Username authentication pattern in web applications include the threat of password guessing and Sequel Query Language Injection (SQLI) attacks [4][5]. Various studies have shown that the cause of SQLI is largely poorly sanitized input from the users [6][7]. Password guessing attacks on the other hand are usually successful against weak passwords, poor password enforcement policies, and poor design of the authentication functionality as well as its implementation [8].

To address the problem of SQL Injection Attacks (SQLIAs), researchers have advanced several techniques ranging from defensive coding best practices to automated frameworks for detection and prevention of these forms of attacks [9]. In the same way, techniques such as account lock out, Completely Automated Public Turin Test to Tell Computers and Humans Apart (Captcha) and the use of graphical passwords have been proposed by researchers to solve the problem of online password guessing attacks on login forms [8][10][11]. Though these techniques are contextually efficient, they fail short of combining SQLI and password guessing attacks in one solution, thus researchers have failed to preconceive the possibility of an attacker using the login form to launch both SQLI and online password guessing attacks.

II. Related Works

2.1 Password Guessing Attacks

A password guessing attack is an attempt by a malicious user to gain unauthorized access to an application resource by using a repeated set of large word lists to guess login credentials [4]. These attacks manifest in various forms such as manual, brute-force and dictionary attacks. The cause of password guessing attacks is the use of weak passwords as well as poor password implementation policies. To defend against this threat, [12] proposed a system known as the Password Guessing Resistant Protocol. This approach sought to minimize the use of Completely Automated Public Turing Test to tell Computers and Human Apart (Captcha).

It enforces Captcha after a limited failed login attempts from a computer that is known to the system or user which according to the technique is identified by their IP addresses as well as cookies sent by the browser which are stored as white-list on the server. Though this approach will slow down the rate of attack, it has a drawback of associating a legitimate user with so many IP addresses that in the long run will be difficult to manage. Similarly, if the user uses different browsers or more than one operating system on the same machine then it will be difficult to effectively identify the user in all cases. The identity of the user can also be manipulated if browser cookies are altered by way of cookie theft.

In a similar way, [13] proposed a hybrid graphical password system that requires the user who has registered with a username, password and a graphic image to provide the username, password and redraw the graphic image to match with the stored graphic during the process of authentication. However this system will be easy to use for the technology savvy users, it will be a challenge for non-technology friendly individuals. Another drawback of the system will be the challenge of drawing the graphics to exact coordinates as the stored one. Another issue with this approach is that it is better suited for smart screens making it difficult for users who do not have such devices to gain authentication.

Another obvious method quite often deployed by developers to mitigate online password guessing attack is the use of account lock out security pattern [10]. Account lockout simply means denying access to a specific account after a failed number of incorrect password attempts. Account lockout can last a specific duration, such as one hour, or the affected account could remain blocked until manually unblocked by the account administrator. Wrongful implementation of this strategy could lead to a manifestation of other forms of attacks such as denial of service (DoS). Also if this functionality is not implemented properly it can be a fertile place for reconnaissance.

2.2 SQL Injection Attacks

SQL Injection is an attack vector that allows a malicious user to inject harmful SQL query statements that are executed by the application logic [6][11][14]. The cause of SQL Injection attack is when the application is designed to process user data without proper validation. SQLIAs represent one of the main security issues in database driven web applications [15]. According to Open Web Application Security Project [16], SQL Injection attacks remains one of the most dangerous attack vectors today and is on top of their annual report. SQL injection is mainly launched through web forms such as login functionality and altering query string parameters. There are various forms of SQLIAs such as tautology, piggy-backed, union queries and blind injection attacks [17][11][6]. Though their attack methodologies vary, the intent is the same: Data theft, Damage to corporate image, Denial of Service etc.

To address the challenge of SQLIAs, [18] proposed a technique that checks user credentials against encrypted values against placeholders in stored procedures. This technique is efficient in attacking code injection, however the technique is based on the assumption that the developer is approaching the development in a particular way especially in the coding of stored procedures, if they are written insecurely, then the approach will be ineffective, again it gives flexibility to the attacker to launch another attack vector such as password guessing attacks.

Another method that is often applied in solving SQLIAs is the application of defensive coding techniques such as pattern matching, input validation etc. [19] argue that the best approach in tackling these vulnerabilities is the use of suitable and applicable defensive and secure coding practices such as proper input sanitization, use of parameterized queries, guiding against input sources, input type checking and hiding default application errors. This explains that secure code is rooted in the behavior, orientation and awareness of developers about security issues as it relates to web applications they develop. This is indeed a convincing reason for proper usage of defensive coding techniques.

Most of these defensive coding practices come at the expense of performance and ease of use which is largely due to the fact that security is treated as add-ons rather considering it as part of the goal of the system right from the design stages [20].

III. Proposed Technique

To mitigate online password guessing and SQLI attacks on our authentication functionality, we present a technique that combines the use of cryptographic hashing algorithm, recognition based graphical password and parameterized queries. A randomly generated salt which is stored in the user's database table is combined with the user's chosen plain text password to produce and save the final encrypted password at the plain text registration phase. A user is also expected to setup his graphical login details as a second stage in the registration process by selecting an image category and choosing a specific image under such category to complete the registration phase. During authentication a user is allowed two login attempts using his plain text credentials, the password is computed at runtime using the salt generated at the registration phase. After two failed attempts, a user is allowed to login using his graphical password. The user who is identified by IP address is subsequently blocked after one failed graphical login attempt. The entire algorithm is summarized in Fig. 1 below:

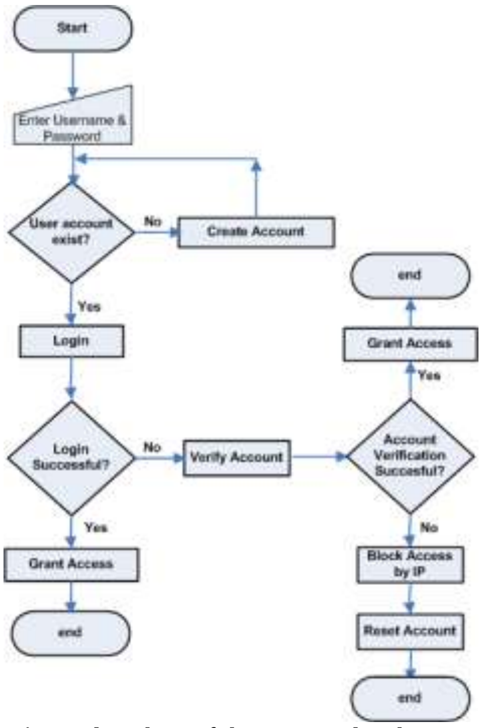


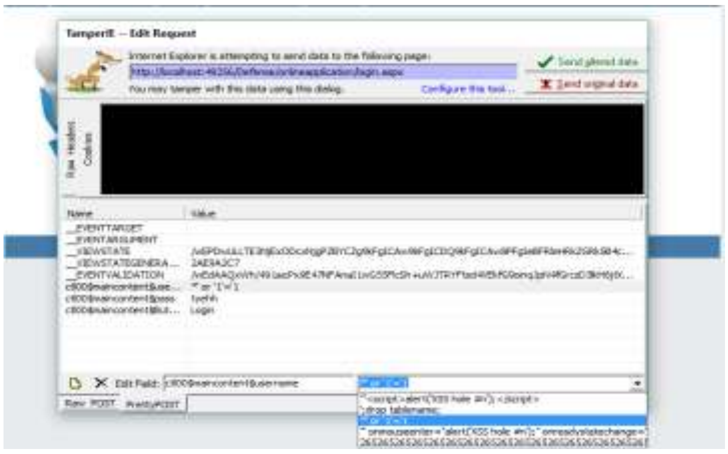
Fig 1: Flowchart of the proposed technique

IV. Materials/Methods

The proposed solution used the Microsoft Secure Development Life-Cycle as the software development model. The program structure was designed using the .NET Code-Behind programming structure while ADO.NET 2.0 was used as the underlying data access logic and MS SQL served as the database server. Microsoft threat modeling methodology was used to model the security of the authentication functionality. To achieve this effectively, Microsoft Threat Modeling Tool (TMT) was used to draw the application Data Flow Diagrams (DFDs) which represent different component of the application. The TMT then classified threats using the in-built Microsoft STRIDE model and rated them using the in-built DREAD model. The proposed authentication solution has been tested using TamperIE to evaluate the effectiveness of the solution.

V. Results

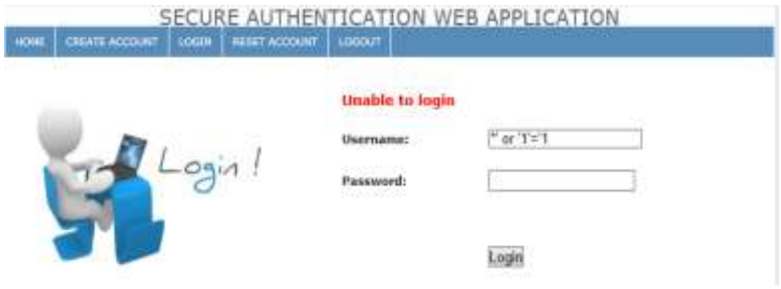
In this section, we present the result obtained by implementing the proposed technique, which is deployed locally as a web application over a secured communication channel using Secure Socket Layer (SSL).



Name	Value
__EVENTTARGET	
__EVENTARGUMENT	
__VIEWSTATE	[illegible]
__VIEWSTATEGENERA...	2AE8A2C7
__EVENTVALIDATION	[illegible]
c800$maincontent$use...	*' or '1'='1
c800$maincontent$pass	[illegible]
c800$maincontent$but...	Login

Figure 2: SQLI attack scenario

Fig. 2 shows a SQL attack on the login form. Upon submission of a login form, TamperIE automatically intercepts the data as shown in Fig. 2 without sending it to the server. To manipulate the name pair values in the TamperIE read/write mode, * 'OR '1'= '1 which is a tautology attack is selected and ran against the login form by clicking "send altered data" button at the top right corner. Several other SQLI constructs in the TamperIE were also selected and ran against the login form. The result was unsuccessful SQLI automated attacks as seen in Fig. 3.

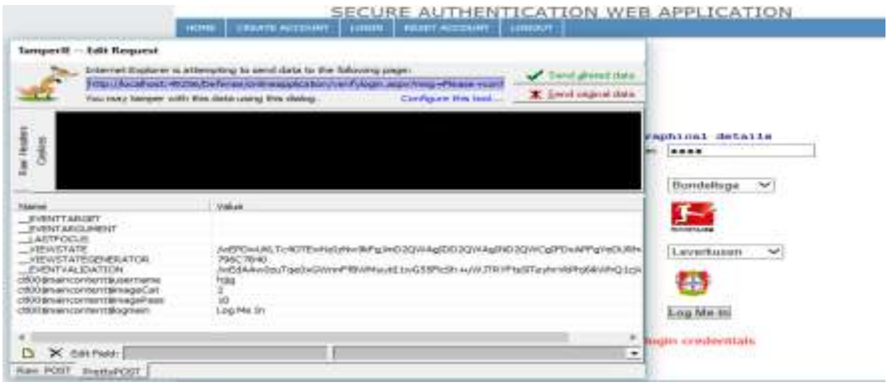


Unable to login

Username: " or '1'='1

Password:

Figure 3: Failed SQLI attack



Internet Explorer is attempting to send data to the following page:
http://localhost:[illegible]/Defense/onlineapplication/verifylogin.aspx?msg=Please+[illegible]
You may tamper with this data using this dialog.

Name	Value
__EVENTTARGET	
__EVENTARGUMENT	
__LASTFOCUS	
__VIEWSTATE	[illegible]
__VIEWSTATEGENERATOR	796C7840
__EVENTVALIDATION	[illegible]
ctl00$maincontent$username	hjg
ctl00$maincontent$imageCat	2
ctl00$maincontent$imagePass	10
ctl00$maincontent$login	Log Me In

raphical details

Bundesliga

Leverkusen

Figure 4: Graphical login page attack

Fig. 4 shows TamperIE used to manipulate the graphical login page by changing image Identity (ID) numbers in an automated attack. This attack scenario is unsuccessful because only one login attempt is allowed using the graphical login details. It prevents a malicious user from launching effective dictionary or brute force attacks on a login functionality based on the user logged IP address, this forces the user to request a password reset while the account is locked against the suspected IP address.

This security feature is important because it does not prevent a legitimate user from accessing his account even when an account is locked against an IP address. This signifies the failure of Denial of Service (DoS) through the login functionality. However it makes life difficult for an attacker to launch password guessing attacks unless the attacker changes IP address which is a cumbersome endeavour in the circumstance.

VI. Conclusion

The paper presents an authentication solution that combines the use of traditional password-username with graphical image login credentials. The solution gives the user the flexibility and convenience of use while at the same time maintaining the security considerations regarding the design of the solution. The test result showed that the use of threat modeling in the design of an application had tremendous impact on the application security, this is because security considerations were part of the development process.

The result showed that the application was not vulnerable to SQLI and online password guessing attacks. Remarkably, it was able to block malicious users based on their IP addresses but allowed access to legitimate users. This technique can be used in web applications that use password-username authentication scheme. In the future a solution will be proposed to enable the application learn automatically from previous attacks.

References

[1]. Kiiski, Lauri. "Security Patterns in Web Applications." *Publications in Telecommunications Software and Multimedia Laboratory, Available at: http://www. tml. tkk. fi/Publications/C/25/papers/Kiiski_final. pdf (Last Accessed: November 2011)* (2007).

[2]. Jesudoss, A, and N Subramaniam. "A Survey on Authentication Attacks and Countermeasures in a Distributed Environment." *Indian Journal of Computer Science and Engineering*, 5(2) (2014).

[3]. Kienzle, Darrell M, and Matthew C Elder. "Final technical report: Security patterns for web application development." *DARPA, Washington DC* (2002).

[4]. Dave, Mr Sachin R, and Vaishali B Bhagat. "Defecating online password guessing attack using 3 tier security." *International Journal of Application or Innovation in Engineering & Management* 2(12) (2013)

[5]. Kindy, Diallo Abdoulaye, and Al-Sakib Khan Pathan. "A detailed survey on various aspects of sql injection in web applications: Vulnerabilities, innovative attacks, and remedies." *arXiv preprint arXiv:1203.3324* (2012).

[6]. Gandhi, Mihir. "JwalantBaria, s "SQL Injection Attacks in Web Application" *International Journal of Soft Computing and Engineering 2(6)* (2013).

[7]. Balasundaram, Indrani, and E Ramaraj. "An authentication mechanism to prevent SQL injection attacks." *International Journal of Computer Applications, 19 (1)* (2011): 30-33.

[8]. Pinkas, Benny, and Tomas Sander. "Securing passwords against dictionary attacks." *Proceedings of the 9th ACM conference on Computer and communications security* 18 Nov. 2002: 161-170.

[9]. Halfond, William GJ, and Alessandro Orso. "Detection and prevention of sql injection attacks." *Malware Detection* (2007): 85-109.

[10]. Scambray, Joel, Mike Shema, and Caleb Sima. Hacking exposed: Web applications. San Francisco: McGraw-Hill, 2006.

[11]. Kumar, K. V. & Das, D. J. "Advanced Detecting and Defensive Coding Techniques to Prevent SQLIAs in Web Applications: A Survey." *International Journal of Science and Modern Engineering (6)* (2013)

[12]. Garg, Nitin, Raghav Kukreja, and Pitambar Sharma. "Revisiting Defences against Large Scale Online Password Guessing Attacks." *International Journal of Scientific and Research Publications* 3.4 (2013).

[13]. Khan, Wazir Zada, Mohammed Y Aalsalem, and Yang Xiang. "A graphical password based system for small mobile devices." *arXiv preprint arXiv:1110.3844* (2011).

[14]. Shehu, Bojken, Aleksander Xhuvani, and Shqiponja Ahmetaj. "Methods of Identifying and Preventing SQL Attacks" *International Journal of Computer Science* 9(6) (2012).

[15]. Ali, Shaukat, Azhar Rauf, and Huma Javed. "Sqlipa: An authentication mechanism against sql injection." *European Journal of Scientific Research* 38(4) (2009): 604-611.

[16]. Wichers, Dave. "OWASP Top-10 2013." *OWASP Foundation, February* (2013).

[17]. Anley, Chris. "Advanced SQL injection in SQL server applications." 31 Jan. 2002.

[18]. Rani, D.R., Kumar,B.S., Rao, T.R., Jagadish, V.T. and Pradeep, M., (2012) Web Security by Preventing SQL Injection Using Encryption in Stored Procedure. *International Journal of Computer Science and Information Technologies* 3(2) (2012):3689-3692

[19]. Borade, Monali R, and Neeta A Deshpande. "Extensive Review of SQLIA's Detection and Prevention Techniques." *International Journal of Emerging Technology and Advanced Engineering* 3(10) (2013).

[20]. Yee, Ka-Ping. "Aligning security and usability." *IEEE Security & Privacy* 5 (2004): 48-55.