Dynamic Password Authentication : Designing Step and Security Analysis

Detchasit Pansa
Department of Information Technology
Faculty of Informatics, Mahasarakham University
Mahasarakham, Thailand
E-mail:detchasit@msu.ac.th

Thawatchai Chomsiri
Department of Information Technology
Faculty of Informatics, Mahasarakham University
Mahasarakham, Thailand
E-mail:thawatchai@msu.ac.th

Abstract

This research purposively proposes the Dynamic Password Authentication procedures to be practically applied with various platforms e.g. web applications, network devices, and mobile applications. The researchers presented the overall design and the developed design with stronger security. Besides, the system analysis was tested for Security, Speed, and "Ease of Use".

Based on the study and testing, it was found that the authentication system designed by the researchers can usefully be applied for various purposes i.e. (1) to replace an authentication on website without using HTTPS to reduce an expense on CA; (2) to be applied on network devices or mobile applications to secure the password sending process and to prevent the password from being cracked. Firmly, this authentication method had been tested and compared with other protection systems and it was qualified as a highly effective system.

Keywords *– Dynamic Password, Authentication, ARP, Sniffer;*

1. Introduction

Nowadays, the computer systems need to work on a network; meanwhile, the first step to communicate is sending a password to claim for an individual user's authentication. Similarly, the Web Server - Web Client (browser) communication requires a password to be sent on the internet, whilst the Database Server - Database Client communication requires the password sending on LAN (Local Area Network). Also, the communication between routers requires an authentication system (e.g. using CHAP [1] in Cisco Router). Typically, the password being sent from one station to another could possibly be captured. Although the complicated encoding processes can secure the password, it could frequently be decoded. According to the previous studies, the password capturing and decoding, or password cheating could be classified as follows.

1.1 Password Cheating on Web Application

To cheat or capture a password being sent by a web client to a web server could be done by various means e.g. DNS Spoofing or attacking to the weakness of DNS protocol, so that the web client mistakenly changes the domain name of a destination website to be the hacker's IP address. This method will lure the client to response to the fake web server or the hacker who is preparing a suspicious application to capture the password and redirect it into the website. The tools popularly used by the hackers are dnsspoof and webmitm on BackTrack [2] CD Live. In addition to DNS Spoof, the hacker could lure a victim for a password by using Phishing Techniques to send a fake link to the victim's email or on either web-board or Facebook. The victim will be lured to click on the link and mistakenly log in. Normally, to use this technique, the hacker would generate a fake website and design it as a website of some financial institution. Another method for the password cheating on web application is to directly attack HTTPS; for example, using a software to fake the certificate e.g. "Cain and Abel" [3] performing as a Man in the Middle between the web server and web client. This software will generate a fake certificate, fake public key, and fake private key, as well as capture and decode all data exchanged between the web server and client, including passwords. Currently, the modern web browsers have been equipped with features to prevent this cheating method by giving a notification after the fake certificate has been detected. Unfortunately, most of the users regularly close a pop-up to ignore the notification and their passwords are finally cracked.

1.2 Password Capturing on LAN

It is easy to capture a password on LAN by manipulating the gap in ARP protocol. Normally, the hacker applies ARP Spoof technique by using a tool, e.g. Switch Sniffer on Windows or arpspoof on Linux, as a Man in the Middle. After that, the hacker will capture the information with the protocol analyzers including Ethereal and Wire Shark. If the data or password is plain text, the hacker will be able to read them immediately. Differently, if the data has been encoded, the hacker needs to decode it first. However, several

communication systems have been encoded with fixed algorithm e.g. DES or AES. In that case, the hacker needs to capture the key first and uses it for decoding. Occasionally, the client and server exchange the password in a form of hashed value (e.g. servers of Microsoft platform), but the hacker still can find the password by cracking the hashed value with John the Ripper. Detchasit Pansa and Thawatchai Chomsiri [4] suggested an architecture and protocols for secure LAN by using a software-level certificate and cancellation of ARP protocol.

1.3 Password Capturing on Network Device

On some network devices, e.g. LinkSys Wireless Access Point, the password is sent (from browser to devices) with a simple encryption by using Base64 algorithm, in which Ethereal or Wire shark is able to decode and show the plain password after it has been captured. Still, only few network devices are installed with this kind of weak security system. In a different way, the devices with strong security system, e.g. Cisco Router, will have its secret encoding with a complicated algorithm i.e. PAP and CHAP. In this regard, it is broadly agreed that PAP has a weak security system, so that Cisco purposively persuades the user to use CHAP and avoid installing PAP on new Cisco IOS. Nevertheless, some websites, e.g. http://www.oxid.it/, still claim that they are able to break the CHAP password using Cain and Abel [3].

2. Related Work

Sending a password that has been hashed before being authenticated has been suggested in Lamport's [5] and Shiumizu's [6] studies. Lamport's method faced with problem after the password had been reset; meanwhile, Shimizu's so-called CINON had nothing to do with password resetting, but its functions did not cover the web authentication. Frankly, these two studies [5] [6] applied PERM protocol [7] and used number randomization mainly on network devices and servers but did not mention its benefits when used on web application. In addition, the mentioned methods [5], [6] and [7] are still complicated and difficult to be practically used.

Lei [8], [9] suggested an authentication with visual password system to be applied with randomized linear generation function. In the same idea, there are a number of related works on visual password system e.g. Xiao's studies [10] that applied this method with "Secret Little Functions" and "Codebooks" to hide the password, which was similar to Haskett's [11] and Szydlowski's [12] "Codebook-Based Solution". Besides, N.M. Haller [13] suggested S/Key one-time password system used with hashing function, in which the server will firstly verify the password of the hashed value. In addition to the above mentioned studies, Preecha Noiumkar and Thawatchai Chomsiri [14] suggested Dynamic Web Session IDs but aimed not to directly fix the problematic password authentication. However, the algorithm for generating Dynamic Web Session IDs has a strong security and rapid processing system, since it utilizes hashing function rather than the encoding key.

To design the password authentication in this research, hashing function was purposively put into a more suitable position, as well as the number randomization was used to constantly change the hashed password to make it more difficult to be cracked.

3. Pre-stage Design

The concept of generating the dynamic password is that the client appended the password with one string, and hash before sent it to the server. Every single string added to the password must not be repeated and should be randomized. In this case, the hashing function was used to prevent the hacker from decoding the password. In details, the dynamic password was designed as illustrated in section 3.1.

3.1 Simple Design

```
Phase-1: [Client]

1: ST = strRandomString   //--- Step1
2: ClientDynamicPassword = Hash( ClientPassword +
   ST )   //--- Step2
3: SendToServer( strUsername, ClientDynamicPassword,
   ST )   //--- Step3

Phase-2: [Server]

4: ServerPassword = GetPassword( strUsername )
   //--- Step4 -- query from DB
5: ServerDynamicPassword = Hash( ServerPassword +
   ST)   //--- Step5
6: if (ClientDynamicPassword =
   ServerDynamicPassword )   then
      AuthenticationPass = TRUE
   else
      AuthenticationPass = FALSE   //--- Step6
```

From the model, it is shown that the password could be changed to be dynamic by an equation in Step 2, since ST would be randomly changed for every authentication process. Technically, the server is able to verify whether the password correctly matches the client, by using the same method as the client's which means adding ST to the password before being hashed as shown in Step 5. Nevertheless, it is not actually secure to use this model since the hacker will be able to capture strUsername, ClientDynamicPassword, and ST that the client sent to the server in Step 3, and this process will be repeated (sent to the server) to bypass the authentication using strUsername's right.

In some related studies, to develop the model requires some factors e.g. IP Address to enhance the authentication system. In contrast, the researchers have noticed the weakness in using IP Address as follows: 1) these factors are unusable when both victim and hacker are behind NAT or Proxy; 2) it

becomes an overhead to verify new factors added (IP Address). As a result, the researchers did not use such IP Address and noted that it is more important to protect the information within the ST from the hacker.

3.2 Developed Design

The researchers have developed the model by assigning the server to generate and send the ST to the client so that it will be added to the password before being hashed. Particularly, sending the ST from server to client requires a secured encoding that can be decoded only by the server and client. Indeed, to send the ST from server to client, the researchers did not generate a new key in order to avoid some problematic sequences, especially the problem in sending the key securely, which is as difficult as securely sending the ST. As a consequence, the researchers have decided to seek new methods and finally found a very interesting one.

The researchers have found that both server and client already have the secret key that only the server and client know. This key is actually derived from the password. Thus, the server is able to send ST to client safely by encoding ST using the password within the server's database as the encoding key. It resulted as Cipher. After that, the server will send Cipher to the client, and then the Cipher will be decoded with the client's password. If both of the passwords correctly matched, it means that the ST has been successfully sent and can be used for the next steps. This suggested model is shown as below.

```
Phase-1 [Client]
1: SendToServer( strUsername )   //--- Step1

Phase-2 [Server]
2: Server_ST = strRandomString  // --- Step 2
3: ServerPassword = GetPassword( strUsername )
   // --- Step 3 -- query from DB
4: Cipher = Encode( ST, ServerPassword )
   ////// ---  ST is data, ServerPassword is key
5: SendToClient( Cipher )

Phase-3 [Client]
6: Client_ST = Decode( Cipher, ClientPassword )
    ////// --- Cipher is data, ServerPassword is key
7: ClientDynamicPassword = Hash( ClientPassword +
Client_ST )
8: SendToServer( ClientDynamicPassword )

Phase-4 [Server]
9:ServerDynamicPassword = Hash( ServerPassword +
  Server_ST )
10: if (ClientDynamicPassword = ServerDynamicPassword)
    then AuthenticationPass = TRUE
    else AuthenticationPass = FALSE
```

The model's operation starts when the client sends strUsername to the server to claim for the authentication and after receiving strUsername, the server will random for ST as shown in Step 2 (the researchers used variable 'Server_ST' to store the value of server's ST). Later, the server will search for the user's password who requests for authentication and uses it as the encoding key on Cipher's line – **Cipher = Encode (Server_ST, ServerPassword)** as shown in Step 4. Cipher will be sent to the client. After receiving Cipher, the client will use the password on the client's side (received from the user's keyboard or those stored in the network devices. It depends on how it will be used) as the key to decode Cipher for ST. If the passwords on both sides correctly match, the client will successfully get the correct ST. For these reasons, this method is very safe for ST sending because only the server and client know this secret decoding key. In particular, after receiving ST, the client will originate the dynamic password with an equation- **ClientDynamicPassword = Hash (ClientPassword + Client_ST)** as illustrated in Step 7. After the equation has been analyzed by researchers, it was found that ClientDynamicPassword will be changed each time that the authentication is activated because Client_ST, received from Server_ST, is the string that has been randomized by the server. Then, in Step 8, the client will send DynamicPassword to the server to verify whether the client's password matches the password in the server's database. Similarly, the server uses **ServerDynamicPassword = Hash (ServerPassword + Server_ST)** as shown in Step 9. If the passwords on both sides correctly match, Server_ST will match Client_ST, and ServerPassword will match ClientPassword. Also, ServerDynamicPassword on the server's side will match ClientDynamicPassword on the client's side. For these reasons, the comparison in Step 10 finally becomes true.

4. Final Design

After designing software for testing, it was noticed that the design mentioned in section 3.2 still has some errors; for example, in case that the hacker enters/sends a username and password with an empty value or space, he will be able to bypass the authentication. That is, when considering Step 3, after GetPassword() has received an input as an empty string, it will return that empty string (a zero-length string or "") to ServerPassword so that the client can decode it for ST (because the password on the client's side is an empty string similar to that on the server's side). Finally the DynamicPassword creation on both client's (Step 7) and server's sides (Step 9) have the same result since they have the same ST and empty string. Consequently, the researchers have developed the design to be more secure and present it as an operational diagram as illustrated in Figure 1.

From Figure 1, S1 is Server_ST mentioned in 3.2, as S2 is Client_ST. U1, P1, and H1 are strUsername, ClientPassword, and ClientDynamicPassword respectively. U2, P2, and H2 are strUsername, ServerPassword, and ServerDynamicPassword respectively. The operational system consists of 4 phases.

Based on the designing errors found in section 3.2 that the hacker is able to bypass the authentication by keying in username and password with empty string, this error can be

protected by adding a condition - **if (U1 not exist) { P2 = RandomByeArray() }** (See Figure 1) to make P2 a filled value that the hacker is unable to know since it was randomized by the server. Therefore, the hacker cannot generate H1 that matches H2 on the server's side.

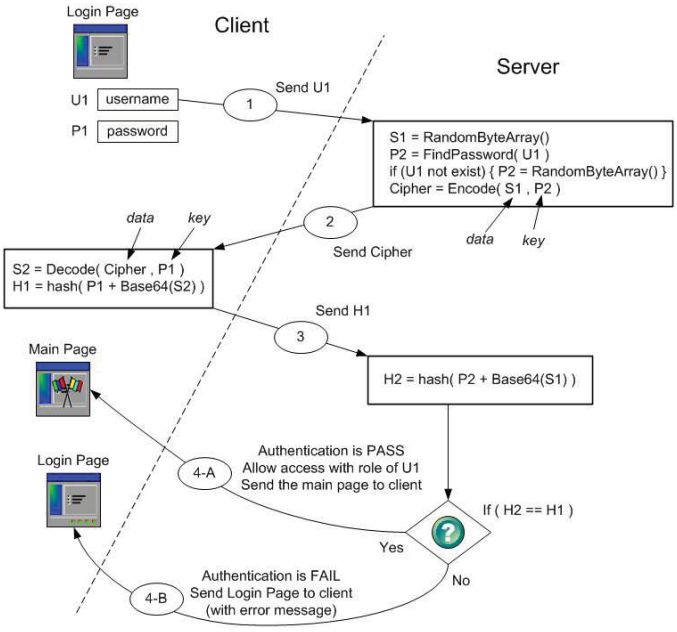

Figure 1: Dynamic Password (Designing Steps and Analysis)

Practically, the researchers are tend to design an operational system with minimum steps and factors (not to use IP Address, MAC Address, Server's time, Client's time, Keys). In the meantime, the researchers need to generate a strong security system that is very difficult to be cracked. Thus, the researchers agreed on that fixing the designing errors in section 3.2 by adding a one-line condition - **if (U1 not exist) { P2 = RandomByeArray() }** is the most effective solution for the mentioned designing errors.

It is notable that it is more applicable to use **S1 = RandomByeArray()** on the server's side to put S1[0]...S1[n-1] on a range between 0 to 255 of ASCII number, rather than using **S1 = RandomCharArray()** that can put S1[0]...S1[n-1] on a range only between 32 to 127 of ASCII number (e.g. 'A'..'Z', 'a'..'z', '0'..'9', space, '!', '@', '#', '$', and other characters). In prior studies, the researchers have tested **S1 = RandomCharArray()** and cracked it with the code on client's side that has been decoded as shown in Figure 1, on a line - **S2 = Decode(Cipher, P1)**. The researchers conducted a brute force by running P1 from 0 to 225 ASCII number, while the length is progressively extended. From the researchers' observation, it was found that when using P1 that mismatches P2, bytes in S2 (S2 [i]) will become binary (ASCII number 0-255). In contrast, if P1 matches P2, the S2 will be the strings (ASCII number from 32 to 127), because a command **S1 = RandomCharArray()** (with a traditional method) on the server's side can make S1 a normal string. Therefore, the problem has been fixed by changing traditional method

S1 = RandomCharArray()

to be

S1 = RandomByeArray()

which is a modern method as illustrated in Figure 1.

In the same case, S1 is required to be hashed with Base64() on a line – **S2 = Hash (P2 + Base64 (S1))** on the server's side.

Note: The notation "+" means appending a string.

Although the system consumes much time to fully complete all steps though user U1 does not exist, the researchers have the reason for that time consuming. That is, whether user U1 is exist or not, the systems should take the same duration for finishing the whole procedure, so that the hacker is unable to detect the user's existence on the system by evaluating the response time. Accordingly, the solution that the researchers have proposed will usefully support both procedural simplicity and security.

5. Performance Analysis

The performance analysis involves in 3 dimensions including Security, Procedural Simplicity, and Operational Speed.

5.1 Security

The feature of this design is the security as explained below.

5.1.1 Hashed Password cannot be reversed.

From Figure 1, as stated in Step 3 that the client sent H1 (Dynamic Password) to the server, the hacker is able to capture H1 but he cannot reversely decode it to get the password (P1), since the password has been already hashed. The password hashing was shown in the H1 creation on the client's side (a one-way encryption of hashing function protects the password from being reversely decoded).

5.1.2 Dynamic Password (H1) cannot be Cracked.

In general, the hacker can crack the password from hashed value, e.g. the user's password in MySQL database, that has been hashed with MD5 or SHA1. In this case, it is possible for the hacker to crack the password because the applications, including MySQL, normally hash the password themselves, so that the hashed values similarly become constant. Unfortunately, these constant values can be easily cracked by brute force technique using John Ripper.

Implementing the researcher's model, the hacker can capture H1 but he cannot crack for P1 since the hashed values become inconstant and keep changing each time when the authentication is activated. In details, to generate the dynamic password, the password will be added with the string that has been randomized (S2 on the client's side) before being hashed. The hashed value of dynamic password will be always changed so that it is impossible for the hacker to use H1 to crack the password.

In case that the hacker has captured H1 and wants to crack P1, he has to know S2 which is perfectly protected as explained below.

5.1.3 S2 (Server's S1) is sent safely.

As previously mentioned, the server will firstly generate S1 by randomizing before encoding it. Cipher of S1 will be sent to the client and it will be decoded to be S2. This process has a strongly secured encryption since "no key is sent across the networks" and both the server and client already have their correctly matched password and use it as the key for encoding and decoding. Besides, S1 has been made in binary that cannot be analyzed and reversed with any statistical approaches.

5.2 Procedural Simplicity

One of the beneficial features of the researchers' design is the procedural simplicity because the whole procedure takes a few steps and requires few factors and resources. This design is easy for the programmer to understand and the system is easy to be practically implemented.

The researchers' model requires a short period for its procedure. From figure 1, it can be seen that the whole procedure only consists of 4 phases. Explicitly, the first procedure on the server's side has only 4 steps (see section 3.2), while the second on server side has 2 steps. Likewise, on the client's side, the procedure has only 2 or 3 steps. Additionally, the model requires a small number of factors and resources. In previous studies, some models depend on many factors to strengthen the security system, in which it is difficult to be practically implemented, takes a longer period for processing, and unnecessarily consumes too much resource. On the contrary, the researchers' designed model has nothing to do with IP Address, MAC Address, Server's time, Client's time, and additional keys (only a password is used as the key). Furthermore, the system proposed by the researchers will never put a heavy overhead on the CPU because it uses only the hashing function that can be processed easily and more applicable for CPU than RSA algorithm. In practical, the encryption with a password as the key, the user who wishes to implement this model can choose a simple algorithm e.g. using DES rather than 3DES or AES since their security are at the same level (when following the procedure suggested in the researchers' design).

The researchers' designed model is easy to understand since it has a short and clear procedure. The encoding is also uncomplicated with hashing function, symmetric encryption, and Base64 function. For that reason, the programmers can simply implement this model and only have to prepare the hashing function and symmetric encryption. This preparation is much easier than preparing the asymmetric encryption which necessarily requires at least 4 keys (i.e. Server's Public key, Server's Private key, Client's Public key, and Client's Private key). In contrast, the researchers' method requires only the symmetric encryption and has no need to manage the keys (e.g. storing or sending the key across the insecure network) because only the password is used as the key.

5.3 Operational Speed

The model takes a short period for processing since it has a simple procedure and uses symmetric encryption that requires only one key. Specially, programmers are free to choose the encoding algorithm with rapid operation and strong security as recently mentioned in section 5.2. In practical, the symmetric encryption functions more rapidly than does the asymmetric encryption, especially when compared with the encoding and decoding of algorithm RSA in HTTPS on web application. This system has no need to generate, store, or read any keys. In addition, the system utilizes the hashing function that works very quickly, when compared to the encryption, because hashing is only the data digestion.

6. Conclusion and Future works

In this research, the researchers have purposively designed the authentication system with dynamic password to prevent the password capturing, sniffing, and decoding. The authentication system has been designed to have a short procedure and utilize a small number of factors (no need to verify an IP address) and resources. The system has been designed to operate rapidly and applicable to work on CPU because it is applied with the symmetric encryption and hashing function. Moreover, this system is easily understandable and can be simply implemented on web applications, mobile applications, and network devices. To make it clear, the final design is illustrated as a processing diagram to present the system's operational procedure and its strong security to the readers. The diagram introduces the readers to the dynamic password generation (adding the password with the randomized string that has been hashed), difficulty in cracking, and safety in sending sensitive data (S1) from server to client with password used as the encoding key. Furthermore, the readers will see the security by not sending the key directly across the networks. As a consequence, with the researchers' considerate designing, a person wishing to implement this model will be guaranteed that this authentication system has a very strong security.

For the future works, the researchers need to be more careful in testing the operational speed e.g. testing and comparing the speed with HTTPS on a web application and testing the speed with the CHAP used in Cisco router. To test the operational speed, the researchers should implement CHAP algorithm with the same language, the same platform,

and the same system using the researchers' algorithm. Then, the operational speeds of each condition will be individually tested and mutually compared. Besides, the researchers notify that the CHAP and the authentications with other algorithms, e.g. NTLM or Kerberos, still have problems with their security. In this regard, the weak points of these algorithms will be pointed out in order to estimate the possibility of being cracked. The cracking duration will be estimated as well. Finally, the security of the mentioned algorithms will be compared with the researchers' algorithm in order to see the differences

7. Reference

[1] Todd Lammle, "CCNA: Cisco Certified Network Associate Study Guide 6 edition", Sybex, 2007, ISBN-13: 978-0470110089.

[2] Vivek Ramachandran, "BackTrack 5 Wireless Penetration Testing Beginner's Guide", Packt Publishing, 2011, ISBN-13: 978-1849515580.

[3] Michelle Perry, "Cain and Abel", Medallion Press , 2005, ISBN-13: 978-1932815030.

[4] Detchasit Pansa and Thawatchai Chomsiri "Architecture and Protocols for Secure LAN by Using a Software-Level Certificate and Cancellation of ARP Protocol", In Proceedings of the 2008 Third International Conference on Convergence and Hybrid Information Technology (ICCIT'2008). Busan. 2008, Vol. 2 , pp. 21- 26.

[5] L. Lamport. Password authentication with in secure communication. Communications of the ACM. 1981, 24 (1): 770–772.

[6] A. Shimizu. A dynamic password authentication method by one-way function. IEICE Trans. Inform. Syst. 1990, 73 (1): 630–636.

[7] A. Shimizu, T. Horioka, and H. Inagaki. A password authentication method for contents communication on the Internet. IEICE Transactions on Communications. 1998: 81 (2): 1666–1763.

[8] M. Lei, Y. Xiao, S. V. Vrbsky, C.-C. Li, and L. Liu, A virtual password scheme to protect passwords. In Proceedings of IEEE International Conference on Communications (ICC'2008). Beijing. 2008, pp. 1536–1540.

[9] M. Lei, Y. Xiao, S. V. Vrbsky, and C.-C. Li. Virtual password using random linear functions for on-line services, ATM machines, and pervasive computing. Computer Communications. 2008, 31 (18): 4367–4375.

[10] Y. Xiao, C.-C. Li, M. Lei, and S. V. Vrbsky. Secret little functions and codebook for protecting users from password theft. In Proceedings of IEEE International Conference on Communications (ICC'2008). Beijing. 2008, pp. 1525–1529.

[11] J. A. Haskett. Pass-algorithms: A user validation scheme based on knowledge of secret algorithms. Communications of the ACM. 1984, 27 (8): 777–781.

[12] M. Szydlowski, C. Kruegel, and E. Kirda. Secure input for web applications. In Proceedings of 23rd Annual Computer Security Applications Conference (ACSAC'2007). Florida. 2007, pp. 375–384.

[13] N.M. Haller. The S/KEY one-time password system. In: Symposium on Network and Distributed Systems Security. San Diego. 1994, pp. 151–157.

[14] Preecha Noiumkar and Thawatchai Chomsiri, "Improving Web Security Using Dynamic Session IDs", in Journal of Convergence Information Technology,Vol.7. No.2 , February 15, 2012, ISSN : 1975-9320 (Print), 2233-9299 (Online), pp.83 - 91.