تأمين تطبيقات لغة البرمجة PHP من الإختراق (2)

تأليف: محمد عباس الأمين صالح

مقدمة

شرحنا في المقال السابق كيفية تنفيذ وإستغلال الثغرات المتعلقة بتطبيقات لغة البرمجة PHP خطوة بخطوة مثل SQL Injection, XSS, Command Injection, Brute Force and File Uploading, CSRF. في هذا المقال سوف نشرح بإسهاب طرق الحماية من تلك الثغرات بإذن الله. طريقة تسلسل الشرح في هذا المقال يبدأ بحماية الثغرات حسب ترتيبها كما ورد في المقال الأول.

الثغرة الأولى: الاستقصاء في الهجوم على حسابات المستخدمين (Brute Force)

يمكن حماية تطبيق لغة PHP من هجمة Brute Force بواحدة من اربعة طرق مختلفة (يفضل استخدامها كلها من أجل حماية قصوى):

أ) استخدام تقنية **CAPTCHA**:

وهي عبارة عن حروف وارقام مولدة بطريقة عشوائية مطبوعة على شكل صورة يقوم المستخدم بإدخال ما يظهر في الصورة في المكان المخصص لها. في حالة ادخال الرقم المولد الصحيح فان العملية تتم بنجاح وإلا يتم إلغاءها. كلمة CAPTCHA مستوحاة من الجملة التالية:

"Completely Automatic Public Turing Test to Tell Computers and Humans Apart"

وهي تعني باللغة العربية:

"اختبار تورنج العام والإوتوماتيكى للتمييز بين الحاسب والإنسان"

هذه الطريقة فعالة جدا في الوقوف في وجه هجوم **Form Login Brute Force** حيث يمكنك تطبيقها بسهولة وإضافتها الى تطبيقك. الشرح هنا يوضح ذلك (هنا)

ب) السماح لعدد معين من محاولات تسجيل الدخول:

تكمن أهمية هذه الطريقة في انها تقوم بايقاف عملية تسجيل الدخول لزمن معين بعد عدد محدد من المحاولات الفاشلة. الطريقة بسيطة وهي أن تقوم بتصميم خوارزمية عملية تسجيل الدخول حسب مكونات وشكل الإستمارة (Form) المخصصة لتلبي ذلك.

ج) استخدام وسائل وتقنيات للحماية من هجمات **DOS/DDOS**:

DOS/DDOS اختصار لي:

DOS: Denial of Service
DDOS: Distributed Denial of Service

وهي عبارة عن هجمات تؤدي لشل وإيقاف الخدمة المستهدفة حيث يتم ارسال عاصفة ان لم يكن إعصار من الطلبات بغض النظر عن حالة نتيجة هذه الطلبات هل حصلت على استجابة او لا. أيضا نلاحظ ان في هجمات Brute Force يتم محاولة الكثير من المحاولات حتى يتم الحصول على الحساب الصحيح. هذه الهجمات يتم التصدي لها في مخدم الشبكة العنكبوتية (Apache2) عن طريق الاضافة Mod_Evasive. الخطوات التالية توضح كيفية اعداد ذلك. قم بالانتقال الى حساب الرووت الجذر (root) ثم قم بكتابة الأوامر السابقة في الطرفية (terminal) واحدة تلو الأخرة بدون علامة # الارقام (الارقام فقط لتوضيح التسلسل).

1) # apt-get install libapache2-mod-evasive
2) # cp /etc/apache2/mods-available/mod-evasive.load /etc/apache2/mods-enabled/
3) # gedit /etc/apache2/mods-enabled/ mod-evasive.conf &

وبعد تنفيذ الامر الثالث يفتح لك كلف قم بنسخ التالي وضعه فيه:

```
<IfModule mod_evasive20.c>
  DOSHashTableSize 3097
  DOSPageCount 2
  DOSSiteCount 50
  DOSPageInterval 1
  DOSSiteInterval 1
  DOSBlockingPeriod 300
<IfModule/>
```

حيث:

DOSHashTableSize: سعة الجدول المستخدم في متابعة IP المختلفة وهنا 3097 ك.ب.
DOSPageCount: عدد الصفحات المسموح به في الفترة DOSPageInterval. هنا فقط صفحتان في الثانية الواحدة.
DOSSiteCount: عدد الصفحات المسموح به لكل الموقع في الفترة DOSSiteInterval. هنا فقط مسموح لخمسين صفحة في الثانية.
DOSBlockingPeriod: زمن حجب الخدمة وهي محسوبة بالثانية. هنا 3 ثواني.

كما لاحظت هنا المعاملات تأخذ قيم متغيرة يمكنك تغييرها على حسب رغبتك لتلبي متطلباتك.

د) استخدام جدار النار **Mod Security**:

نجد ان Mod Security من أقوى الإضافات التي تضاف الى مخدم الويب (Apache2) حيث يمثل جدار النار على مستوى التطبيق بالنسبة له. يساعد جدار النار Mod Security على التصدي لاغلب ان لم كل الهجمات التي تستهدف مخدم الويب Apache2.

الخطوات التالية توضح كيفية اعداد ذلك. قم بالانتقال الى حساب الرووت الجذر (root) ثم قم بكتابة الأوامر السابقة في الطرفية (terminal) واحدة تلو الأخرة بدون علامة # الارقام (الارقام فقط لتوضيح التسلسل).

1) # apt-get install libapache-mod-security
2) # cp /etc/apache2/mods-available/mod-security.load /etc/apache2/mods-enable/

3) # gedit /etc/apache2/mods-available/mod-security.conf &

الامر الثالث يقوم بفتح ملف، قم بنسخ التالي فيه:

```
<LocationMatch ^/login>
SecAction "initcol:ip=%{REMOTE_ADDR},pass,nolog"

SecRule RESPONSE_BODY "Username does not exist" "phase:4,pass,setvar:
ip.failed_logins=+1,expirevar:ip.failed_logins=60"

SecRule IP:FAILED_LOGINS "@gt 3" deny
<Location/>
```

حيث يقوم بمنع الوصول الى الصفحة لمدة 60 ثانية في حالة فشل في تسجيل الدخول ثلاثة مرات. يمكنك ايضا التلاعب في هذه القيم حسب الرغبة.

الآن قم باعادة تشغيل مخدم الويب Apache2 بحساب الرووت الجذر (root) من اجل اعتماد التغيرات ومن ثم حاول من جديد تنفيذ واستغلال الثغرة الأول كما يلي.

```
# /etc/init.d/apache2 restart
```

ما هي النتيجة؟ لا شك انه فشل في استغلال الثغرة ;-).

كما اسلفنا انه يفضل استخدام جميع الأربعة طرق من أجل حماية قصوى ضد هجمات Brute Force الخطرة، إلا أنه قد يؤثر سلبا على الأداء وسرعة تنفيذ العمليات بسبب كثرة الطرق. لذا تحديد الخيارات الأفضل يرجع الى فريق التصميم والبرمجة.

الثغرة الثانية: حقن الأوامر عن طريق الشل (Command Execution)

يمكن الحماية من هذه الهجوم باستخدام احدى الطريقتين (يفضل استخدامهم الاثنان لضمان حماية أقوى):

أ) عن طريق التحقق من صحة المدخلات (Input Validation). يمكنك الحصول على شفرة التحقق هذه عن طريق إعادة مستوى الأمان إلى الوضع الأقصى (high) ومن ثم اضغط على زر View Source (في الزاوية اليمنى السفلى) لمشاهدة الشفرة المستخدمة في عملية التحقيق من صحة المدخلات كما يتضح في الصورة التالية:

```php
<?php

if( isset( $_POST[ 'submit' ] ) ) {

    $target = $_REQUEST["ip"];

    $target = stripslashes( $target );


    // Split the IP into 4 octects
    $octet = explode(".", $target);

    // Check IF each octet is an integer
    if ((is_numeric($octet[0])) && (is_numeric($octet[1])) && (is_numeric($octet[2])) && (is_numeric($octet[3]))) {

    // If all 4 octets are int's put the IP back together.
    $target = $octet[0].'.'.$octet[1].'.'.$octet[2].'.'.$octet[3];


        // Determine OS and execute the ping command.
        if (stristr(php_uname('s'), 'Windows NT')) {

            $cmd = shell_exec( 'ping  ' . $target );
            echo '<pre>'.$cmd.'</pre>';

        } else {

            $cmd = shell_exec( 'ping  -c 3 ' . $target );
            echo '<pre>'.$cmd.'</pre>';

        }

    }

    else {
        echo '<pre>ERROR: You have entered an invalid IP</pre>';
    }


}

?>
```

شكل 1: شفرة التحقيق من صحة مدخلات أوامر الشل.

ب) عن طريق استخدام جدار النار (Mod_Security) لكتابة الاوامر والمعاملات المسموح بها. في مثالنا هذا نريد ان نمنع أي من علامتي **&&** أو **ls** (التي تستخدم لإلحاق أوامر اضافية لعرض محتويات المجلد الحالي) في المعاملات. يمكن تحقيق ذلك عن طريق اضافة الشرط التالي:

SecRule ARGS "(&&|ls)" "deny,status:400"

الثغرة الثالثة: استغلال ثغرة Cross Site Request Forgery (CSRF)

كما ذكرنا أن هذه الثغرة تسمح بتعديل معلومات الزبون دون علم منه. السبب في ذلك واضح وهو خطأ في برمجة النطبيق والتي تسمح بتعديل معلومات الزبون. مثلا كأن يقدم التطبيق خدمة تغيير كلمة المرور من غير أن يطلب التطبيق نفسه من المستخدم إدخال كلمة المرور الحالية. مثل هذه الحلول تمثل ثغرة خطرة يمكن استغلالها. يمكنك الإستعانة بالنصائح التالية من أجل تحقيق الحماية المنشودة:

أ) لا تسمح إطلاقا بإجراء عمليات في غاية الأهمية مثل تغيير كلمة المرور او تحويل مبلغ مالي او ما شابهه من دون ان تتأكد ان المستخدم نفسه هو من يقوم بذلك. فيمكنك مثلا عند إجراء عملية تغيير لكلمة المرور او تحويل مبلغ مالي سؤال الزبون او المستخدم عن كلمة مرورة الحالية من أجل إكمال هذه العملية وحتى ولو كان المستخدم مسجلا للدخول.

ب) هناك حلا أخر يمكن أن يكون بديلا للخيار الأول وهو استخدام مفهوم التذاكر (**Tokens**). المثال التالي يوضح ذلك حيث أن التطبيق يتكون من ملفين الاول اسمه (items.php) يقوم بإرسال الكمية إلى الملف الثاني وليكن اسمه مثلا (buy.php):

الملف الأول: items.php

```
<?php

      session_start();
      $token = md5(uniqid(rand(), TRUE);
      $_SESSION[‘token’]= $token;
      $_SESSION[‘token_time’] = time();

?>

<form action=”buy.php” method=”post”>
       <input type=”hidden” name=”token” value=”<?php echo $token; ?>”  />
       Amount: <input type=”text” name=”amount”>
      <!-- Other Fields -- >
       <input type=”submit” value=”Send”>
</form>
```

بعد إدخال الكمية والضغط على زر Send ترسل الكمية إلى الملف الثاني (buy.php) في التطبيق حيث يقوم بالتحقق من أن التذكرة (token) مطابقة أم لا، في حالة التطابق تتم عملية البيع. شفرة الملف buy.php كما يلي:

```
<?php

      if (isset($_SESSION[‘token’]) && $_POST[‘token’] = = $_SESSION[‘token’])
         {
            /* Valid Token */
         }
      else {
               /* Wrong Token */
             }
?>
```

الثغرة الرابعة: استغلال ثغرة إدراج الملف (File Inclusion)

للحماية من هذه الثغرة يجب التحقيق من المدخلات (Input Validation) ومقارنتها بقائمة تحتوي على المسموح من الملفات وهي ما يسمى بالقائمة البيضاء (White List). استخدام القائمة البيضاء (White List) هنا افضل من القائمة السوداء (Black List) حيث أنه من غير المعروف ماهي الملفات التي تكون في القائمة السوداء في حين انه نعرف ما هي الملف (او الملفات) التي سوف نسمح بها. يمكنك المقارنة بين الشفرتين (في حال كان مستوى الامان منخفض او عال) بالضغط على زر View Source. كذلك جدار النار (Mod_Security) يساعد في الحماية من هذه الهجمة كما يلي:

SecRule REQUEST_HEADERS “../” “t:urlDecode, phase:1, deny, status:400”

أيضا الإعدادات الجيدة للبيئة المستخدمة في التطبيق (مثل PHP و Apache) يساعد في توفير حماية افضل. المقال التالي سوف يتحدث عن ذلك – إن شاء الله تعالى– .

الثغرة الخامسة: استغلال ثغرة حقن أوامر قاعدة البيانات (SQL Injection)

هذه الثغرة تشكل خطورة عالية كما لاحظنا في المقال الأول إذ انها تسمح بالحصول على معلومات مثل اسماء المستخدمين وكلمات مرورهم بطريقة غير مصرح بها. وللحماية منها يجب التركيز على الحماية القصوى وذلك بأخذ النصائح التالية:

أ) التحقيق من صحة المدخلات ونظافتها (نعني بنظافتها أي أنها خالية من الإستخدام السيئ الغير مخصص لها) وذلك بتمريرها الى الدالة **mysql_real_escape_string()** كما يلي:

```php
<?php

if(isset($_GET['Submit'])){

    // Retrieve data

    $id = $_GET['id'];
    $id = stripslashes($id);
    $id = mysql_real_escape_string($id);

    if (is_numeric($id)){

    $getid="SELECT first_name, last_name FROM users WHERE user_id = '$id'";
    $result=mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>' );

    $num=mysql_numrows($result);

    $i=0;

    while ($i < $num) {

        $first=mysql_result($result,$i,"first_name");
        $last=mysql_result($result,$i,"last_name");

        echo '<pre>';
        echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
        echo '</pre>';

        $i++;
    }
    }
}
?>
```

شكل 2: شفرة التحقق من صحة مدخلات أوامر MySQL لتجنب الحقن.

ب) عن طريق استخدام جدار النار **GreenSQL** المخصص لقواعد البيانات والذي يعتبر ذو فعالية عالية في الحماية من ثغرات **SQL Injection**. يمكنك تنزيله من (هنا) وطريقة تركيبه (هنا) واستخدامه (هنا).

ج) عن طريق اضافة شروط (**Rules**) لجدار النار **Mod_Security** كما يلي:

SecRule ARGS "(select|union|union\s+|drop\s+|into\s+outfile)" "t:lowercase,deny,status:400"

الثغرة السادسة: استغلال ثغرة رفع الملفات (File Uplodaing)

للحماية من هذه الثغرة يجب التحقيق من المدخلات (Input Validation) لامتدادات الملفات المرفوعة للتأكد من صلاحيتها هل هو مسموح بهذا الامتداد ام لا. أيضا يجب التأكد من صحة مسار المجلد الذي سوف ترفع فيه الملفات. بالشفرة الموجودة في View Source تحقق ذلك كما يظهر في الصورة التالية:

```php
<?php
if (isset($_POST['Upload'])) {

            $target_path = DVWA_WEB_PAGE_TO_ROOT."hackable/uploads/";
            $target_path = $target_path . basename($_FILES['uploaded']['name']);
            $uploaded_name = $_FILES['uploaded']['name'];
            $uploaded_ext = substr($uploaded_name, strrpos($uploaded_name, '.') + 1);
            $uploaded_size = $_FILES['uploaded']['size'];

            if (($uploaded_ext == "jpg" || $uploaded_ext == "JPG" || $uploaded_ext == "jpeg" || $uploaded_ext == "JPEG") && ($uploaded_size < 100000)){


                if(!move_uploaded_file($_FILES['uploaded']['tmp_name'], $target_path)) {

                    echo '<pre>';
                    echo 'Your image was not uploaded.';
                    echo '</pre>';

                  } else {

                    echo '<pre>';
                    echo $uploaded_name . ' succesfully uploaded!';
                    echo '</pre>';

                    }
            }

            else{

                echo '<pre>';
                echo 'Your image was not uploaded.';
                echo '</pre>';

            }
        }

?>
```

شكل 3: شفرة التحقق من صحة المدخلات المسموح بها لرفع الملفات.

الثغرتان السابعة والثامنة: استغلال ثغرة (XSS) بنوعيها Reflected و Stored

كما لاحظنا خطورة هذه الثغرات في المقال السابق إذ انها تحتل الرقم واحد طبقا لتصنيف SANS للثغرات الأكثر خطورة. يمكن الحماية منهما بواحدة من الطريقتين (يفضل استخدام الأثنان معا لأجل توفير حماية أفضل):

أ) تمرير المخرجات قبل عرضها إلى إحدى الدالتين التاليتين:

1) htmlspecialchars($varName, ENT_QUOTES)
2) htmlentities ($varName, ENT_QUOTES)

بالنظر إلى شفرتي Reflected XSS و Stored XSS بالضغط على زر View Source لكل منهما نلاحظ كيف يمكن الحماية من هذه الهجمات كما يلي:

```php
<?php
    if($_GET['name'] == NULL || $_GET['name'] == ''){

        $isempty = true;

    } else{

        echo '<pre>';
        echo 'Hello ' . htmlspecialchars($_GET['name']);
        echo '</pre>';

        }
?>
```

شكل 4: شفرة الحماية من ثغرة Reflected XSS.

```php
<?php

if(isset($_POST['btnSign']))
{

   $message = trim($_POST['mtxMessage']);
   $name    = trim($_POST['txtName']);

   // Sanitize message input
   $message = stripslashes($message);
   $message = mysql_real_escape_string($message);
   $message = htmlspecialchars($message);

   // Sanitize name input
   $name = stripslashes($name);
   $name = mysql_real_escape_string($name);
   $name = htmlspecialchars($name);

   $query = "INSERT INTO guestbook (comment,name) VALUES ('$message','$name');";

   $result = mysql_query($query) or die('<pre>' . mysql_error() . '</pre>' );

}

?>
```

شكل 5: شفرة الحماية من ثغرة Stored XSS.

ب) عن طريق استخدام جدار النار **(Mod_Security)** كما يلي:

SecRule ARGS "<script|<script\s+|<iframe|iframe\s+" "t:lowercase,deny, status:400"

المقال القادم:

شرح الإعدادات الآمنة والصحيحة لمنصة تطبيقات لغة البرمجة PHP والمكونة من PHP و Apache و MySQL. كذلك سوف نتاول بإذن الله كيفية مراقبة ملقم الويب Apache للتعرف على الهجمات المحتملة من خلال مراقبة ملفات تسجيل العمليات (Logs) وإجراء عملية التحقيق في حالة الإختراق للتعرف على المجرمين.

ولنا تكملة إن شاء الله ;-) >>