A SUPERLATIVE LAMP TECHNOLOGY SAMPLE, COOKIES VS SESSIONS PART I: BATTLE FOR SECURITY

M. C. Luis Manuel Martínez Ramírez
www.ramptors.net/luis
luis.martinez@ramptors.net

Ing. Brenda Esthefania Romero Díaz
www.ramptors.net/esthefania
esthefania.romero@ramptors.net

M. C. Erika López Prado
www.ramptors.net/erika
erika.lopez@ramptors.net

Ing. Jazmín Villegas Soriano
www.ramptors.net/jazmin
jazmin.villegas@ramptors.net

Ing. Jeniffer Alicia Toral Ponce
www.ramptors.net/jenny
jeniffer.toral@ramptors.net

Ing. Verónica Vazquez Rojas
www.ramptors.net/vermox
veronica.vazquez@ramptors.net

M. C. Martín Gregorio Martínez Martínez
www.ramptors.net/goyo
gregorio.martinez@ramptors.net

Universidad Tecnológica de Nezahualcóyotl
Circuito Universidad Tecnológica sin número. Colonia Benito Juárez. Nezahuacóyotl. Estado de México. C. P. 57000

Instituto Tecnológico de Iztapalapa
Avenida Telecomunicaciones sin número. Col. Chinampac. Iztapalapa. Ciudad de México. C. P. 09208

Universidad Autónoma del Estado de México - Ecatepec
Calle José Revueltas 17. Colonia Tierra Blanca. Ecatepec. Estado de México. C. P. 55020

ABSTRACT

It is showed how using piece of data that the web server sends to a web browser called COOKIE to check if two requests come from the same web browser and how SESSION allow to persist data across pages in a web application are used to avoid hackers to LOGIN, MENU and OPTIONS of MENU. And above all, using the technology of the sessions, another hackers attack will be avoided both to the login and to the menu how to any menu option.

INTRODUCTION

Many of the world's original hackers were computer hobbyists, programmers, and students during the 1960s. Originally, the term hacker described people with advanced programming skills. Hackers use these programming skills to test the limits and capabilities of early systems. These early hackers were also involved in the development of the first computer games. Many of these games included paladins and skills. As the hacking culture evolved, he incorporated the lexicon of these games into the culture itself. Even the outside world began to project the image of powerful champions onto this misunderstood hacking culture. Books like Where Wizards Stay up Late: The Origins of The Internet published in 1996 added the mystique of hacking culture. The image and the lexicon stagnated. Many hacking groups today adopt these images. One of the most infamous hacker groups goes by the name of Legion of Doom. Understanding cyber culture is important to understanding cyber world criminals and their motivations. Sun Tzu was a Chinese philosopher and warrior in the 6th century BC. C. Sun Tzu wrote the book called The Art of War which is a classic work on the strategies available to defeat the enemy. His book has guided strategists for centuries. One of Sun Tzu's guiding principles was knowing his opponent. Although he was referring specifically to war, much of his advice translates into other aspects of life, including the challenges of cybersecurity. This chapter begins by explaining the structure of the cybersecurity world and why it continues to grow. This paper will analyze the role of cybercriminals and their motivations attacking the login and how the Technological University of Nezahualcoyotl, the Technological Institute of Iztapalapa and the Autonomous University of Mexico State have solve a strategy how to avoid the attack to LOGIN, MENU, and other Options becoming into cyber heroes help to defeat cyber criminals who threaten the cyber world.

COOKIES. The World Wide Web works based on the HTTP protocol. The HTTP protocol is stateless. When the web browser requests a page from a web server, the web server responds with the page content. Later, the same web browser requests the same page again, the web server has no information that the request is from the same web browser. Then, Cookies solve this stateless challenge. A cookie is a piece of data that a web server sends to the web browser. The web browser may store it and send it back in the subsequent requests to the same web server. By using the same cookie, the web server knows that two requests come from the same web browser. Cookies are also known as web cookies, HTTP cookies, or browser cookies. We'll use the cookies to make it short. The following flow chart illustrates how cookies work:

How it works. First, the web browser sends a request to the web server. The web server doesn't have any information about the web browser. The web server creates a cookie with a name return and a value 1 and attaches the cookie to the HTTP response header. To create a cookie, it will be use the

```
setcookie ( );
```

function. Second, the web browser stores the cookie. Third, the web browser sends the second request with the stored cookie in the header of the HTTP request to the web server. On the web server, PHP can access the cookie via the

```
$_COOKIE
```

the superglobal variable and do something accordingly. Finally, the web server responds with the content of the request. Typically, it responds to the web browser with the content based on the value of the cookie. A web browser can store a cookie with a maximum size of 4KB. However, it's different between web browsers. A cookie has an expiration date. Typically, web browsers store cookies for a specific duration. And the web server can specify the expired time for a cookie. A cookie also stores the web address (URL) that indicates where it comes from. And the web browser only sends back the cookie that was originally set by the same web address. In other words, a website won't be able to read a cookie set by other websites. Most modern web browsers allow users to choose to accept cookies. Therefore, you should not completely rely on cookies for storing critical data.

WHY USING COOKIES. In general, websites use cookies to enhance user experiences. For example, without cookies, it would have to log in to a website again after you leave it. Typically, you'll use cookies for the following purposes:

Session management: cookies allow a website to remember users and their login information or anything else that the web server should remember.

Personalization: cookies can store user's preferences, themes, and other settings.

Tracking: cookies store user behavior. For example, on an Ecomerce website, it can used cookies to record the products that users previously viewed.

Later, it can be used this information to recommend the related products that users might be interested in.

SETTING A COOKIE IN PHP. PHP makes it easy to work with cookies using the `setcookie ( );` function. The `setcookie ( );` function allows you to send an HTTP header to create a cookie on the web browser.

```
<?php

setcookie ( "venta_ID", $venta_ID, time( ) + ( 86400 * 30 ), "/" );

?>
```

The following table illustrates the arguments of the setcookie() function:

Argument	Meaning
$name	The name of the cookie
$value	The value of the cookie. It can be any scalar value such as string or integer.
$expires	The time (in a UNIX timestamp) the cookie expires. If $expires is not set or set to 0, the cookie will expire when the web browser closes.
$path	The path on the webserver on which the cookie will be available. For example, if the path is '/', the cookie will be available within the domain.
$domain	The domain to which the cookie will be available.
$secure	if $secure is set to true , the cookie should be transmitted over a secured HTTP (HTTPS) connection from the web browser.
$httponly	if $httponly is true, the cookie can be accessed only via the HTTP protocol, not JavaScript.

The `$option` argument is an array that has one or more keys such as `expires`, `path`, `domain`, `secure`, `httponly` and `samesite`. The `samesite` can take a value of `None`, `Lax`, or `Strict`. If you use any other key, the `setcookie ( );` function will raise a warning.

The `setcookie ( );` function returns `true` if it successfully executes. Notice that it doesn't indicate whether the web browser accepts the cookie or not. The `setcookie ( );` function returns `false` if it fails.

$_COOKIE. The `$_COOKIE` an associative array that stores the HTTP cookies. To access a cookie by a name, you use the following syntax:

```
<?php

$_COOKIE[cookie_name];

?>
```

If the cookie name contains dots (.) and spaces (' '), you need to replace them with underscores (_). To check if a cookie is set, you use the `isset ( )` function:

```
<?php

if(isset($_COOKIE[cookie_name]))
    {
    }
?>
```

Since `$_COOKIE` is a superglobal variable, it can be can access it anywhere in the script.

READING A COOKIE. Before reading a cookie value, it should always been checked if it has been set by using the `isset ( )` function:

```
<?php

$cookie_name1 = "venta_ID";
$cookie_value1 = "0";

if ( isset ( $_COOKIE [ $cookie_name1 ] ) )

    {

    }

?>
```

DELETING A COOKIE. If you don't use a cookie, you can force the browser to delete it. PHP doesn't provide a function that directly deletes a cookie. However, you can delete a cookie using the `setcookie ( );` function by setting the expiration date to the past. The following code deletes a cookie with the `cookie_name1` in the next page request:

```
<?php

unset($_COOKIE['cookie_name']);

setcookie('cookie_name', null, time()-3600);

?>
```

PHP COOKIE EXAMPLE. The following example shows how to use a cookie to display a greeting message to a new visitor or a returning visitor creating a new ID for venta_ID.

```
<?php
if ( isset ( $_COOKIE [ "venta_ID" ] ) )
            {
                        $venta_ID = $_COOKIE [ "venta_ID" ];
            }
else
            {
                        require "venta.php";
                        $venta_ID = venta_NEW ( 3 , 3 ) ;
                        setcookie ( "venta_ID", $venta_ID, time( )
+ ( 86400 * 30 ), "/" );
            }
?>
```

HOW IT WORKS. First, define a constant that stores one week in

```
<?php

define('ONE_WEEK', 7 * 86400);

?>
```

Second, set the returning_visitor to false:

```
$returning_visitor = false;
```

Third, check the cookie with the name return. If the cookie is not set, create it with the value 1 and the expiration date one week. Otherwise, set the `$returning_visitor` variable to true.

```
if ( isset ( $_COOKIE ['return'] ) )
     {
       $returning_visitor = true;
     }
else
     {
       setcookie('return', '1', time() + ONE_WEEK);
     }
```

Finally, display the greeting message based on the value of the `$returning_visitor` variable. And if it is opened the web developer tool, you'll see the cookie as shown in the following picture:

Since the web browser already stores the cookie with the name return and value of venta_ID. This cookie will last for 30 days set by the web server. Of course, from the web browser, it can manually delete the cookie.

SESSIONS. The HTTP protocol is stateless. For example, when you visit the product page `product.php`, the web server responds with the page:

Suppose, it is clicked the add to cart button on the `product.php` page and navigate to the `cart.php` page, the web server won't know that you have added the product to the cart. To persist the information across the pages, the web server uses sessions. In this example, when you click the add to cart button, the web server will store the product on the server. When you view the `cart.php` page, the web server gets the products from the session and displays them on the `cart.php` page:

HOW IT WORKS. First, the web browser requests for the product.php page. Second, the web server responds with the product.php page's content. Third, you click the Add To Cart button on the product.php page. The page will send an HTTP request (either POST or GET) to the web server. The web server validates the product and generates a session id. It also creates a new text file on the server to store the information related to the selected product. Fourth, the web server responds to the web browser with the PHPSESSID cookie in the response header. If the web browser allows cookies, it will save the PHPSESSID cookie, which stores the session id passed by the web server. Fifth, in the subsequent request, for example, when you view the cart.php page, the web browser passes the PHPSESSID back to the web server. When the web server sees the PHPSESSID cookie, it will resume the session with the session id stored in the cookie. Finally, the web server returns the cart page with the products that you selected.

Sessions allow it is stored data on the web server associated with a session id. Once you create a session, PHP sends a cookie that contains the session id to the web browser. In the subsequent requests, the web browser sends the session id cookie back to the web server so that PHP can retrieve the data based on the session.

CREATING A NEW SESSION. To create a new session, it can be called the `session_start ( );` function:

```
<?php

session_start();

?>
```

When the `session_start ( );` runs at the first time, PHP generates a unique session id and passes it to the web browser in the form of a cookie named PHPSESSID. If a session already exists, PHP checks the PHPSESSID cookie sent by the browser, the `session_start ( );` function will resume the existing session instead of creating a new one.

Since PHP sends the PHPSESSID cookie in the header of the HTTP response, you need to call the `session_start ( );` function before any statement that outputs the content to the web browser. Otherwise, you will get a warning message saying that the header cannot be modified because it is already sent. This is a well-know error message in PHP.

WHERE PHP STORES SESSION DATA. By default, PHP stores session data in temporary files on the web server. You can find the location of the temporary files using directive `session.save_path` in the PHP configuration file. The `ini_get ( );` function returns the value of the session.save_path directive:

```
<?php

echo ini_get('session.save_path');

?>
```

Or you can call the `session.save_path ( );` function:

```
<?php

echo session_save_path();

?>
```

Typically, the session data is stored in the /tmp folder of the web server e.g, /AMP/www/tmp.

ACCESSING SESSION DATA. Unlike cookies, it can be stored any data in the session. To store data in the session, you set the key and value in the `$_SESSION` superglobal array. For example, in the index.php file, it is stored the user string and roles array in the session as follows:

```
<?php
if ( isset ( $_SESSION ) )
     {
       // NADA
     }
else
     {
       session_start ( );
     }
?>
<?php

if ( isset ( $_SESSION [ MM_Username ] ) &&
     isset ( $_SESSION [ MM_EMail ] ) &&
     isset ( $_SESSION [ MM_Url ] )
   )
     {
       $st_USERNAME = $_SESSION [ MM_Username ];
       $st_EM = $_SESSION [ MM_EMail ];
       $st_URL = $_SESSION [ MM_Url ];
```

How it works: First, create a new session by calling the `session_start ( );` function. Second, set the session data with the key user and roles to the 'del_USERNAME' and the array ['Username', 'EM', 'URL']. The del_MENU.php displays the name of the delegate.

DELETING THE SESSION DATA. Whenever you close the web browser, PHP automatically deletes the session. Sometimes, you want to explicitly delete a session, e.g., when you click the logout link. In this case, you can use the `session_destroy();` function:

```
<?php

session_destroy();

?>
```

This `session_destroy();` deletes all data associated with the current session. However, it does not unset data in the `$_SESSION` array and cookie. To completely destroy the session data, it is necesary to unset the variable in `$_SESSION` array and remove like this:

```
<?php

session_start ( );

$_SESSION [ MM_Username ] = NULL;

$_SESSION [ MM_EMail ] = NULL;

$_SESSION [ MM_URL ] = NULL;

unset ( $_SESSION [ MM_Username ] );

unset ( $_SESSION [ MM_EMail ] );

unset ( $_SESSION [ MM_URL ] );

session_unset ( );

session_destroy ( );

header ( "Location: http://ramptors.net/lyt" );

?>
```

Notice that we used the `session_name();` function to get the session name instead of using the PHPSESSID. This is because PHP allows you to work with multiple sessions with different names on the same script.

USE CASE: BATTLE FOR SECURITY – LOG IN. First we review the case when an User wants to login into the web site. The problem is that always the hackers attack the login and the menu. Then UTN Nezahualcoyotl PHP Ramptors have solved this attack using the next strategy. See the following chart:

It shows the technology to avoid hacker attacks without any effort only using one of the PHP magic, THE SESSIONS.

It shows the technology to avoid hacker attacks without any effort only using the technology of the sessions The magic of php


```
<FORM METHOD = "POST";
      ACTION = "del_VALIDATE.php";
      NAME = "FORM1";
      ID = "FORM1";
      >

  <div class="row uniform">
    <div class="12u$">
      <LABEL FOR "del_USERNAME" >Usuario</LABEL>
      <INPUT TYPE = "TEXT";
              NAME = "del_USERNAME";
              ID = "del_USERNAME";
              VALUE = "";
              PLACEHOLDER = "Usuario";
              ONBLUR = "valida_USERNAMExx ( ); ";
             />
    </div><!--class="12u$"-->
  </div><!--class="row uniform"-->

  <div class="row uniform">
    <div class="12u$">
      <LABEL FOR "del_PASSWORD" >Contrase&ntilde;a</LABEL>
      <INPUT TYPE = "PASSWORD";
              NAME = "del_PASSWORD";
              ID = "del_PASSWORD";
              VALUE = "";
              PLACEHOLDER = "Contrase&ntilde;a";
              ONBLUR = "valida_PASSWORDzz ( ); ";
             />
    </div><!--class="12u$"-->
  </div><!--class="row uniform"-->

    <INPUT TYPE = "HIDDEN";
        NAME = "LL_VALIDATE";
        ID = "LL_VALIDATE";
        VALUE = "FORM1"; />

    <INPUT TYPE= "HIDDEN";
        NAME = "MM_VALIDATE";
        ID = "MM_VALIDATE";
        VALUE = "del_VALIDATE"; />
```

So now it routes to validate and valid with a query and it takes care of starting the session

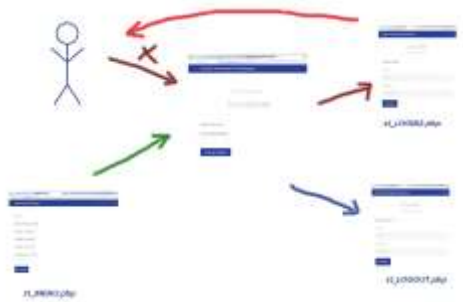
```
<?php
if  (                    ( isset ( $_POST [ LL_VALIDATE ] ) ) &&
                         ( isset ( $_POST [ MM_VALIDATE ] ) ) &&
                         ( $_POST [ LL_VALIDATE ] == "FORM1" ) &&
                         ( $_POST [ MM_VALIDATE ] == "del_VALIDATE"
            )
{
            $del_USERNAME = $_POST [ del_USERNAME ];
            $del_PASSWORD = $_POST [ del_PASSWORD ];
            $del_USERNAMEsha1 = SHA1 ( $del_USERNAME );
            $del_PASSWORDsha1 = SHA1 ( $del_PASSWORD );
            $del_EM = strtolower ( $del_USERNAME );

            include "Connections/conexionUP.php";
            $queryUP = "SELECT del_EM, del_USERNAME FROM delegate
                    WHERE ( del_USERNAME = '$del_USERNAMEsha1' AND
                            del_PASSWORD = '$del_PASSWORDsha1' )
                    OR    ( del_EM = '$del_EM' AND
                            del_PASSWORD = '$del_PASSWORDsha1'); ";
            $resultUP = mysql_query ( $queryUP, $conexionUP );
            include "Connections/desconexionUP.php";

            if ( $rowUP = mysql_fetch_array ( $resultUP ) )
                        {
                        session_start ( );
            $_SESSION [ MM_Username ] = $rowUP[del_USERNAME];
            $_SESSION [ MM_EMail ] = $rowUP[del_EM];
            $_SESSION [ MM_URL ] = "";
            header ( "Location: del_MENU.php" );
                        }
            else
                        {
            header ( "Location: del_LOGIN_FAIL.php" );
                        }
            }
?>
```

Now if the whole session is correct then route to menu and if it is not correct send call del_LOGIN_FAIL.php.

And then the del_MENU.php only takes care of sending the session data and applying it to the username.

```
<?php
if ( isset ( $_SESSION ) )
                {
                        // NADA
                }
else
                {
                        session_start ( );
                }
?>
<?php
if (            isset ( $_SESSION [ MM_Username ] ) &&
                isset ( $_SESSION [ MM_EMail ] ) &&
                isset ( $_SESSION [ MM_Url ] )
   )
                {
                        $st_USERNAME = $_SESSION [ MM_Username ];
                        $st_EM = $_SESSION [ MM_EMail ];
                        $st_URL = $_SESSION [ MM_Url ];
                        include "Connections/conexionUP.php";
                        $queryUP = "SELECT st_NOM
                                        FROM student
                    WHERE st_USERNAME = '$st_USERNAME'
                    OR    st_EM = '$st_EM'; ";
                        $resultUP    =    mysql_query    (    $queryUP,
$conexionUP );
                        $rowUP = mysql_fetch_array ( $resultUP );
                        include "Connections/desconexionUP.php";
                }
else
                {
                        header ( "Location: st_LOGIN.php" );
                }
?>
```

Of this mode if the hackers attack attacan a login but no can attack to the session and if they attack the menu the session or not allows enter or the session is not activated. And If the hackers want to attack one of the menu options, these are independently protected with another validation and another login.



Finally, to LOGOUT check the next code where it shows how the sessions are destroyed.

```
<?php
session_start ( );
$_SESSION [ MM_Username ] = NULL;
$_SESSION [ MM_EMail ] = NULL;
$_SESSION [ MM_URL ] = NULL;
unset ( $_SESSION [ MM_Username ] );
unset ( $_SESSION [ MM_EMail ] );
unset ( $_SESSION [ MM_URL ] );
session_unset ( );
session_destroy ( );
header ( "Location: http://ramptors.net/lyt" );
?>
```

CONCLUSIONS

A cookie is a piece of data that the web server sends to a web browser to check if two requests come from the same web browser. Use the PHP `setcookie ( );` function to set a cookie that is sent along with HTTP header from the web server to the web browser. Use the superglobal variable `$_COOKIE` to access the cookies in PHP. Sessions allow you to persist data across pages in a web application. Call the `session_start ( );` function before any statement that outputs to the web browser for creating a new session or resuming an existing session. Use the `$_SESSION` superglobal array to access the session data. Call the `session_destroy ( );` function to completely delete session data. And above all, using the technology of the sessions, another hackers attack has been avoided both to the login and to the menu how to any menu option.

OBSERVATIONS

For the next two papers, Part 2 and Part 3 it will be shown how to use cookies for security in e-commerce and finally how cookies are combined with sessions and a simple if statement to safeguard the security of a Website without further technology that the PHP Magic.

REFERENCES

[1] Luis Manuel Martínez, Marcia Granciano, Gilberto Pacheco, Juan Mexica, Norberto Vera. An AMP application developed on a noncompatible platform. IEEE Computer Society. (The Institute of Electrical and Electronics Engineers, Inc., 2004). http://www.ramptors.net

[2] Brian Behlenford, Apache Web Server.
(New York: John Wiley & Sons LTD, 1999). http://www.apache.org

[3] Scott Hughes, MySQL Data Base.
(New York: John Wiley & Sons LTD, 2004). http://www.mysql.com

[4] Rasmus Lerdorf, PHP & MySQL.
(New York: Addison Weasley, 2004). http://www.php.net

[5] Kevin Loney, Oracle Database Complete Reference.
(Atlanta: Osborne Press Series, 2009).

[6] Luis Manuel Martínez, Gilberto Pacheco, Juan Mexica, Esmeralda Contreras. Replication Technology on MySQL Server for LAMP Applications Part I, II, III & IV. IEEE Computer Society. (The Institute of Electrical and Electronics Engineers, Inc., 2016). http://www.ramptors.net

CURRICULUM

M. C. Luis Manuel Martínez Ramírez. He studied at Universidad Autónoma Metropolitana Applied Mathematics to Computer Science. Master in Mathematics at Centro de Investigación y Estudios Avanzados. Education Master at Universidad del Valle de México. He had worked at Universidad Tecnológica de Nezahualcóyotl as Researcher Teacher since 1993. IEEE member 41509686.

M. C. Erika López Prado. She studied at Instituto Politecnico Nacional UPIICSA, Computer Science Career. Education Master Science at Universidad Universidad Interamericana para el Desarrollo. Now, she is researching on Languages and Programming. She had worked at Universidad Tecnológica de Nezahualcóyotl as Researcher Teacher since 2002.

Ing. Jeniffer Alicia Toral Ponce. She studied at Instituto Tecnológico de Iztapalapa, Systems Engineer. Now, she is learning the fundamentals of Programming and Servers and Applications Development World Wide Web. She works as a Network teacher at Instituto Tecnológico de Iztapalapa. IEEE member 97938055.

M. C. Martin Gregorio Martínez Martínez He studied at Universidad Autónoma del Estado de Nuevo León. Master in Systems at Universidad de San Carlos. He had worked at Universidad Autónoma del Estado de México as Researcher Teacher since 2004.

Ing. Brenda Esthefania Romero Díaz. She studies at Universidad Tecnológica de Nezahualcóyotl, Information Technology Engineer. Now, she is learning the fundamentals of Programming and Data Bases and Applications Development World Wide Web. Works at Artificial Nerds as QA Engineer. She is the IEEE Student Branch Nezahualcoyotl. IEEE member 97873418.

Ing. Jazmin Villegas Soriano. She studied at Universidad Tecnológica de Nezahualcóyotl, Information Technology Engineer. She is the Systems Computer Career Dean at Instituto Tecnológico de Iztapalapa. Now, she is researching Networks and Protocols. IEEE member 97937973.

Ing. Veronica Vazquez Rojas. She studies at Instituto Tecnológico de Iztapalapa, Systems Engineer. Now, she is learning the fundamentals of Programming and Servers and Applications Development World Wide Web.