Enhancement of Web Application Security by Using Dynamic Secure Cookie

[1]Jinho Jeong, [2] Youngwook Cha
[1] Andong National University , [2] Andong National University
"This research was supported by the MIST(Ministry of Science, ICT), Korea, under the National Program for Excellence in SW), supervised by the IITP(Institute of Information & communications Technology Planning & Evaluation) in 2023"(2019-0-01113)

Abstract— Cookies and sessions are used to overcome the statelessness of HTTP and provide statefulness. Cookies and sessions are used for user authentication, such as login, and are used to temporarily store information that users want to store, such as shopping carts, in online shopping malls. Cookie stores the cookie value in the browser, while session stores the session value in the server, so it has the advantage of being more secure than cookie but has the disadvantage of putting a load on the server. General cookies are less secure than sessions, but are better than sessions in terms of service performance because they do not place a load on the server. A secure cookie must satisfy confidentiality, integrity, authentication, and anti-replay attack, but the previously proposed secure cookie method does not satisfy all four of these. Additionally, most secure cookies are vulnerable to security because they become fixed values unless the data to be encrypted is added or updated. In this paper, we propose DSC (Dynamic Secure Cookie) that is more secure than a session and is superior in terms of usability. Our proposed DSC satisfies all the conditions for secure cookie: confidentiality, integrity, authentication, and anti-replay attack. In addition, it is designed to prevent authentication when manipulating the cookie value, and the cookie is safely protected by continuously changing its value each time the page is changed.
Index Terms—*anti-replay, authentication, cookie, web security*

Introduction

HTTP (Hypertext Transfer Protocol) is a stateless protocol that has been and is still widely used in the web and various other applications [1]. However, in actual use on the web, statefulness is required in many aspects, such as user login and adding and saving items to the shopping cart at a shopping mall. Cookies and sessions were proposed as a way to make HTTP stateful [2]. Session data is stored on the server, and the user's session ID is created as a cookie in the browser. When the browser is closed, the created cookie deletes, and the session file remains on the server for a certain period of time. On the other hand, for cookies with an expiration time, the data is stored in the user's browser, and the cookie data remains on the user's computer even after the browser is closed [3].

The more sessions saved, the more the load on the server increases, so it is recommended to use cookies store in the browser. However, the use of sessions that store information on the server is more secure in terms of security. A session with these characteristics satisfies the confidentiality, integrity, and authentication of user data. However, if the session ID is exposed, an attacker can gain user privileges by changing the session through the browser, so there is a vulnerability in replay attacks.

In this paper, we propose DSC (Dynamic Secure Cookie) that is more secure than a session and is superior in terms of usability by improving the security aspects of cookies without using sessions that can put a load on the server. Previously proposed secure cookies [4,5] did not satisfy all of confidentiality, integrity, authentication, and anti-replay attack. DSC proposed in this paper creates a hash value which is used as a cookie, by combining user ID, IP address, User-Agent, the cookie expiration time newly created each time the page is moved, and the server key hidden in the server. Our proposed DSC uses hash values

generated by inputting various items, so it satisfies all of the confidentiality, integrity, authentication, and anti-replay attacks of cookies. Since the continuously changing hash value is used as a cookie, it is secure to cookie stealing attacks and personal information stored in the cookie is also protected.

Related Works

A. Session and Cookie

To provide statefulness to HTTP's stateless communication, the server and client exchange cookie or session information by passing Set-Cookie and Cookie headers. The cookie transmission between the client which is the web browser and the web server is shown in Figure 1. The cookie first created on the server is delivered to the Set-Cookie header of the HTTP response message and stored in the browser. At this time, the cookie classifies the data that needs to be stored on the client by cookie-name and stores the data in the cookie value. The session uses the session name according to the web server instead of cookie-name, and the session-id generated by the server is stored as the cookie value.

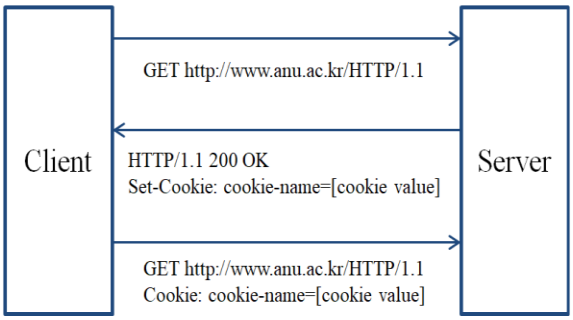

Fig. 1. Cookie transmission between server and client.

The value for maintaining statefulness can be different depending on the user's browser operation. In the cookie method, updated cookie values are frequently transmitted to the server to confirm data. On the other hand, in the session method, the session-id stored in the browser as a cookie value is not updated, but the data is updated and stored in the session file of the web server corresponding to the session-id. When generating a cookie through Set-Cookie, it is generally generated by entering the expiration time of the cookie. This means that the cookie cannot be used after its expiration time. A session does not have an expiration time. When the user closes the browser, the session value stored in the browser is deleted.

B. Security threats of sessions and cookies

The expiration time of both sessions and cookies can be manipulated. If the expiration time of the session is manipulated, the session will be maintained even if the browser is closed. Even if the cookie expiration time is set as 1 hour later when creating a cookie, the cookie expiration time can be manipulated to a desired time. If the expiration time of a cookie is manipulated, the cookie remains alive until the manipulated expiration time. The server cannot know whether the session or cookie's expiration time has been manipulated.

Cookies and sessions are stored in the cookie storage file of the browser used. Normal sessions are deleted from the cookie storage file when the browser is closed, but cookies remain in the storage file even after the expiration time passes after the browser is closed. Since the session file of the web server is maintained during the session maintenance time of the web server, the user can restore the session with the session-id within the session maintenance time. This means that if someone knows the session-id, another user can also restore the session. If the session is restored within that period, the session retention time is reset.

Sessions and cookies whose expiration time has been manipulated remain in the cookie storage file even after the expiration time has passed after the browser is closed. However, in the case of a session, if the session maintenance time of the web server has expired, it will not be restored even if the session is restored with the session-id, and the session-id itself does not include user data. On the other hand, cookies can be replay through stored cookie values, and they can be maintained continuously by manipulating the expiration time. Additionally, since user information is contained in the cookie value, if the cookie value is not encrypted, user information can be known through the cookie value. In this respect, sessions can be considered more secure than cookies.

Sessions are stored as session files on the web server. The maximum size of the session file varies depending on the web server settings, but the default maximum size of the PHP-based session file is 128MB [6]. The more data is stored in the session

file, the larger the file size becomes, and the file loading speed decreases, which puts a load on the server. Cookies store values in the user's browser, and the maximum size varies slightly depending on the browser, but is 4096 bytes for the Google Chrome browser [7]. Cookies do not place a load on the web server compared to sessions because their maximum size is not larger than that of sessions, and data is stored in the browser.

C. Secure Cookie Scheme

An insecure cookie scheme can have a fatal impact on the security of the server by exposing personal information or manipulating the cookie value by exposing the cookie value transmitted between the server and the client. A secure cookie scheme proposed to use cookies securely without using sessions that load the server means that it satisfies all of the confidentiality, integrity, authentication, and anti-replay attacks [8].

1) Cookie Confidentiality: Cookies are stored in the browser so the user can see the cookie name, value, and expiration time. High-level cookie confidentiality means preventing other clients and attackers except the server from reading cookie information stored in the client [9]. Cookies may contain the user's sensitive information, and if the information is exposed, it may lead to personal information leakage. Therefore, secure cookies must be able to satisfy confidentiality.

2) Cookie Integrity: Integrity means protecting specific data from being randomly created, deleted, or manipulated. Assuming that the cookie value contains the user's login information, it is possible to modify the cookie to log in with another user's information. Additionally, if a product that does not actually exist is manipulated and inserted into a shopping cart cookie in a shopping mall, it can have a fatal impact on server-side security [10].

3) Cookie Authentication: Authentication means that the user's access rights have been confirmed by the server. When a user attempts to log in with the ID and password, the server verifies the user's ID and password. If the user is determined to be a legitimate user, an authentication cookie is created and sent to the user, and the user saves the received cookie. When communicating with the server, the browser transmits the stored cookie and the server verifies whether the cookie is valid [11].

4) Cookie Anti-Replay Attack: XSS (Cross-Site Scripting) is an attack that was the 7th threat in the OWASP TOP 10 2017 and became part of Injection, the 3rd threat in the OWASP TOP 10 2021 [12]. XSS is an attack in which a malicious script is inserted into a website, and the malicious script is executed in the browser of the user who views the web page with this script inserted. An attacker who steals a user's cookie or session with a malicious script is used in an attack to log in by replaying the user's cookie or session. In addition to XSS attacks, cookies can be stolen from an infected PC through a virus, or through cookie files stored in the browser of a public PC or another PC. When an attacker replays stolen cookies and attempts authentication, replay attacks can be prevented if the web server can detect that they are invalid [4].

D. Problems of Existing Secure Cookies

In a secure cookie scheme, confidentiality, integrity, authentication, and anti-replay attacks must all be satisfied. However, there is no research yet on a cookie scheme that satisfies all of these requirements. Previously proposed secure cookies [4,5] do not sufficiently provide one or two items such as confidentiality or anti-replay attacks.

1) Alex's Secure Cookie: According to July 2023 statistics from W3Techs, a web program statistics website, Wordpress is the most used CMS (Content Management System) in the world [13]. Wordpress accounts for 63.1% of the total CMS market and 43.2% of all websites. Wordpress login is currently using the cookie scheme proposed by Alex [4,14]. Alex's secure cookie scheme is shown in Figure 2.

$$\text{user name}|\text{expiration time}|(\text{data})_k$$
$$|\text{HMAC}(\text{ user name}|\text{expiration time}|\text{data}|\text{session ID}, k)$$
$$\text{where } k=\text{HMAC}(\text{user name}|\text{expiration time}, sk)$$

Fig. 2. Alex's cookie scheme.

A cookie is created by concatenating the user name, cookie expiration time, data encrypted with the key value k, the result of hashing the user name, cookie expiration time, data, and SSL (Secure Sockets Layer) Session ID with k. Here, data refers to data that the user will use on the website. For example, if the user put an item in the shopping cart, this data is maintained even if the user moves to another page. The problem in this case is that the user's name is not encrypted, so it is possible to

know which user the client was authenticated as through the stored cookie file. This can be seen that the confidentiality of cookies is not satisfied.

It is impossible to get the cookie expiration time from not only Javascript but also backend languages such as PHP, JSP, and ASP. Accordingly, Alex's cookie had an expiration time inserted into the cookie to check the expiration time of the cookie when the user logged in. When a user logs in, the expiration time is created by adding the cookie retention time to the login time, so the expiration time must be long enough to guarantee user availability. Otherwise, when the user stays on the website for a long time, the user will be logged out frequently, causing inconvenience to the user. If you allow the creation of session cookies by setting the expiration time of cookies to zero to solve this inconvenience, it is vulnerable to security because you can decrypt cookies at any time without user authentication. Therefore, Alex's cookies have no choice but to keep them for a long time unless the user logs out and forcibly deletes the cookies.

Alex's cookie used an SSL Session ID to prevent replay attacks. However, most backend languages cannot get the SSL Session ID value. Therefore, anti-replay attacks cannot be achieved in backend languages such as PHP or ASP that do not support SSL Session ID values. According to July 2023 statistics from W3Techs, a website that provides web program statistics, about 16.7% of websites around the world are still using HTTP, not HTTPS [15]. Since SSL is not used on websites that only use HTTP, Alex's cookie's anti-replay attack function does not work. Before starting a web application service, it is common to work with the IP address of a private or public server. Alex's cookies are highly unsuitable in terms of compatibility because they cannot be used on test servers that do not use SSL, or because they must be modified to use the cookie system. Accordingly, Wordpress also excludes the SSL Session ID when using Alex's cookies [4,14]. When used except for SSL Session ID, Alex's cookie becomes a fixed value unless the data is updated. Therefore, if another user can have the cookie value, the cookie can be played with the cookie value and the expiration time of the cookie can be manipulated to use the service in the name of another user.

2) Lee's Secure Cookie: Lee's cookie scheme [5], which complements the confidentiality of the Alex's cookie scheme, is shown in Figure 3.

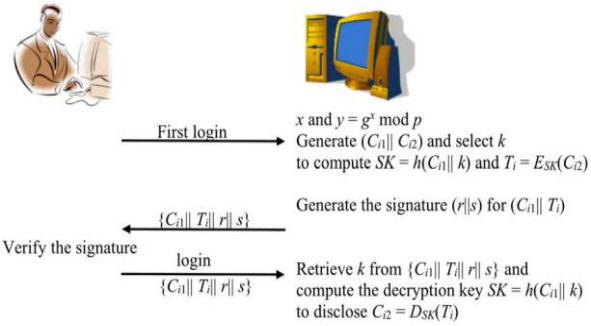

Fig. 3. Lee's cookie scheme.

This method begins by first distinguishing between general data and sensitive data. A key is created through the selection and calculation of a large prime number, attached to general data, and then a hash value is generated using a hash function to create a server key. Sensitive data is encrypted with a symmetric key using the generated server key, and Lee's cookie is created by combining general data and encrypted data. The created cookie is stored in the user's browser, and the server uses the cookie stored in the user's browser to calculate and find the key generated through the previously selected prime number. When the data encrypted through the found key is decrypted, it is valid if the decrypted data is recognizable, otherwise authentication is rejected. Lee says that this method is fast because it uses symmetric key encryption, is secure because it does not require a sensitive verification table on the server, and the key is simple.

The problem with this cookie scheme is that the cookie created when authenticating each user always has the same value if the user does not re-authenticate or the cookie data is not updated. Therefore, if the cookie is stolen from an attacker, there is still a vulnerability to replay attacks that can be authenticated from outside. Even if an attacker replays the cookie and uses it, there is no problem in verifying its validity because the cookie value itself has not been manipulated. Additionally, when a user or an attacker manipulates the expiration time of this cookie, there is no way to recognize the manipulated expiration time, so the cookie can be maintained indefinitely, making it vulnerable to security. If Lee's cookie has a cookie expiration time like Alex's cookie, the cookie expiration time will

have to be as long as Alex's cookie, so it will be vulnerable to replay attacks as well.

II. Enhancement of Web Application Security by Using Dynamic Secure Cookie

A. Creating Dynamic Secure Cookie

The Dynamic Secure Cookie (DSC) proposed in this paper is divided into two methods, as shown in Figure 4, depending on existing or not existing of data that requires encryption.

DSC1 = Hash(Username|IP Address|User-Agent|Cookie's new Expiration|Secret Key)
(1) Without data to be encrypted

DSC2 = Hash(Username|IP Address|User-Agent|Cookie's new Expiration|Secret Key)|(Data|Secret Key)k
where k = Hash(Username|IP Address|User-Agent|Cookie's new Expiration|Secret Key)|Secret Key
(2) With data to be encrypted

* | is separator * || is merging string

Fig. 4. Dynamic secure cookie depending on the pr esence or absence of encrypted data.

First, the member's ID or unique information, the currently accessed IP address and User Agent, the expiration time of the cookie that adds a short time to the current time, and the server secret key are connected to the separator | to generate a hash value DSC1. If there is no data that requires encryption, user authentication can be performed using a cookie using this hash value. The encryption key k, used when there is data that needs to be encrypted by symmetric encryption function, is created by adding the server secret key to the end of DSC1. After adding a server secret key with a separator in front of or after the data, the connected item is encrypted with a symmetric key with k.

The result created by connecting the hash value DSC1 and the encrypted value with the separator | becomes the cookie value DSC2 when there is data that requires encryption. The web server stores the generated hash value DSC1 and the cookie expiration time as a specific column in the database member table. When a user moves to another page or refreshes the page, a short expiration time is added to the current time to dynamically regenerate a secure cookie value as shown in Figure 4. Afterwards, the newly created hash value DSC1 and the cookie expiration time are updated in the database.

B. Verification of Dynamic Secure Cookie

In web applications, the verification process shown in Figure 5 is performed to check the validity of the created DSC.

Cookie Verification

*** TRUE : Cookie is valid**

*** FALSE : Cookie is invalid, so delete the cookie in web browser, hash value and cookie expiration time in database. After that, go to index page.**

1. Compare the cookie's hash value in the user's browser with hash value in database. If the values are the same, return TRUE otherwise FALSE.
2. Compare the cookie's expiration time in database with current server time. If the cookie is not expired yet, return TRUE otherwise FALSE.
3. Compare the following three values below:
   a. hash value in database,
   b. browser cookie value,
   c. Hash(Username|IP Address|User-Agent|Cookie Expiration|Secret Key)
   (Username and Cookie Expiration are in database. IP Address and User-Agent are from the user's browser.).
   If the above three values are the same, return TRUE otherwise FALSE.
4. If data exists, decrypt with k in the right side of |.
   If the decryption data includes "|Secret Key", return TRUE otherwise FALSE.

Fig. 5. Verification process of Dynamic Secure Cook ie.

When TRUE is returned through the DSC verification process, user authentication is maintained. On the other hand, when FALSE is returned, the user's login is canceled and the browser's cookie as well as the cookie hash value and expiration time stored in the database are deleted. If the cookie hash value and expiration time are removed from the database, authentication by an attacker is impossible even if the user's cookie value is stolen. Number 3 of Figure 5 compares the hash value of the database, the cookie value stored in the browser, and the hash value created with the user name currently authenticated and used by the user, the currently connected IP address and User Agent, the cookie expiration time stored in the database, and the server secret key. By comparing all three values, including the created hash value, not only manipulation of cookie values in the user's browser but also manipulation of the database can be detected and blocked, thereby protecting user information more safely.

C. Confidentiality and Integrity of Dynamic Secure Cookie

The server secret key used to create DSC is a value hard-coded within the server and used as a

secret key. In Figure 4, the member's unique user name, IP address, User Agent, and cookie expiration time used to create the DSC1 are information that an attacker may know, and the attacker can attempt to crack by combining these information to generate a hash value. However, by including the server secret key here, cracking by an attacker is almost impossible. If there is data that requires encryption, an attacker may guess that the encryption key value of the encrypted data value is the hash value in front of the separator |. The secret key was once again included in the hash value, making the data encryption key value unpredictable. The secure hash function has one-way features and cannot be decrypted [16]. Accordingly, the confidentiality of the cookie is guaranteed because the key value of the result of the data encrypted with the symmetric key generated by this hash value and the server secret key is unknown.

To check the integrity of a value, a hash value generated by a hash function is generally used. This is related to collision resistance, which is one of the characteristics of the hash function, which is that if the input values are different, the hash values are always different [17]. DSC1 is a hash value generated by combining the user name which is unique information, IP address, user agent, cookie expiration time, and server secret key. When there is data to be encrypted, DSC2 is the result of symmetric key encryption by combining the data and the server secret key and connect with DSC1 using the separator |. An attacker who does not know what values the cookie consists of will not be able to generate and authenticate the same cookie with a random value depending on the collision resistance of the hash function.

Cookies can be manipulated, so when an attacker manipulates cookies in the data part, there is a risk of causing load or tampering with data depending on the system. It can be checked whether the cookie has been manipulated with by checking whether a separator such as | exists in the completed cookie value. Additionally, when decrypting data, if the decrypted value can be verified by checking only the server secret key attached to the front or back of the data, you can be safe from the risk of cookie manipulation. For example, assuming that the server secret key is 'ANU13579', if an attacker manipulates the encrypted data value, the system decrypts the manipulated password value with the key and determines whether 'ANU13579' including | is left or right side of the data. If 'ANU13579' exists, it is processed as the correct data value. If it does not exist, the cookie can be deleted to be secure from attacker attacks.

D. Login Authentication based on Dynamic Secure Cookie

The overall login authentication process based on DSC is shown in Figure 6.

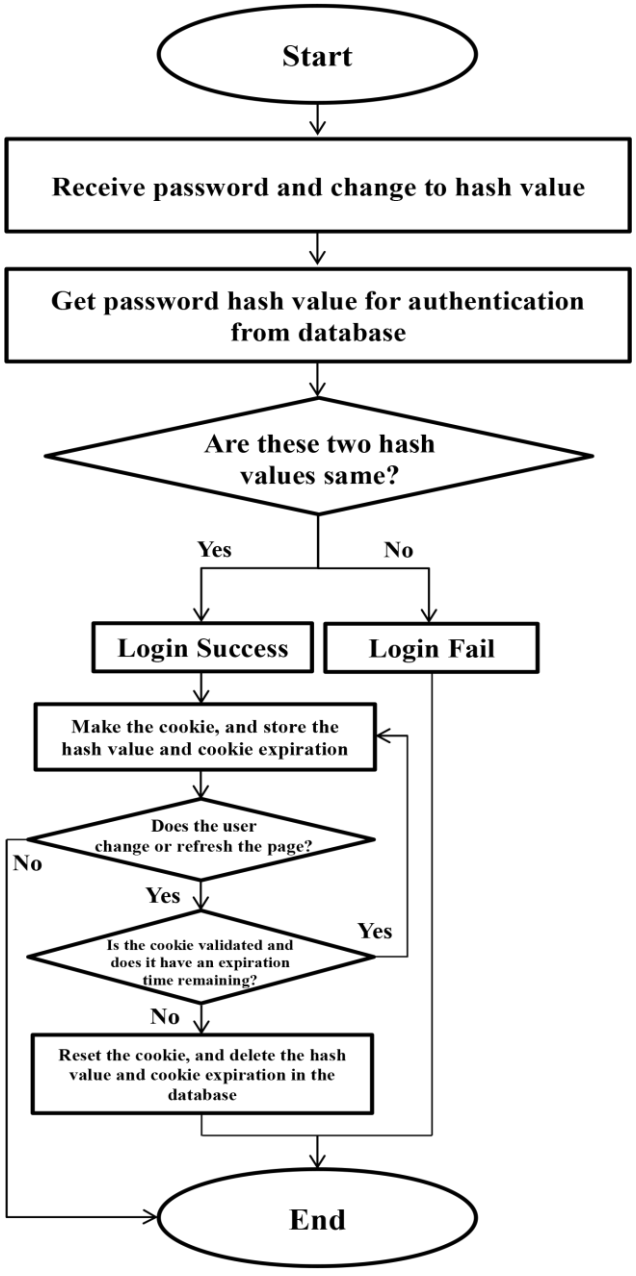

Fig. 6. Login Authentication based on Dynamic Secure Cookie

When a user enters an ID and a password, the server determines whether the password of the corresponding user information matches. If the information matches, a cookie is created and the generated hash value and cookie expiration time are stored in the database. The created cookie consists of the user's ID, the access IP address and User Agent, and the cookie expiration time, so it is definitely a unique value. Accordingly, users can get user information by searching the hash column of the member table in the database using cookies stored in the browser. When the user moves to a page or refreshes the page, the server side checks the cookie verification and expiration time. If the cookie has not been manipulated and the expiration time remains, the hash value and cookie expiration time stored in the user's browser and database are updated. If cookie verification fails or the user presses the logout button, the cookie stored in the browser and the hash value and cookie expiration time stored in the database are deleted. If there is no hash value and cookie expiration time in the database, the user's cookie cannot be verified, so logging in through manipulation is impossible in the DSC scheme. By forcibly deleting the user's cookie and expiration time, the website administrator can force logged-in users to log out, thereby strengthening member authentication. In addition, it can help with member management by searching for users whose cookies stored in the database have remaining expiration time and identifying users who are currently connected.

In accordance with the privacy policy of the U.S. NIST and many other countries, website personal information processors must store visitors' access records in the database [18]. Dynamic Secure Cookie uses access records such as IP address and User Agent that are stored when the user logs in. Accordingly, it is possible to detect and block intrusions from external users other than the user's IP address or User Agent.

E. Anti-Replay Attack of Dynamic Secure Cookie

Among the components of DSC, the hash value and expiration time of the cookie are stored as values when the user logs in. Even if an attacker knows the secret key and creation method of the DSC, if the corresponding value is not in the database, authentication is impossible. The cookie expiration time is updated by adding a short time to the current time every time the user moves the page or the page is refreshed. Accordingly, the expiration time changes when the user moves or refreshes the page, so the cookie changes continuously. Cookie replay attacks require the attacker to get the cookie within the expiration time and operate it in the attacker's browser. If an attacker does not get the user's cookie within the short cookie expiration time, a cookie replay attack is not possible. Even if an attacker get the cookie, if the user's IP address and User Agent contained in the cookie are not the same, user authentication through cookie replay is impossible. Additionally, if the user moves the page or refreshes the page, the cookie changes, so a replay attack cannot be made. Even if the cookie does not change because the user turns off the browser without pressing the logout button, and the IP address and User Agent are the same due to the use of a public PC or manipulation by an attacker, a replay attack is not easy because the cookie expiration time is short. Even though an attacker manipulates to forcibly increases the expiration time, authentication can be blocked because the expiration time is stored in the database.

The short time to add to the current time for generating the expiration time of the cookie is determined by depending on the content of the website. According to a study by Chao [19], the average time spent on a web page is less than 1 minute. However, on websites with video streaming content such as YouTube, one minute can be a very short time. The EU's ePrivacy Directive states that the cookie duration should not exceed 12 months [20].

By predicting how long a user stays on a page on the website, the short time can be set to generate the expiration time. If the logged-in user automatically refreshes the cookie before its expiration time, it will be possible to prevent the user from logging out after the expiration time while on the page and doing other work. According to PHP.net, the default value of session expiration time (session.gc_maxlifetime) is 1440, which is 24 minutes [21]. If the cookie expiration time is shorter than the session expiration time, it can be a better security effect than a session in terms of maintaining the authentication value after the browser is forcibly closed.

F. Comparison of Security between DSC and Existing Cookies

Table 1 shows whether the elements of the secure cookie scheme are supported by the existing cookie methods and DSC proposed in this paper. Our proposed DSC supports all secure cookie scheme elements: confidentiality, integrity, authentication, and anti-replay attacks. On the other hand, Alex's cookie does not support confidentiality and anti-replay attacks [4], and Lee's cookie does not support anti-replay attacks [5].

Table I
Comparison of Existing Cookies and Dynamic Secure Cookie

Elements of Secure Cookie Scheme	Cookie Methods		
	Alex	Lee	Dynamic Secure Cookie
Confidentiality	X	O	O
Integrity	O	O	O
Authentication	O	O	O
Anti-replay attack	X	X	O

In the existing cookie methods, if the cookie creation method and secret key are exposed to an attacker, the attacker can freely create a cookie value for authentication using the user name and cookie expiration time for which the attacker wants to authenticate. On the other hand, for Dynamic Secure Cookie, cookie authentication is not possible if the cookie value and expiration time are empty in the database. Even if the user forcibly closes the browser without pressing logout and the cookie value remains in the database, authentication is not easy due to the short cookie expiration time.

Implementation And Testing

G. Implementation of Login Authentication based on Dynamic Secure Cookie

To test the compatibility of DSC and compare performance with existing cookies, login authentication based on DSC was implemented as shown in Figure 7. The implemented code was tested for execution in PHP version 7.3 and MySQL version 8.

```
$USERNAME = "anumember";
$IP = "211.122.12.5";
$USER_AGENT = "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
               AppleWebKit/537.36 (KHTML, like Gecko)
               Chrome/77.0.3865.90 Safari/537.36";
$EXPIRATION = time()+600;
$SECRET_KEY = "anu13%&(";
$DATA = "anu13579";

function Make_Secure_Cookie($USERNAME, $IP, $USER_AGENT,
                            $EXPIRATION, $SECRET_KEY, $DATA='')
{
    global $connect;

    $cookie_hash = hash("sha256", $USERNAME."|".
                                  $IP."|".
                                  $USER_AGENT."|".
                                  $EXPIRATION."|".
                                  $SECRET_KEY);

    if($DATA == '')
        $encrypted_data = '';
    else
    {
        $encryption_key = $cookie_hash.$SECRET_KEY;
        $encrypted_data = "|".AES256_Encryption_Function(
                          $DATA."|".$SECRET_KEY, $encryption_key);
    }

    $cookie_value = $cookie_hash.$encrypted_data;

    SetCookie("Authentication", $cookie_value, $EXPIRATION, '/',
              null, false, true);

    $sql = "UPDATE member SET member_hash = '$cookie_hash',
            member_expiration = '$EXPIRATION'
            WHERE member_account = '$USERNAME' ";
    mysqli_query($connect, $sql);
}
```

Fig. 7. Login Authentication based on DSC implemented with PHP

In the PHP code, the Make_Secure_Cookie function creates the cookie proposed in this paper and stores it in the member table of the database. The ID (USERNAME), IP address (IP), User Agent (USER_AGENT), cookie expiration time (EXPIRATION), and server secret key (SECRET_KEY) of the member who successfully verified login are entered as required arguments for the Make_Secure_Cookie function, and user data (DATA) is optionally entered only when there is a value. The entered member's ID, IP address, User Agent, cookie expiration time, and server secret key are separated by | and combined into one string, then a hash value (cookie_hash) is generated using the SHA-256 hash function. If the user's data is not used, an empty value is inserted into the encrypted data (encrypted_data) and combined with the previously created hash value to generate the final cookie value. On the other hand, when using user data, the server secret key is added to the output hash value to generate a key (encryption_key) to encrypt user data. After combining the user data and the server secret key with |, the encrypted value of AES-256 is generated with the previously created symmetric key. The final cookie value

(cookie_value) is created by combining the encrypted data (encrypted_data) with the hash value (cookie_hash) and |.

SetCookie, a built-in function of PHP, is a function that the server sends and stores a cookie to the client [22]. The first argument is 'Authentication' as the name of the cookie. The second one is the created cookie value, and the third one indicates the cookie expiration time. The fourth argument is the cookie path, and the fifth one indicates the domain to which the cookie will be applied. The sixth one is the secure attribute, which means that if true, cookies are used only in HTTPS. The secure property is set to false so that dynamic secure cookies can be applied even to websites that operate via HTTP without an SSL certificate. The last argument indicates whether httponly is used among the cookie properties. The httponly attribute is a means of preventing cookies from being retrieved through JavaScript, and is a proven method for XSS defense. [23]. Even if httponly is used, there are no restrictions on the scheme use, so the httponly property is set to true. After the cookie is stored, the hash value and cookie expiration time used to create the cookie value are stored in the database member table. Cookies are destroyed when the expiration time of cookies stored in the browser expires, but the expiration time of cookies stored in the browser can be manipulated. To prevent manipulation of the expiration time, the expiration time is stored once again in the database.

The Make_Secure_Cookie function is executed every time the user moves the page, so not only does the cookie value change continuously, but also the previous cookie value can no longer be used for user verification. Therefore, if the attacker fails to get and replay the user's cookie within 600 seconds specified as the cookie expiration time, the replay attack will fail. Even if an attacker get a user's cookie, replay attack can be successful only if the attacker's IP address and User Agent are the same as the user's IP address and User Agent, so it can be very difficult to replay user's cookie. In the cookie proposed by Alex [4], replay attacks could be prevented only in HTTPS environments and web applications using specific back-end languages such as JSP, but DSC proposed in this paper can be used in both HTTP and HTTPS environments and all back-end languages that support cookies.

H. Performance Comparison

Table 2 shows the result of comparing the cookie generation time, login authentication time, and cookie verification time when moving to a page of DSC method presented in this paper with the cookie method of Alex[4] and Lee[5]. The time required for each test was recorded as the average of 100 measurements per test. The hash algorithm used SHA-256, and the symmetric key algorithm used AES-256. The code for the test was implemented in PHP 7.3 and tested in MySQL version 8, 3.40GHz CPU, and DDR3 16GB memory environment.

Table II
Performance Comparison Of Existing Cookies And Dynamic Secure Cookie

Category	Alex	Lee	DSC
Generating Cookie Value	0.025 ms	0.026 ms	0.024 ms
Login Authentication	12.786 ms	12.668 ms	12.596 ms
Cookie Verification on Moving Page	12.724 ms	12.659 ms	21.435 ms

The cookie value generating time in Table 2 is measured to check the difference in generation time for each cookie method in the back-end code. The time measured is the time it takes to generate a random cookie value to be sent to the client without using a database. Login authentication time is the time to use the database, receive the user's ID and password, authenticate the login, and process the generated cookie. Login authentication time was measured to compare the difference in processing time between the DSC, which uniquely stores the cookie's hash value and expiration time in the database, and other cookies. Cookie verification time when moving a page is the time it takes to check the validity of the cookie and get the logged in user's information. Cookie verification time was measured to compare the processing time of DSC, which update the cookie value when moving or refreshing the page, with the processing time of other cookies.

For the generation of cookie values through random data, Alex's cookie measured about 0.025 ms, Lee's cookie took about 0.026 ms, and DSC took about 0.024 ms. The measured time was rounded off from the fourth decimal place. All three cookie

methods have very small differences, so it can be seen that there is almost no time difference. In login authentication, Alex's cookies, Lee's cookies, and DSC were measured at times of 12.786ms, 12.668ms, and 12.596ms, respectively.

The reason why the three cookie methods have increased time in login authentication, unlike generating cookie value, is because of the use of a database. In the test of cookie value generation, cookie values were generated from random data without using a database. In login authentication, the ID and password entered by the actual user for login are checked in the database, and a cookie value is generated with the verified user information. And in accordance with the privacy policy, access records such as the user's IP address and User Agent are stored in the database [18]. Alex and Lee's cookies only save the IP address and User Agent when saving the user's access records in the database. However, in DSC, the hash value and cookie expiration time are also stored to detect changes in the cookie value and expiration time used to generate the cookie. Alex's cookie uses HMAC-SHA256 to generate cookie, while DSC uses SHA256. Lee's cookie involves a complex calculation process, such as selecting a large prime number and finding the number of coprimes of -1 to that prime number, but DSC is simple and do not require a complex calculation. In addition, even if not only the access records but also the hash value and cookie expiration time are stored in the database with one query, there is almost no time difference compared to other cookie methods.

Lastly, in cookie verification when moving to a page, Alex and Lee's cookie took 12.724ms and 12.659ms, respectively, and the DSC took about 21.435ms. The reason DSC takes more time than Alex or Lee's cookies, is that the added time to create a cookie and store the updated cookie and expiration time again to the database every time the page is moved is about 9ms. Storing updated cookies in the database prevents replay attacks by continuously changing the cookies, and the updated expiration time by adding a short time not only prevents replay attacks, but also makes cookies more secure than sessions.

When comparing the performance of DSC and existing cookies, the performance was almost identical in generating cookie value and login authentication. In cookie verification when a user moves to a page, the time for generating a new cookie increased by about 9ms compared to existing cookies. According to MIT News, MIT neuroscientists discovered that the human brain can identify images that are visible for 13 ms, but cannot identify images that are visible for less than 13 ms [24]. Therefore, 9ms can be said to be a short time that is difficult for humans to detect. Although the time has increased by 9ms compared to the existing cookie methods, DSC provides all the conditions of secure cookie scheme: confidentiality, integrity, authentication, and anti-replay attacks.

CONCLUSION

In this paper, we proposed Dynamic Secure Cookie that supports confidentiality, integrity, authentication, and anti-replay attacks, which are the conditions for secure cookie scheme that were not satisfied in previous studies [4,5]. DSC uses a hash value generated by concatenating the user name, logged in IP address, User Agent, cookie expiration time, and server secret key. Integrity is guaranteed because it is possible to check whether the information elements that make up the hash value have been tampered with. User data which is combined with server secret key, is unpredictable and is encrypted with a symmetric key using a long hash value as the key, so confidentiality is guaranteed in DSC.

Each time the user moves to a page, the hash value and cookie expiration time are continuously updated. Therefore, even if the cookie value is exposed, if the cookie value and cookie expiration time are changed by the user, user authentication cannot be performed using the previous cookie value. In addition, even if the cookie generating method and secret key are exposed, authentication is not possible unless the cookie value and cookie expiration time are the same as the value stored in the database, so it is secure. If the expiration time of the cookie is set short, even if the user fails to update the cookie value, cookies that have passed the expiration time cannot be authenticated. In addition, even if an attacker manipulates the expiration time, authentication cannot be performed because the cookie expiration time is stored in the database. Although an attacker attempts to authenticate by stealing the cookie value of the browser or the cookie value of the

database within a time shorter than the cookie expiration time, if the user's IP address and User Agent are not the same, authentication will not be possible, so it is secure from replay attacks.

DSC showed similar processing times in cookie value generation time and login authentication compared to existing studies [4,5], and in the cookie verification test when moving pages, the performance only increased by about 9ms, a time that is difficult for humans to detect [24]. However, compared to existing researches, DSC supports all conditions of secure cookie scheme, including confidentiality, integrity, authentication, and anti-replay attacks.

The user's password is stored in one-way encryption so that not even the server administrator can know it [25]. Like password management, user authentication and cookie information must be managed so that they are not exposed not only to attackers but also to server administrators. In order to protect user information more securely, we will later study a method to ensure that no one other than the user can authenticate the user or know the cookie value.

REFERENCES

[1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, “RFC2616: Hypertext Transfer Protocol -- HTTP/1.1,” RFC Editor, 1999.

[2] D. Kristol and L. Montulli, “RFC2109: HTTP State Management Mechanism,” RFC Editor, 1997.

[3] CookieYes, “What are session cookies? Do they need consent?,” CookieYes, 2023 [Online]. Available: https://www.cookieyes.com/blog/session-cookies/. [Accessed: 20-Aug-2023]

[4] Alex X. Liu, Jason M. Kovacs, and Mohamed G. Gouda, “A secure cookie scheme,” Computer Networks, vol. 56, no. 6, pp. 1723-1730, 2012.

[5] Lee, W-B, Chen, H-B, Chang, S-S, Chen, and T-H, “Secure and efficient protection for HTTP cookies with self-verification,” International Journal of Communication Systems, vol. 32, 2019.

[6] PHP.net, “Description of core php.ini directives,” PHP.net. 2023 [Online]. Available: https://www.php.net/manual/en/ini.core.php. [Accessed: 20-Aug-2023]

[7] Tutorials Point, “What is the maximum size of a web browser's cookies value?,” Tutorials Point, 2023 [Online]. Available: https://www.tutorialspoint.com/What-is-the-maximum-size-of-a-web-browser-s-cookies-value. [Accessed: 20-Aug-2023]

[8] Kevin Fu, Emil Sit, Kendra Smith, and Nick Feamster, “Dos and don'ts of client authentication on the web,” In Proceedings of the 10th conference on USENIX Security Symposium, Vol. 10 (SSYM'01), 2001.

[9] J.S. Park and R.S. Sandhu, “secure cookies on the web,” IEEE Internet Computing, vol.4, pp. 36–44, 2000.

[10] Barth, A., “HTTP State Management Mechanism,” RFC 6265, DOI 10.17487/RFC6265, April 2011.

[11] Barbato, S., Dorigotti, S., and T. Fossati, Ed., “SCS: KoanLogic's Secure Cookie Sessions for HTTP,” RFC 6896, DOI 10.17487/RFC6896, March 2013.

[12] OWASP, “OWASP Top Ten,” OWASP, 2021 [Online]. Available: https://owasp.org/www-project-top-ten/. [Accessed: 20-Aug-2023]

[13] W3Techs, “Usage statistics and market share of WordPress,” W3Techs, 2023 [Online]. Available: https://w3techs.com/technologies/details/cm-wordpress. [Accessed: 20-Aug-2023]

[14] Wordpress, “wp_generate_auth_cookie(),” Wordpress, 2023 [Online]. Available: https://developer.wordpress.org/reference/functions/wp_generate_auth_cookie/. [Accessed: 20-Aug-2023]

[15] W3Techs, “Usage statistics of Default protocol https for websites,” W3Techs, 2023 [Online]. Available: https://w3techs.com/technologies/details/ce-httpsdefault. [Accessed: 20-Aug-2023]

[16] R. C. Merkle, “Secrecy, Authentication and Public Key Systems,” Ph.D. thesis, Department of Electrical Engineering, Stanford University, Stanford, 1979.

[17] Preneel B, “Collision resistance,” In: van Tilborg H.C.A. (eds) Encyclopedia of Cryptography and Security, Springer, 2005.

[18] NIST, “Site Privacy,” NIST, 2023 [Online]. Available: https://www.nist.gov/privacy-policy, [Accessed: 20-Aug-2023]

[19] Chao Liu, Ryen W. White, and Susan Dumais, “Understanding web browsing behaviors through Weibull analysis of dwell time,” In Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval (SIGIR '10), pp. 379–386, 2010.

[20] GDPR, “Cookies, the GDPR, and the ePrivacy Directive,” GDPR, 2023 [Online]. Available: https://gdpr.eu/cookies/. [Accessed: 20-Aug-2023]

[21] PHP.net, “Runtime Configuration”, PHP.net. 2023 [Online]. Available: https://www.php.net/manual/en/session.configuration.php. [Accessed: 20-Aug-2023]

[22] PHP.net, “setcookie,” PHP.net, 2023 [Online]. Available: https://www.php.net/manual/en/function.setcookie.php. [Accessed: 20-Aug-2023]

[23] OWASP, “HttpOnly,” OWASP, 2023 [Online]. Available: https://owasp.org/www-community/HttpOnly. [Accessed: 20-Aug-2023]

[24] Anne Trafton, “In the blink of an eye,” MIT News, 2014 [Online]. Available: https://news.mit.edu/2014/in-the-blink-of-an-eye-0116. [Accessed: 20-Aug-2023]

[25] The Ministry of Public Administration and Security (Personal Information Protection Policy), “Basic Measures for Securing Safety of Personal Information,” South Korea, 2019.