Enhancing the Session Security of Zen Cart based on HMAC-SHA256

Lihui Lin, Kaizhi Chen, and Shangping Zhong

College of Mathematics and Computer Science, Fuzhou University

Fuzhou, China

[e-mail: N140327018@fzu.edu.cn; ckz@fzu.edu.cn; spzhong@fzu.edu.cn]

*Corresponding author: Kaizhi Chen

Received August 16, 2016; revised November 14, 2016; accepted November 26, 2016; published January 31, 2017

Abstract

Zen Cart is an open-source online store management system. It is used all over the world because of its stability and safety. Today, Zen Cart's session security mechanism is mainly used to verify user agents and check IP addresses. However, the security in verifying the user agent is lower and checking the IP address can affect the user's experience. This paper, which is based on the idea of session protection as proposed by Ben Adida, takes advantage of the HTML5's sessionStorage property to store the shared keys that are used in HMAC-SHA256 encryption. Moreover, the request path, current timestamp, and parameter are encrypted by using HMAC-SHA256 in the client. The client then submits the result to the web server as per request. Finally, the web server recalculates the HMAC-SHA256 value to validate the request by comparing it with the submitted value. In this way, the Zen Cart's open-source system is reinforced. Owing to the security and integrity of the HMAC-SHA256 algorithm, it can effectively protect the session security. Analysis and experimental results show that this mechanism can effectively protect the session security of Zen Cart without affecting the original performance.

Keywords: Zen Cart, session security, HMAC-SHA256, sessionStorage

https://doi.org/10.3837/tiis.2017.01.025 **ISSN : 1976-7277**

1. Introduction

Nowadays, web application is basically based on HTTP/HTTPS with stateless protocol, which will cause poor user experience for e-commerce sites because it cannot remember whether the user has logged in, what items have been added to the shopping cart, and what other actions have been completed by users. In 1995, Netscape introduced cookies, small chunks of data that a web server can assign to a browser using HTTP return headers, which the browser is expected to send back to the web server on every subsequent request. Using cookies, the web was made stateful [1]. However, the application of cookies exposed a variety of problems. Given that the transmission between the web server and the client is not encrypted, the datum of cookies contains sensitive user information (such as login name and password), which can be easily stolen, thereby resulting in the theft of accounts and the leakage of the user's sensitive information.

Subsequently, session mechanism was developed, which can store the user's information on the web server. The data of each user possesses an identity called session ID that can be sent back to the browser by a cookie. The browser then attaches the session ID in the requested head and the web server will find the corresponding user datum that exists in the web server, which demonstrates stateful access and effectively protects the user information. Obviously, the session ID is susceptible to eavesdropping through HTTP transmission without any encryption. The eavesdropper can use the session ID to establish a connection with the web server, and the web server cannot identify whether the connection is legitimate, thereby resulting in a session hijack.

In view of the problems in the session management mechanism, many scholars have proposed a variety of solutions to solve the problem. BeamAuth [2] is used to defend against phishing attacks for web authentication by a bookmark including a secret token in the fragment identifier. One-Time Cookies [3] proposes to replace the static session identifier with disposable tokens per request, similar to the concept of Kerberos service tickets. Origin-Bound Certificates (OBC) [4] is an extension for TLS that establishes a strong authentication channel between the browser and server, without falling prey to active network attacks. However, TLS-OBC obviously depends on a TLS-only deployment. Juels et al. [5] propose to use "cache cookies" for security: the browser cache stores secret tokens for two-channel authentication at secure sites, e.g., online banking. Shpf [6] proposes to identify based on some features whether the client is from the same user, such as CSS and HTML5.

Many scholars use the hash-based message authentication code (HMAC) [7] algorithm to solve the problems in the session management mechanism. We improve the implementation of SessionLock [1] by saving the shared key in sessionStorage [8], which belongs to the web

storage of HTML5 and is used to store the same session data in local and SHA-256 hash function with higher security. BetterAuth [9] proposed an idea that is similar to ours. The difference is that the BetterAuth employs the user password as a shared key. Relative to our method, the BetterAuth concept is too dependent on the user password. If the user password is so simple, then the security is reduced. HTTP integrity header [10] also proposed an idea that is similar to ours. The difference is that the method of using a shared key between the client and server is based on Diffie–Hellman algorithm. However, the HTTP integrity header uses the original Diffie–Hellman protocol, which only establishes a secret token at the client after the first request and response have been exchanged. This approach leaves the setup phase of the session vulnerable to passive network attacks. SecSess [11] proposed an idea that is similar to ours. The difference is that the method requires access to the contents of the request message, so the deployment is relatively cumbersome and has low availability.

Zen Cart employs the following mechanisms to solve the problems in the session management mechanism. The user agent is verified and the IP address is checked. The user agent is part of the HTTP protocol header and the header field, which is a special string head that provides a browser type and version and operating system version, browser kernel, and other identification information necessary to access the site. Zen Cart validates the user agent in the request header to determine whether it is equal to the user agent stored in the server and to decide whether to accept the request. However, the string head of the user agent is limited and fixed, and the attacker, by collecting user's browser and operating system information, can be easily constructed by the same user agent; thus, security is extremely low. At the same time, Zen Cart can protect the session security by checking whether the IP address is equal. To a certain extent, the effect of session protection can be achieved. However, with the widespread use of wireless networks (Wi-Fi) today, users may be browsing the Internet through free public Wi-Fi at any time, which greatly reduces its security. If the user is in a local area network (LAN), then all LAN users have the same IP address on the outside, and checking the IP address cannot prevent an attack from the same LAN. At the same time, checking the IP address also severely affects the user experience because the user may be using proxy network access, and the IP address of the proxy network is not fixed, thereby causing the system to always ask the user to re-login.

In this paper, the specific implementation process is shown in **Fig. 1**. When a user logs in to the system, the server assigns a key *token* to the browser, and the browser stores the key *token* in sessionStorage. In the next request of the user, the program will obtain the request path, the parameters of the request, and the current timestamp (*timestamp)* and use the key *token* to encrypt the data by HMAC-SHA256. The results of the encryption *sig* (In the following, the HMAC-SHA256 encryption result will be expressed by HS-value) are attached to the message in the request. When the web server receives the request, using the timestamp (*timestamp*) determines whether the request is in accordance with the time range. If the request is within the specified time, then the save key is used to recalculate the

HMAC-SHA256 value. If the two results are equal to receive the request, then it is rejected. If the request is not within the specified time, then it is refused. Through such a protection mechanism, the session security is effectively protected. Even if attackers steal the session ID, obtaining the key and getting the same HS-value are still necessary for successful access. If attackers want to crack the key by signing *sig* and requesting the path, parameters, and current timestamp (*timestamp*), then an attacker needs to complete the break within the user login time. At the same time, we set the same session of the HS-value error several times, and the user will be required to re-enter the password, thereby increasing the difficulty of the crack. Thus, this mechanism can have a certain degree of effective protection of system session security.

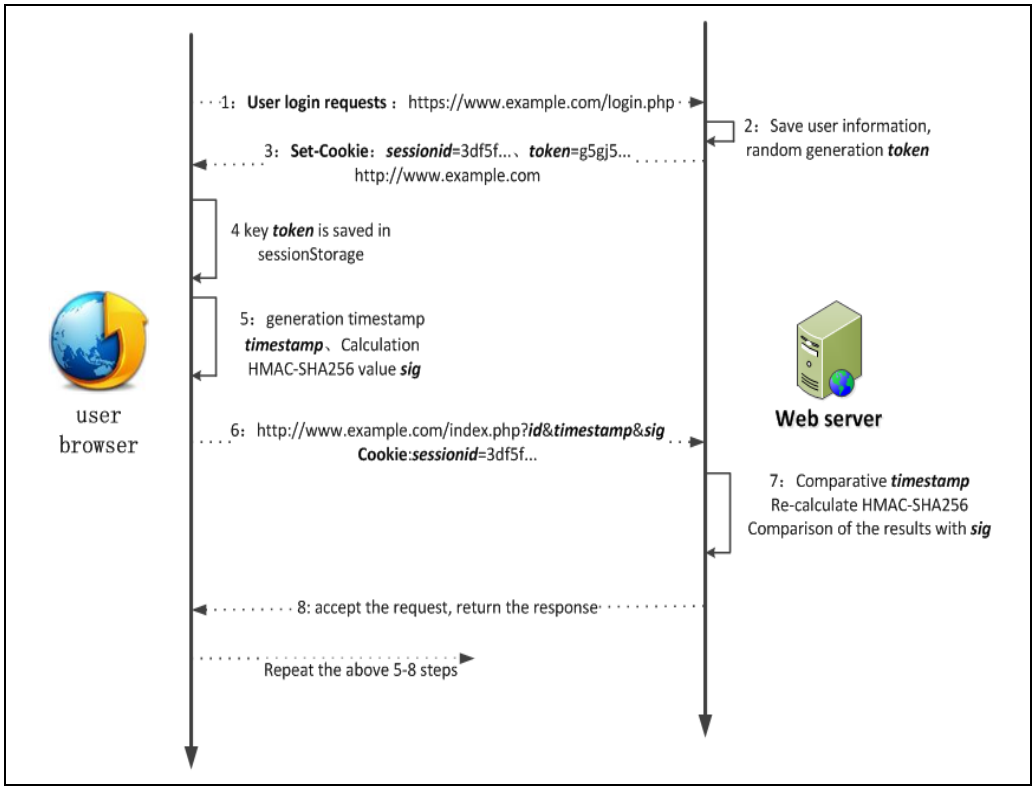

Fig. 1. Session reinforcement process

2. Related Technology

2.1 Introduction to Zen Cart

Zen Cart is an online store management system that is based on PHP language, MySQL database, and HTML components and can support numerous languages and currencies. Zen Cart is freely available under the GNU General Public License [12]. Zen Cart is widely applied around the world and, according to statistics on Common Vulnerabilities and

Exposures [13], a total of 19 vulnerabilities were discovered as of 2015. The vulnerability in version V1.51 has been remediated. Although those vulnerabilities do not contain session management, potential threats still exist. The current version of Zen Cart, as long as the user disables the cookie such that it cannot be logged on, can affect the user experience. The mechanism adopted in this study can solve this problem effectively. At the same time, this mechanism can prevent cross-site request forgery attacks.

2.2 Introduction to current session security

In the cookie-based session management mechanism, the web server randomly generates a session identifier and passes through the set-cookie header back to the browser, and then the browser saves the session ID. When the browser accesses the same domain website, it will send the cookie of the same domain to the server by attaching it to the HTTP header. The set cookie contains the following six properties:

- name: Set cookie name.
- value: Set cookie value.
- expire: Set effective period of cookie; if the value is 0, then turn off browser. Cookie is invalid.
- path: Set cookie server path.
- domain: Set valid domain.
- secure: Set whether to transfer cookie through a secure HTTPS connection.
- HttpOnly: Settings can be accessed only in the HTTP protocol environment, and client script languages cannot be accessed (PHP5.2 added).

Among these properties, the secure and HttpOnly attributes can increase the session security to a certain degree. By setting the attribute value of secure to true, the cookie can be specified only through the HTTPS transmission, which ensures that the cookie can only be transmitted by encryption, thereby protecting the session identification and improving the session security. By setting the attribute value of HttpOnly to true, cookies are available only in the HTTP protocol and client scripting languages cannot be obtained, effectively preventing the attacker from accessing the session identifier by XSS (cross-site scripting attacks). Unfortunately, owing to the user's sloppiness and the limitations of the session mechanism itself, stealing the session identifier is easy, resulting in unauthorized session connection, such as common session hijacking attack and session fixed attack. Session management vulnerabilities rank second in the OWASP top 10 web application vulnerabilities [14]. Visible and current web applications have a high probability of existence in the session management mechanism vulnerability.

Currently, the main threats to session security are the following: 1) Session hijacking attack: The attackers obtain a session ID of the user through script attack or eavesdropping on the network, and then by manually modifying the session ID in the request message, the attacker can establish unauthorized transmission with the server. While the secure and

HttpOnly attributes of cookies can prevent the session identifier from gaining access directly, the service rate of HTTPS is not high and therefore affects the performance, so it is unsuitable to be promoted at present. At the same time, simply setting the attribute value cannot be prevent eavesdropping. 2) Session fixation attacks: Attackers use session identification constant vulnerability of the web server and then use social engineering to induce the usage of the session identification that the attacker sets to the login server, causing the attacker to obtain the session identifier directly, thereby establishing a connection with the web server.

For the session security threats, the following measures have been taken: 1) As mentioned, we can set the secure and HttpOnly attributes to enhance the security of the session. 2) Digest authentication based on HTTP protocol layer [15], and the mechanism generates a message authentication code by encrypting the user name and password to avoid the plaintext of the user name and password being directly transmitted between the server and client. Thus, to a certain degree, digest authentication can protect the session from eavesdropping. However, the application is not widespread because the user experience is not good, among many other reasons. 3) Binding the session and IP address together, as mentioned in Section 1, because of the popularity of wireless networks, many users have public network access to the site, greatly decreasing the effectiveness of the method. 4) Using SSL encryption transmission, the eavesdropper can be prevented from directly stealing the session identification and getting only the encrypted string, which can protect the user's session identification and the security of the session. 5) When a user logs in, regenerating the session identification prevents session fixed attacks and other malicious threats.

2.3 HMAC-SHA256

HMAC-SHA256 refers to the use of hash function SHA-256 computing message authentication code. As shown in Algorithm 1 [16], the algorithm takes the shared key *K* and message *M* as the input to perform the operation. The shared key *K* length is 256 bits in the SHA-256. If we generate less than 256-bit shared key, then the algorithm will be automatically filled with 0 to 256 bits.

ALGORITHM 1: HMAC-SHA256

Input: K, M
Output: HMAC (K, M)

1. $HMAC(K, M) = SHA_{256}(K \oplus opad \mid SHA_{256}(K \oplus ipad \mid M))$

where *K* represents the authentication password, *M* represents a message input, *B* indicates the SHA-256 block size, *OPAD* 0x5C is repeated B times, *iPad* 0x36 is repeated B times, and operations $\oplus$ and $\mid$ represent the XOR and series, respectively.

In this article, when the user is logged in, the web server randomly generates a string and calculates the MD5 value of the string, the value as a shared key *K*, the client request path, parameters and the current timestamp for the input message *M*, and operation of HMAC-SHA256.

2.4 sessionStorage

sessionStorage provided by HTML5 is a new method of storing data in the client side. It is the data storage for one session, similar to the session mechanism. When the user closes the browser, the data are deleted. At the same time, sessionStorage supports the same-origin policy, and the page of different sources cannot obtain variable values from each other. Most of the mainstream browsers now support the sessionStorage, as shown in **Table 1** [8].

Table 1. Browser of support sessionStorage

Feature	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari (WebKit)
sessionStorage	5	2	8	10.50	4

In this paper, we use sessionStorage to store the shared secret key that is used for HMAC-SHA256 encryption. When the browser wants to send a request to the web server, the shared key is obtained from the sessionStorage to calculate the HMAC-SHA256 value. Thus, the shared key need not be obtained from the server every time and the shared key need not be placed in the URL fragment identifier [17].

3. Session encryption mechanism based on HMAC-SHA256

In this study, the session encryption mechanism is divided into three stages: shared-key generation stage, page processing stage, and request/response stage. The shared-key generation stage can be skipped after user login. In this mechanism, our discussion is based on the premise of shared key secure transmission and safekeeping in sessionStorage. We assume that the Zen Cart system does not have other unknown vulnerabilities that will cause the shared key to be stolen.

3.1 Shared-key Generation Stage

The shared-key generation stage is the stage where the server generates the shared secret key and passes it to the client. Users need to enter the username and password to the login system. The process is shown in **Fig. 2**.

Step 1: User inputs the username (*username*) and password (*passwd*) in the login page.

Step 2: The user browser submits the *username* and *passwd* to the web server.

Step 3: After the validation of the web server, a string is randomly generated and calculated for the string MD5 value, and the string MD5 value is used as a shared key *token* that can improve the complexity and security of shared keys. At the same time, the key *token*

is saved in the server and returned to the client.

Step 4: The user browser saves the shared key *token* in sessionStorage.

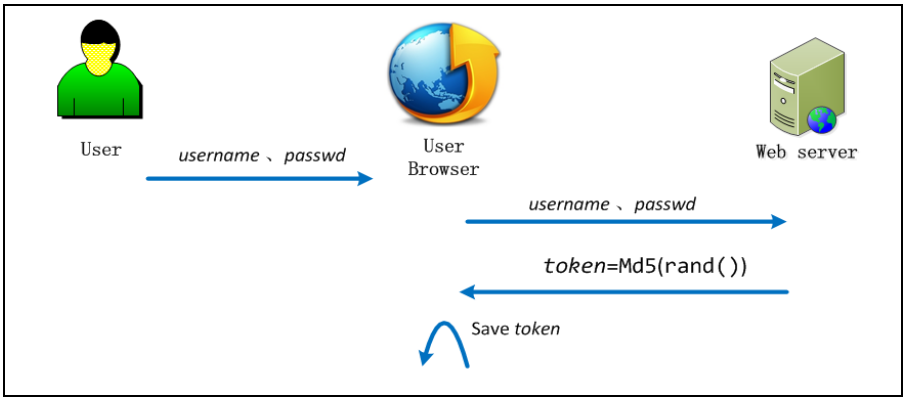

Fig. 2. Process of shared-key generation stage

3.2 Page Processing Stage

The page processing stage handles triggering events (such as onclick, onsubmit, and so on) of all request links in the current loading page. The process is shown in **Fig. 3**. When the page is loaded, all request link elements in the page are saved, which are mainly <*a*><*form*> and <*tr*> in the Zen Cart system, and their triggering event (onclick or onsubmit) is changed. Thus, as timestamp and HS-value are automatically attached, the request is triggered.

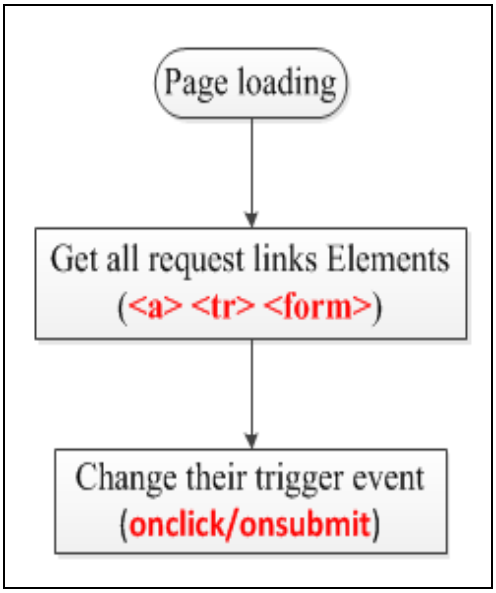

Fig. 3. Process of page processing stage

3.3 Request/Response Stage

The request/response stage is the stage where the user submits the request to the web server and receives a response. Owing to the operation of the page processing stage, it will trigger onclick or onsubmit event of the request link before the request. This type of event adds a timestamp and HS-value (HMAC-SHA256 value) parameter to the request and submits it to the server. The server verifies whether the current timestamp and HS-value are legitimate. The server responds to the request after authentication; otherwise, the request is refused. The process is shown in **Fig. 4**.

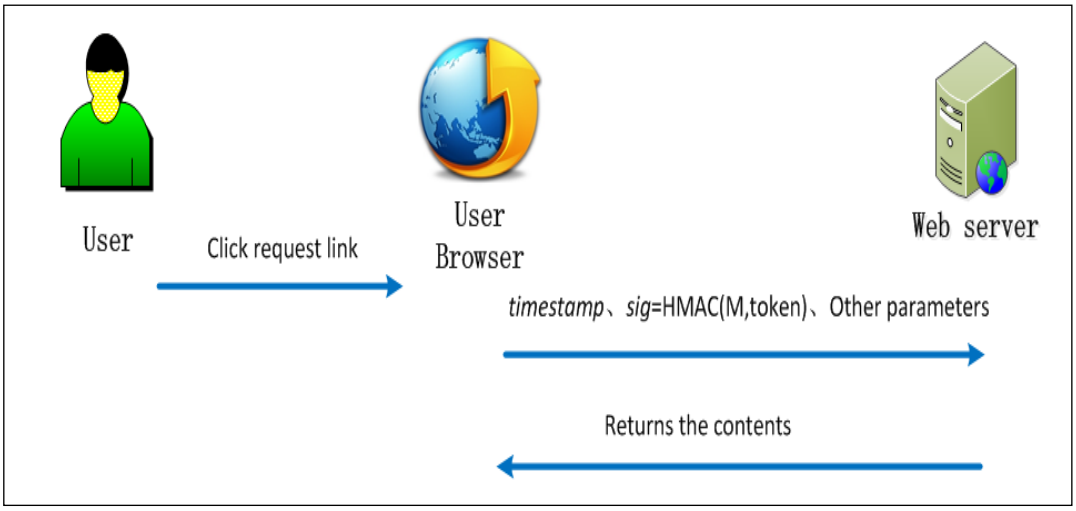

Fig. 4. Process of request/response stage

Step 1: User clicks the request link to trigger events.

Step 2: The event obtains the current timestamp (*timestamp*) and the current request parameter as a message *M* input and then takes the shared key *K*, which is saved in sessionStorage in the login phase, as the key to calculate the HMAC-SHA256 value *sig*. Finally, it submits the HMAC-SHA256 value (*sig*) and timestamp (*timestamp*) and the other parameters to the web server.

Step 3: The web server obtains the current timestamp first and compares it with the submitted timestamp to verify whether it is within the specified time range (the time range can be set by itself). If yes, the web server recalculates the HMAC-SHA256 value based on the shared key that is saved in the web server in the shared-key generation stage. If the value and *sig* are equal, the server will return the response content; otherwise, the request will be refused.

4. Implementation and Result Analysis

4.1 Integration into Zen Cart system

This paper presents an experiment on the Chinese version V1.51 of Zen Cart, but this technology is secondarily developed according to the requirements of laboratory projects, without changing its general frame. In particular, the following path is relative to the root path of the background or foreground system.

4.1.1 Foreground System

The Zen Cart's foreground system is presented as a hub with the summing point file "index.php." Thus, we find the header file in the "index.php" and introduce the script file of adding HS-value and timestamp. The specific code is shown in **Fig. 5** and the path is */includes/templates/template_default/common/html_header.php*.

Furthermore, the code for generating the shared key *token* is added to the login page, and the new shared key, code of login, and new session sign are set. The specific code is shown in **Fig. 6** and the path is */includes/modules/pages/login/header_php.php*.

Moreover, the new shared key set by set-cookie, the code of login, and new session are added to the logoff page. The specific code is shown in **Fig.7** and the path is */includes/modules/pages/logoff/header_php.php*.

Finally, the *token_get_ssl* folder is added to path */includes/modules/pages/*, and the folder named *header_php.php* is created. The file code is shown in **Fig. 8.**

```
<!-- the implementation code of parsing URL -->
<script src="/sessionlock/static/parseuri.js" language="javascript"></script>
<!--the implementation code of SHA-256 function -->
<script src="/sessionlock/static/hmac.js" language="javascript"></script>
<!--the implementation code of add HASH-SHA256 value and timestamp  -->
<script src="/sessionlock/static/locksession.js" language="javascript"></script>
```

Fig. 5. Introducing the script files

```
if (SESSION_RECREATE == 'True') {
  //Landing success . update session_id
  zen_session_recreate();
  //Generating shared key
  $token=md5(uniqid(rand(), true));
  //Set token flag to identify whether the shared key is updated
  zen_setcookie('token',$token,0, '/', (zen_not_null($current_domain) ? $current_domain : ''),TRUE);
  //Set newSession flag to identify whether the session is newly generated
  zen_setcookie('newSession', $current_domain,0, '/',
  (zen_not_null($current_domain) ? $current_domain : ''),FALSE);
  //Set login flag to identify whether the user has logged in
  zen_setcookie('login','1',0, '/', (zen_not_null($current_domain) ? $current_domain : ''),FALSE);
  $_SESSION['token']=$token;
}
```

Fig. 6. Generating shared keys and set-cookie

```php
if (!empty($_SESSION['customer_id']) || !empty($_SESSION['customer_guest_id'])) {
  //Destroy session
  zen_session_destroy();
  //Set the token value to '';
  zen_setcookie('token',  '',0, '/', (zen_not_null($current_domain) ? $current_domain : ''),TRUE);
  //Set the newSession value to '';
  zen_setcookie('newSession','',0, '/', (zen_not_null($current_domain) ? $current_domain : ''),FALSE);
  //Set the login value to '';
  zen_setcookie('login','',0, '/', (zen_not_null($current_domain) ? $current_domain : ''),FALSE);
  zen_redirect(zen_href_link(FILENAME_LOGOFF, $logoff_lang));
}
```

Fig. 7. Signing off set-cookies

```php
<script language="javascript">
SSL_PREFIX = "https://";
var token=Cookie.get('token');

if (!token || token == '') {
   // return the token setup page,which use to generate shared key
    document.location.replace(SSL_PREFIX + document.location.hostname +
    '/sessionlock/templates/locksession/token_setup.html?return_url=' +
    encodeURIComponent(document.location));
} else {
    //Return shared key value
    document.location.replace('http://' + document.location.hostname +
    '/sessionlock/templates/locksession/token_get.html#' + token);
}
</script>

<?php
require(DIR_WS_MODULES . zen_get_module_directory('require_languages.php'));
$breadcrumb->add(NAVBAR_TITLE);

//Clear new session flag
zen_setcookie('newSession','',0, '/', (zen_not_null($current_domain) ? $current_domain : ''),FALSE);

// include template specific file name defines
$define_page = zen_get_file_directory(DIR_WS_LANGUAGES . $_SESSION['language'] . '/html_includes/',
FILENAME_DEFINE_PAGE_2, 'false');
?>
```

Fig. 8. Token_get_ssl file content

4.1.2 Background System

In the Zen Cart background system, the structure of the web page is not the same as the foreground system. However, each function corresponds to a page. Therefore, differences exist in the integration of the foreground system. The added code for each step is basically similar to the foreground system.

First, we should introduce the script file of the added HS-value and timestamp. Considering that the background system of each page contains */includes/header.php* file, we added the code for introducing the script file in this file. However, the script file should be introduced separately in each page so that the other scripts on the page would not be affected. Second, the function of the verified username and password adds the code for generating a shared key and setting the flags of the new shared key, code of login, and new session by the set-cookie. The specific path is */includes/functions/admin_access. php*.

Then, the logoff page adds the code for signing off the flags of the new shared key, code

of login, and new session. The specific path is */logoff.php*.

Finally, we need to create the *token_get_ssl.php* file under the background system root path to obtain the shared key.

4.1.3 Server-side Code

Regardless of the foreground or background system, the *verify_hmac.php* file needs to be added to the corresponding folder under the path */includes/init_includes/*. The code of the file is similar, whereas some of the names are changed, as shown in **Fig. 9.** Then, the *verity_hmac.php* file needs to be included in the path */includes/application_top.php*.

```
//To determine if the user is logged in
if(isset($_SESSION['customer_id']) && $_SESSION['customer_id'] != '')
{

    if(isset($_GET['ls_timestamp']) && (time()-$_GET['ls_timestamp'])>1800000)
    {
        zen_session_destroy();
        zen_setcookie('token',  '',0, '/', (zen_not_null($current_domain) ? $current_domain : ''),TRUE);
        zen_redirect(zen_href_link(FILENAME_LOGIN, '', 'SSL'));
    }
    elseif (isset($_POST['ls_timestamp']) && (time()-$_POST['ls_timestamp'])>1800000)
    {
        zen_session_destroy();
        zen_setcookie('token',  '',0, '/', (zen_not_null($current_domain) ? $current_domain : ''),TRUE);
        zen_redirect(zen_href_link(FILENAME_LOGIN, '', 'SSL'));
    }
    //Gets the result of client encryption
    if(isset($_GET['ls_sig']))
    {
        $c_hmac=$_GET['ls_sig'];
        if($c_hmac != "")
        {
            $query_string=$_SERVER['QUERY_STRING'];
            $query_string=str_replace("&ls_sig=".$c_hmac,"",$query_string);
            $full_url=$_SERVER['PHP_SELF'].'?'.$query_string;
            //To judge whether the HMAC value is equal, if not the same,
            //the current session is canceled, transferred to the login page
            if($c_hmac != hash_hmac("sha256",$full_url,$_SESSION['token'])){
              zen_session_destroy();
              zen_setcookie('token',  '',0, '/', (zen_not_null($current_domain) ? $current_domain : ''),TRUE);
              zen_redirect(zen_href_link(FILENAME_LOGIN, '', 'SSL'));
            }
        }
    }elseif (isset($_POST['ls_sig'])){
        $c_hmac=$_POST['ls_sig'];
        if($c_hmac != "")
        {
            $full_url=$_SERVER['PHP_SELF'].'?';
            ksort($_POST);
            $para="";
            foreach($_POST as $key=>$value){
                if($key =="ls_sig" || $key == 'x' || $key == 'y' || $key == "products_delivery" ||
                 $key == "btn_submit_x" || $key == "btn_submit_y" ){
                  continue;
                }else{
                  $para=$para.$key."=".rawurlencode($value)."&";
                }
            }
            $full_url=$full_url."".substr($para, 0,$para.strripos("&")-1);
            if($c_hmac != hash_hmac("sha256",$full_url,$_SESSION['token']))
            {
              zen_session_destroy();
              zen_setcookie('token',  '',0, '/', (zen_not_null($current_domain) ? $current_domain : ''),TRUE);
              zen_redirect(zen_href_link(FILENAME_LOGIN, '', 'SSL'));
            }
        }
    }
}
```

Fig. 9. Server-side authentication HS-value code

4.2 Result Analysis

After this proposed mechanism is integrated into Zen Cart, the result shows that the foreground and background systems are highly compatible, the user basically cannot perceive the difference of time, and the safety of the session is effectively protected.

In terms of performance, the experimental platform is shown in **Table 2**. We obtain the Chrome browser loading time to compare the time between adding the HS-value and not adding the HS-value. The results are shown in **Fig. 10**. From all the data in the graph, the loading time with the HS-value takes more time than that with no HS-value, and the average

time over cost is 28.32 *ms*. In addition, only two of these data are more than 50 *ms*. Based on the time taken, when the HS-value is less than having no HS-value, it is related to the network environment at that time, but is basically relatively stable.

Table 2. Experimental platform

Operating environment	Details
Hardware environment	Common PC, CPU Intel (R) Core (TM) i5-4460 CPU @ 3.20 GHz, Memory 8 G, Hard disk 500 G
Software environment	Operating system: Windows 7 Service Pack 1 Browser: Chrome, version 51.0.2704.103
Network environment	Able to access LAN

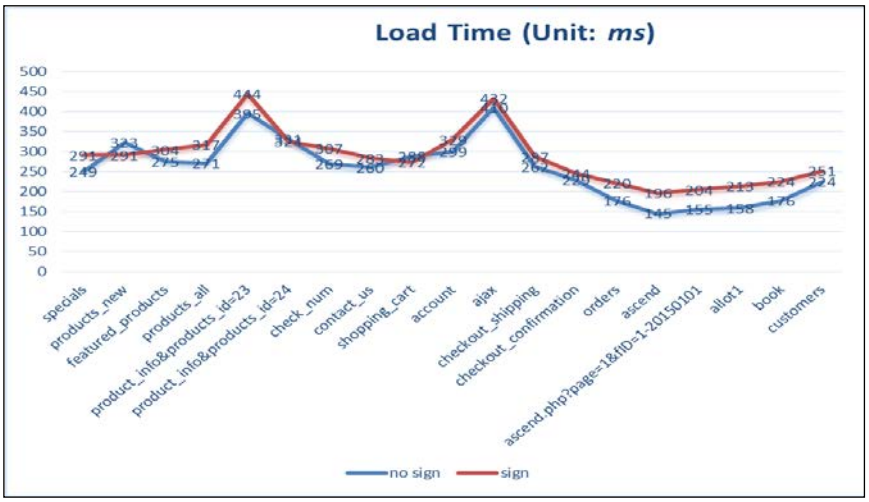

Fig. 10. Loading time contrast

5. Security Analysis

The mechanism proposed in this paper can effectively prevent replay attacks, guessing attacks, session hijacking, and fixed attacks. The security analysis is based on the properties of hash function described as follows [18] [19]:

Property 1. Pre-image resistance (one-way): Essentially for all of the pre-specified output, finding any input unfeasibly makes the hash value equal to the output in the calculation. That is, given as $Y = H(M)$, and find M', where $Y = H(M')$, is difficult.

Property 2. The second pre-image resistance: For any given input, finding any of the second inputs that are equal to the given input is unfeasible. That is, given as M and $H(M)$, and finding $M \neq M'$ is difficult, so that $H(M) = H(M')$.

Property 3. Collision resistance: Finding a set of $M \neq M'$ is difficult, so that $H(M) = H(M')$.

This paper uses the SHA-256 hash function. SHA-256 is more secure than MD5 and SHA1 in resisting birthday attacks and existing differential attacks. At present, SHA256 is considered as one of the most secure hash functions [20].

At the same time, Fig. 11 shows the attack tree [21] for the HS-value threat by using SeaMonster security modeling software. The root of the tree represents the name of the threat, which is "the HS-value threat," and the attacks are represented as nodes in the tree [22]. According to the attack tree, we can see that the security threats of this mechanism mainly include two aspects: guessing the shared secret key and stealing the HS-value. We can steal the HS-value by controlling the user's IP routing, phishing sites, and other ways. After obtaining the hash value, the attacker can only use the way of replay attacks and cannot modify the path and parameters of the request. According to the integrity of the HASH function, if the attacker changes the request of the path and parameters, then the HASH value will change. For the replay attack, the security is discussed in Section 5.1. Guessing the shared secret key is actually a guessing attack, which is analyzed in Section 5.2. Finally, Section 5.3 discusses the session hijacking and session fixed attack, which is the common attack mode of the session mechanism.

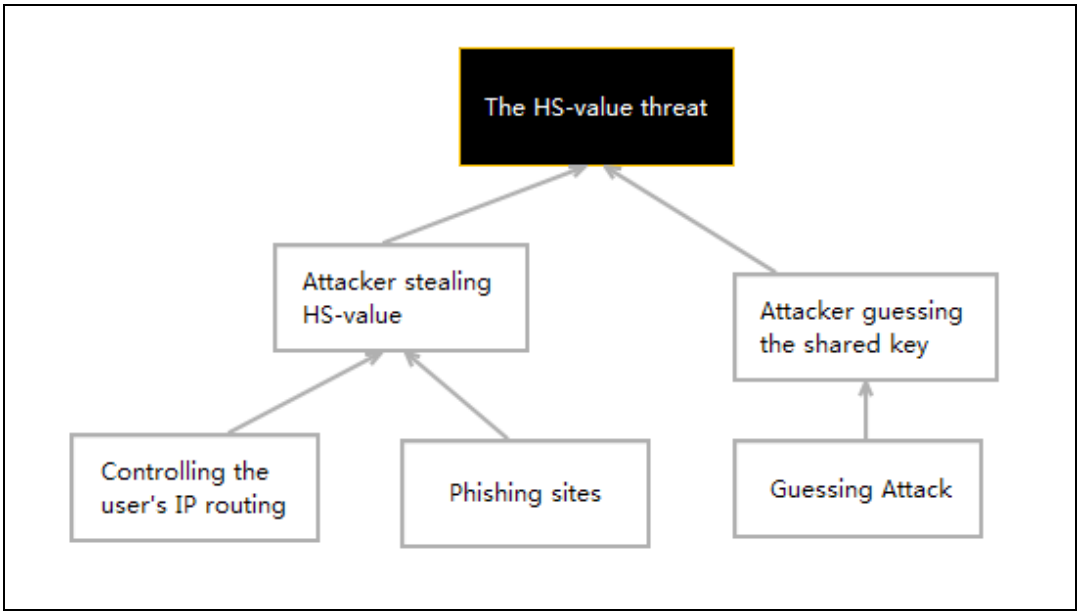

Fig. 11. Attack tree for hash value threat

5.1 Replay Attack

To prevent replay attacks, we add a timestamp to the request to set a time limitation. The timestamp is used as the input of the message when calculating the HMAC value. Thus, in accordance with the second property of hash, it is infeasible that the attacker wants to regenerate a timestamp for the request and obtain the server response.

5.2 Guessing Attack

It is infeasible that an attacker wants to crack the key *K* by obtaining a number of requests M and H(M) (the message and HS-value). For one thing, the key *K* is valid only when the user logs in. When the user exits, the key is reallocated. For another, when a request has been rejected several times, the password has to be re-entered.

5.3 Session Hijacking and Fixed Attack

The attackers obtain the session identification through a certain method, such as controlling the user's IP routing, phishing sites, and others. However, if the attacker does not have the same HS-value, then the request is rejected by the server. According to the third property of the hash function, even if the attacker can obtain the HS-value in the transmission, it cannot modify the parameters, path, and so on. This condition means that the current HS-value is valid only for the current request, which can effectively prevent the middle man from attacking. If an attacker wants to obtain a HS-value through a crack, according to the preceding analysis, then this attempt is almost impossible. Thus, even if an attacker obtains the session ID, he/she can do nothing. Thus, our mechanism can effectively protect the session security and prevent session hijacking and fixed attacks.

Finally, if the shared key is stolen in the transmission or local position, then the security of this mechanism is seriously reduced. To address this limitation, we have to improve and enhance our mechanism in the future.

6. Conclusions

In this paper, the browser comes with the sessionStorage to save the shared key. To a certain extent, this mechanism can improve the security of the locally saved shared key related to the theory of using fragment identification to save the key, as proposed by Ben Adida [1]. The reason is that the fragment identifier can be read directly in the URL and is easy to steal, but for sessionStorage, the shared key can be obtained only in the case of homology and same session identification. Furthermore, this mechanism uses the SHA-256 hash function to calculate the HMAC value, which improves the difficulty of HMAC value crack and further enhances the security. Finally, deploying the mechanism to Zen Cart can significantly improve the Zen Cart session security and place the performance cost within an acceptable range. Furthermore, the scheme applies to all projects using the Zen Cart frame.

However, this mechanism is implemented by JavaScript on the client side. If the users disable the JavaScript of the site, then the mechanism will not be achieved. Moreover, if the site has XSS vulnerabilities, then the shared key is easy to steal. In this case, the security of this mechanism will be seriously reduced. Finally, when the user logs into the site, the site needs to use the HTTPS protocol. If the site is not configured, then the shared key will be transmitted in clear text. Thus, the security will be weakened. These limitations have to be

overcome in the future.

Acknowledgments

The research is supported by the Educational Research Project for Middle-aged and YoungTe achers of Fujian Province (Grant JA15066), and the Science and Technology Development F und of Fuzhou University (Grant 2014-XY-20).

References

[1] Adida, Ben, "Sessionlock: securing web sessions against eavesdropping," in *Proc. of 17th international conference on World Wide Web*, pp. 517-524, April 21-25, 2008. Article (CrossRef Link).

[2] Adida, Ben, "Beamauth: two-factor web authentication with a bookmark," in *Proc. of 14th ACM conference on Computer and communications security*, pp. 48-57, 2007. Article (CrossRef Link).

[3] Dacosta, I., Chakradeo, S., Ahamad, M. and Traynor, P, "One-time cookies: preventing session hijacking attacks with stateless authentication tokens," *Acm Transactions on Internet Technology*, vol. 12, no. 1, pp. 336-345, June.2012. Article (CrossRef Link).

[4] Dietz, M., Czeskis, A., Balfanz, D. and Wallach, D. S., "Origin-bound certificates: a fresh approach to strong client authentication for the web," in *Proc. of 21st USENIX Security Symposium (USENIX Security 12)*, pp. 317-331, August 8-10, 2012. Article (CrossRef Link).

[5] Juels, A., Jakobsson, M. and Jagatic, T. N, "Cache cookies for browser authentication," in *Proc. of 2006 IEEE Symposium on Security and Privacy (S&P'06)*, pp. 301-305, May 21-24, 2006. Article (CrossRef Link).

[6] Unger, T., Mulazzani, M., Frühwirt, D., Huber, M., Schrittwieser, S., and Weippl, E., "Shpf: Enhancing http(s) session security with browser fingerprinting," in *Proc. of 2013 Eighth International Conference on Availability, Reliability and Security (ARES)*, pp.255-261, Sept 2-6, 2013. Article (CrossRef Link).

[7] Krawczyk, Hugo, Mihir Bellare, and Ran Canetti, "Hmac: Keyed-hashing for message authentication," February 1997. Article (CrossRef Link).

[8] Ende93, AlexChao, "Window.sessionStorage," last modified on Oct 27, 2015. Article (CrossRef Link).

[9] Johns, M., Lekies, S., Braun, B. and Flesch, B, "BetterAuth: web authentication revisited," in *Proc. of 28th Annual Computer Security Applications Conference*, pp.169-178, December 03 -07, 2012. Article (CrossRef Link).

[10] Hallam-Baker, Phillip, "Http integrity header," 2012. Article (CrossRef Link).

[11] De Ryck, P., Desmet, L., Piessens, F. and Joosen, W, "SecSess: keeping your session tucked away in your browser," in *Proc. of ACM Symposium on Applied Computing*, pp.2171-2176, April 13-17, 2015. Article (CrossRef Link).

[12] Wikipedia, "Zen Cart ," last modified on August 18, 2016. Article (CrossRef Link).

[13] CVE Details, “Zen-cart : Vulnerability Statistics.” Article (CrossRef Link).

[14] D. Wichers, “Owasp top 10,” OWASP Foundation, 2013. Article (CrossRef Link).

[15] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A. and Stewart, L, “HTTP authentication: Basic and digest access authentication,” No. RFC 2617, 1999. Article (CrossRef Link).

[16] Gebotys, C. H., White, B. A. and Mateos, E, “Preaveraging and carry propagate approaches to side-channel analysis of HMAC-SHA256,” *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 15, no. 1, pp. 1-19, 2016. Article (CrossRef Link).

[17] Berners-Lee T, Fielding R, Masinter L, “Uniform Resource Identifiers (URI): Generic Syntax,” *Staff.kmutt.ac.th*, vol. 4, no. 3, pp. 84 – 87, 2005. Article (CrossRef Link).

[18] Wei, K. J., Lee, J. S., and Chen, S. J, “Enhancing the Security of Credit Card Transaction based on Visual DSC,” *Ksii Transactions on Internet & Information Systems*, vol. 9, no. 3, pp. 1231-1245, 2016. Article (CrossRef Link).

[19] Wei Guo, “Security analysis and construction of chaotic Hash function,” *Southwest Jiaotong University*, China, 2011. Article (CrossRef Link).

[20] HE Run-min and MA Jun, “Analysis safety of SHA-256 algorithm,” *Electronic Design Engineering*, vol.22, no.3, pp.31-33, 2014. Article (CrossRef Link).

[21] Saini, Vineet, Q. Duan, and V. Paruchuri, "Threat modeling using attack trees," *Journal of Computing Sciences in Colleges*, vol.23, no.4, pp.124-131, 2008. Article (CrossRef Link).

[22] Ismail, Reem Jafar. "A Secure Session Management Based on Threat Modeling," *Iraqi Journal of Science*, vol.54, no.4 (5), pp.1176-1182, 2013. Article (CrossRef Link).

Shangping Zhong received his Ph.D. in computer science and technology from Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2005. Now, he is a Professor at the College of Mathematics and Computer Science, Fuzhou University in Fuzhou, China. His research interests include machine learning, intelligent image analysis and information security.

Kaizhi Chen graduated at the College of Information Science and Engineering Southeast University in Nanjing, China in 2011, where he received a Ph.D. in information and communication engineering. Now, he is a lecturer at the College of Mathematics and Computer Science, Fuzhou University in Fuzhou, China. His research interests include intelligent image analysis and machine learning.

Lihui Lin received the B.S. degree in information security in 2014. He is currently pursuing her M.S. degree in computer technology in Fuzhou University, Fujian, China. His current research interests include information security and software development.