

# HTTP 请求和爆破

---

## HTTP 请求和爆破

### 前言

#### 一、HTTP 请求的方法和基本格式

##### 1. HTTP 请求的概念

##### 2. HTTP 请求的格式

###### 2.1 请求行

###### 2.2 请求头

###### 2.3 请求体

##### 3. HTTP请求的方法

###### 3.1 GET 方法

###### 3.2 POST 方法

##### 4. HTTP响应格式

###### 4.1 HTTP响应状态行

#### 二、进行 HTTP 请求的工具

##### 1. 浏览器

##### 2. Python 的 requests 库

###### 2.1 安装库

###### 2.2 使用文档

###### 2.3 使用示例

##### 3. Hackbar

###### 3.1 安装插件

###### 3.2 使用示例

##### 4. Postman

###### 4.1 安装 Postman

###### 4.2 使用示例

#### 三、抓包的工具

##### 1. 使用浏览器进行简单的抓包

##### 2. BurpSuite 介绍

##### 3. BurpSuite 专业版和 Proxy SwitchyOmega安装

##### 4. 基本抓包和发包的使用

#### 四、目录爆破

##### 1. 目录爆破简介

##### 2. dirsearch 安装

##### 3. dirsearch 的使用

##### 4. dirsearch 的使用示例

#### 五、口令爆破

##### 1. 明文验证爆破

##### 2. HTTP 认证爆破

##### 3. Hash 爆破

### 💡 Tip

温馨提示，本文可能有点长，可以根据自身需要点击目录中的链接快速跳转。

“二、进行 HTTP 请求的工具”中除浏览器外任意一个工具的使用示例指引，对解决“成为 HTTP 大师”这一题有帮助。“四、目录爆破”和“五、口令爆破”对解决“崩坏：重返物华之尘白碧蓝原神方舟档案之眼”这一题有帮助。

# 前言

本文章将会向你介绍 **HTTP 请求** 以及 **目录爆破** 和 **口令爆破** 的相关知识。通过本文，你将掌握：

- HTTP 请求和响应的基本格式
- 使用 Python 的 requests 等工具发送 HTTP 请求
- 使用 浏览器的开发人员工具 和 BurpSuite 进行抓包和发包
- 使用 dirsearch 进行目录爆破
- 使用 BurpSuite 进行口令爆破，编写 Python 脚本进行口令爆破

## 一、HTTP 请求的方法和基本格式

### 1. HTTP 请求的概念

首先，作为一个教程文档，肯定是要从某个地方粘贴一段简介过来的。

**HTTP** 是一种用作获取诸如 HTML 文档这类资源的协议。它是 **Web 上进行任何数据交换的基础**，同时，也是一种客户端—服务器（client-server）协议，也就是说，请求是由接受方——通常是 Web 浏览器——发起的。完整网页文档通常由文本、布局描述、图片、视频、脚本等资源构成。

客户端与服务端之间通过交换一个个独立的消息（而非数据流）进行通信。由客户端发出的消息被称作请求（request），由服务端发出的应答消息被称作响应（response）。

关于 HTTP 请求的详尽文档，可以参考：[HTTP | MDN](#)

### 2. HTTP 请求的格式

HTTP 请求由请求行、请求头和请求体三部分构成，各部分由 CRLF( `\r\n` ) 分割，其中请求头和请求体之间需要有一个多余的空行。

	描述	示例
请求行	方法 + URL + 协议版本	<code>GET /index.html HTTP/1.1</code>
请求头	附加信息（键值对）	<code>Content-Type: application/x-www-form-urlencoded</code>
请求体	传输数据（可选）	<code>username=test&amp;password=123</code>

例如：

```
POST /example?user=1234 HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/91.0.4472.124 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Content-Length: 9
Cookie: 10v3=ctf;
X-Forwarded-For: 114.51.4.19

pond=5678
```

接下来将逐一讲解每一个部分的格式。

## 2.1 请求行

请求行包含请求方法、请求的目标和 HTTP 版本，用空格进行分割。例如：

```
GET /index.html HTTP/1.1
```

意思是：使用 GET 请求方法，请求服务器上的 `/index.html` 地址，使用 HTTP 的 1.1 版本。

一个完整的 URL 一般由如下几个部分组成：

```
协议://主机名:端口号/路径?查询参数#片段标识符
```

比如：`https://blog.yemaster.cn/post/164`

在 HTTP 请求行中，一般可以省略具体的协议、主机名和端口号，只保留 `/路径?查询参数#片段标识符` 即可，因为主机名等信息在建立连接时就已经确定，另外 Headers 中的 Host 字段也会明确指定请求的主机名和端口号。

当前主流的版本有 HTTP/1.1（广泛兼容）、HTTP/2.0（性能优化）、HTTP/3.0（基于 UDC 的 QUIC 协议），具体差异可以参考 [HTTP 的发展 - HTTP | MDN](#)。

关于请求方法的介绍请参见第 3 节 HTTP 请求的方法。

## 2.2 请求头

请求头用于提供关于请求的附加信息，由一系列格式为 `Header: value` 的键值对构成，每个键值对独占一行。下面介绍一些常用的请求头，对于所有的请求头的含义，可以参阅 [HTTP 标头 - HTTP | MDN](#)。

字段名	作用	示例
Host	指定目标主机和端口（必填）	Host: blog.yemaster.cn:80
User-Agent	用于标识客户端的身份，例如 浏览器、操作系统、设备类型等	User-Agent: Mozilla/5.0 (windows NT 10.0; win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36
Content-Type	用于定义请求体的数据格式，具体遵循 MIME 规范，即 <code>type/subtype</code> 的格式	Content-Type: application/json
Content-Length	声明请求体字节长度	Content-Length: 256
Accept	用于告知可接受的响应数据类型。	Accept: text/html, application/json
Cookie	Cookie 用于在客户端和服务端之间保持会话状态。包含一个或多个键值对，多个键值对之间用分号和空格分隔。	Cookie: session=abcde; username=yema

字段名	作用	示例
X-Forwarded-For	记录客户端 IP（代理场景）	X-Forwarded-For: 114.51.4.19

### 2.3 请求体

请求体是 HTTP 请求中可选的部分。请求体中包含实际要传输的数据，其格式由请求头中的 `Content-Type` 字段指定。

详见第 3.2 节 POST 方法的参数传递方式。

## 3. HTTP请求的方法

HTTP 协议定义了多种请求方法，包括 GET、POST、PUT、DELETE 等，每种方法都有对应的用途。

方法	用途	参数位置	场景示例
GET	获取资源	URL 查询字符串	网页访问、API 查询
POST	提交数据	请求体	表单提交、文件上传
PUT	更新资源	请求体	修改用户资料
DELETE	删除资源	URL 或请求体	删除文件、记录

其中，最常用也最基础的是 **GET** 和 **POST** 方法。

### 3.1 GET 方法

GET 方法用于从服务器获取资源，是最常用的请求方法之一。例如，当你在浏览器地址栏输入一个网页地址并回车，浏览器就会向服务器发送一个 GET 请求来获取该网页的内容。

#### 参数传递方式


GET 请求的参数通过 **查询字符串**（Query String）传递：

```
GET /search?keyword=yema&version=1 HTTP/1.1
```

参数部分以一个问号 `?` 开始，每个参数以 `参数名=参数值` 的形式出现，多个参数用 `&` 连接。上述请求中传递了两个参数：`keyword=yema` 和 `version=1`。

如果参数中包含特殊字符（如 `(空格)`、`&`、`=`、`#` 等），则必须对他们进行 **URL 编码**，否则可能引起解析错误。

GET 请求**一般**不需要请求体。

 **Tip**

- 1. GET 参数直接暴露在 URL 中，因此 GET 不适合传输敏感信息。
- 2. URL 的长度有最大限制，所以 GET 请求也不适合传输大量数据。



## 3.2 POST 方法

POST 方法用于向服务器提交数据，通常用于表单提交、文件上传、用户登录等场景。

### 参数传递方式

与 GET 不同，POST 请求的参数不出现在 URL 中，而是封装在 **请求体 (Request Body)** 中进行传输。

POST 请求的参数格式有多种，需要指定 Content-Type 向接收端说明参数，这里介绍几种最常用的：

- `application/x-www-form-urlencoded`

描述：这是 HTML 表单提交数据时默认采用的编码方式。格式类似 Query String。

示例请求：

```
POST /login HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 25

username=testuser&password=123456
```

- `multipart/form-data`

描述：用于处理包含文件上传或者二进制数据的表单提交。请求体被分割成多个部分，每个部分都有自己独立的 `Content-Type` 和 `Content-Disposition` 头信息，各部分之间使用指定的边界（boundary）分隔。具体可以看下面的示例：

示例请求：

```
POST /upload HTTP/1.1
Host: example.com
Content-Type: multipart/form-data; boundary=-----iamboundary
Content-Length: 312

-----iamboundary
Content-Disposition: form-data; name="username"

testuser
-----iamboundary
Content-Disposition: form-data; name="avatar"; filename="avatar.jpg"
Content-Type: image/jpeg

<二进制图片数据>
-----iamboundary--
```

另外，常见的还有 `application/json`、`text/plain`、`application/xml` 分别对应的就是 json 数据、纯文本数据和 xml 数据。

## 4. HTTP响应格式

HTTP 响应仍然由 3 部分组成：状态行、响应头和响应体，由 CRLF(`\r\n`) 分割，响应头和响应体之间仍需有一个空行。

	作用说明	示例格式
状态行	结果状态，协议版本+状态码+原因短语	<code>HTTP/1.1 200 OK</code>
响应头	传递附加信息（缓存、编码等）	<code>Content-Type: text/html; charset=utf-8</code>
响应体	存放实际数据（网页代码、文件内容）	实际数据内容（如 HTML 文本或二进制数据）

### 4.1 HTTP响应状态行

状态行由 协议版本+状态码+原因短语 构成，中间用空格分隔，例如：

```
HTTP/1.1 200 OK
```

#### 状态码和原因短语

状态码指示 HTTP 请求的响应状态，原因短语则是对状态码含义的简单描述。状态码按照首位数字可以分为 5 大类：

状态码	分类
1xx	临时相应
2xx	成功
3xx	重定向
4xx	客户端错误
5xx	服务器错误

接下来介绍几种常见的状态码：

状态码	原因短语	说明
200	OK	网页 / API 请求正常返回数据
301	Moved Permanently	永久重定向，请求的资源已被永久移动到新的 URL，搜索引擎会更新索引
400	Bad Request	客户端发送的请求有语法错误，服务器无法理解，比如请求参数格式不正确等
403	Forbidden	服务器理解客户端的请求，但拒绝执行该请求，通常是因为客户端没有足够的权限访问资源
404	Not Found	服务器无法找到请求的资源，可能是 URL 输入错误或资源已被删除等原因

状态码	原因短语	说明
500	Internal Server Error	服务器内部发生错误，无法完成请求，通常是服务器端代码出现问题等导致

## 二、进行 HTTP 请求的工具

学习了第一部分的内容之后，想必你已经掌握 HTTP 请求的基本知识了。但是，如果手写 HTTP 请求体来发送请求并处理响应，不仅过程繁琐复杂，而且容易出错。在实际开发和测试中，这样的操作效率也非常低下。下面将介绍几种常用的 HTTP 请求的工具，帮助我们更高效、便捷地进行 HTTP 请求相关的操作。

### 1. 浏览器

没错，浏览器是我们身边最常见的 HTTP 请求工具。当我们在地址栏中输入一个网址并按下回车键时，浏览器就会自动向服务器发送一个 HTTP GET 请求。浏览器会处理服务器返回的响应，将网页内容进行解析、渲染并展示给我们。

浏览器也具备一定的调试能力，但是相比于其他工具来说功能较少。因此，更推荐选择下面的几种 HTTP 发包工具来进行发包。

### 2. Python 的 requests 库

`requests` 库是 Python 中一个非常流行且实用的 HTTP 库，它提供了简洁而强大的 API，让你可以用 Python 代码轻松发送 HTTP 请求。

#### 2.1 安装库

既然是 Python 的库，肯定要装 Python 环境的，这个你应该在上节课已经配置好了。

你可以使用 `pip` 直接安装 `requests` 库：

```
pip install requests
```

#### 2.2 使用文档

使用 `import requests` 库之后可以将其引入到代码中，接着就可以用库里的函数了。

`requests` 提供了 `get` 和 `post` 函数来进行 GET 和 POST 请求：

函数	描述	示例
<code>requests.get(url, params=None, **kwargs)</code>	发送一个 HTTP GET 请求到指定的 URL。 <code>params</code> 是一个字典，用于传递查询参数， <code>**kwargs</code> 可以包含其他可选参数，如 <code>headers</code> 、 <code>cookies</code> 等。	<pre>response = requests.get('https://example.com')</pre>

函数	描述	示例
<code>requests.post(url, data=None, json=None, **kwargs)</code>	发送一个 HTTP POST 请求到指定的 URL。 <code>data</code> 用于传递表单数据， <code>json</code> 用于传递 JSON 格式的数据， <code>**kwargs</code> 同样可包含其他可选参数。	<pre>response = requests.post('https://example.com/api', data={'key': 'value'})</pre>

你可以进一步指定的函数的参数，来设置请求的 请求头，Cookies 等信息：

参数	描述	适用函数	示例
<code>data</code>	用于传递表单数据，通常是字典、元组列表、字节或文件对象。适用于需要以表单形式提交数据的场景。	<code>post</code>	<pre>requests.post('url', data={'name': 'John', 'age': 30})</pre>
<code>json</code>	用于传递 JSON 格式的数据，会自动设置请求头 <code>Content-Type</code> 为 <code>application/json</code> 。适用于向 API 发送 JSON 数据。	<code>post</code>	<pre>requests.post('url', json={'name': 'John', 'age': 30})</pre>
<code>headers</code>	用于设置请求头，以字典形式传递。可以用来模拟浏览器请求、设置认证信息等。	<code>get</code> 、 <code>post</code> 等	<pre>headers = {'User-Agent': 'Mozilla/5.0'}; requests.get('url', headers=headers)</pre>
<code>cookies</code>	用于设置请求的 cookie，以字典或 <code>CookieJar</code> 对象形式传递。可以用来维持会话状态。	<code>get</code> 、 <code>post</code> 等	<pre>cookies = {'session_id': '12345'}; requests.get('url', cookies=cookies)</pre>

进行请求之后，会返回一个 `response` 对象，包含了 HTTP 请求响应的一些信息：

属性 / 方法	描述	示例
<code>status_code</code>	返回 HTTP 响应的状态码，如 200 表示成功，404 表示未找到。	<pre>print(response.status_code)</pre>
<code>text</code>	返回响应内容的文本形式，通常用于处理 HTML、JSON 等文本数据。	<pre>print(response.text)</pre>
<code>json()</code>	尝试将响应内容解析为 JSON 格式，如果响应是有效的 JSON 数据，会返回一个 Python 字典或列表。	<pre>data = response.json()</pre>
<code>headers</code>	返回响应的头信息，以字典形式呈现。	<pre>print(response.headers)</pre>
<code>cookies</code>	返回响应中包含的 cookie 信息。	<pre>print(response.cookies)</pre>

## 2.3 使用示例

接下来将通过多个示例来帮助你更好的掌握 requests 库的用法。

### 进行 GET 请求并设置 Query String 参数

```
import requests

# 定义请求的 URL
url = "http://example.com"
# GET 请求的参数
params = {
    "yema": 1234
}

response = requests.get(url, params=params)
print("请求响应内容如下: ")
print(response.text)
```

### 进行 POST 请求，并设置 POST 参数

```
import requests

url = "http://example.com"
params = {
    "yema": 1234
}
# 定义 POST 请求数据
data = {
    "pond": 5678
}

response = requests.post(url, params=params, data=data)
print("请求响应内容如下: ")
print(response.text)
print("响应头如下: ")
print(response.headers)
```

### 设置请求的 Cookies 和 Headers

```
import requests

url = "http://example.com"
# 定义 Headers
headers = {
    "X-Forwarded-For": "114.51.4.191"
}
# 定义 Cookies
cookies = {
    "10v3": "ctf"
}
```

```
response = requests.post(url, headers=headers, cookies=cookies)
print("请求响应内容如下：")
print(response.text)
```

## 3. Hackbar

Hackbar 是一个功能强大的浏览器插件，其中包含了自定义 HTTP 请求发送的功能。

### 3.1 安装插件

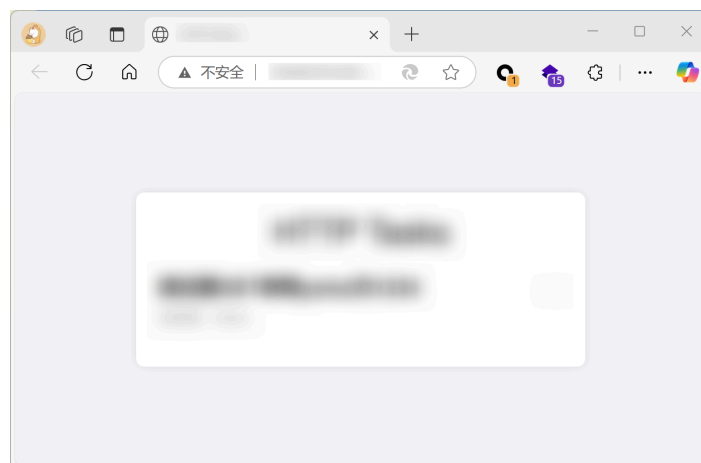
Hackbar 是一个浏览器插件，可以在浏览器的插件市场中搜索并安装：

- Edge / Chrome插件：[HackBar - Chrome 線上應用程式商店](#)
- Firefox插件：[HackBar - 下载 🦊 Firefox 扩展 \(zh-CN\)](#)

你也可以访问 Github 仓库 [0140454/hackbar: A browser extension for Penetration Testing](#)，从 Releases 中下载对应的 zip 包并手动安装插件（[Edge如何装插件？Edge添加插件的方法 - 知乎](#)）。

### 3.2 使用示例

首先，我们在浏览器随便打开一个网页（网址和网页内容涉及到敏感信息，故打码处理，下同）：

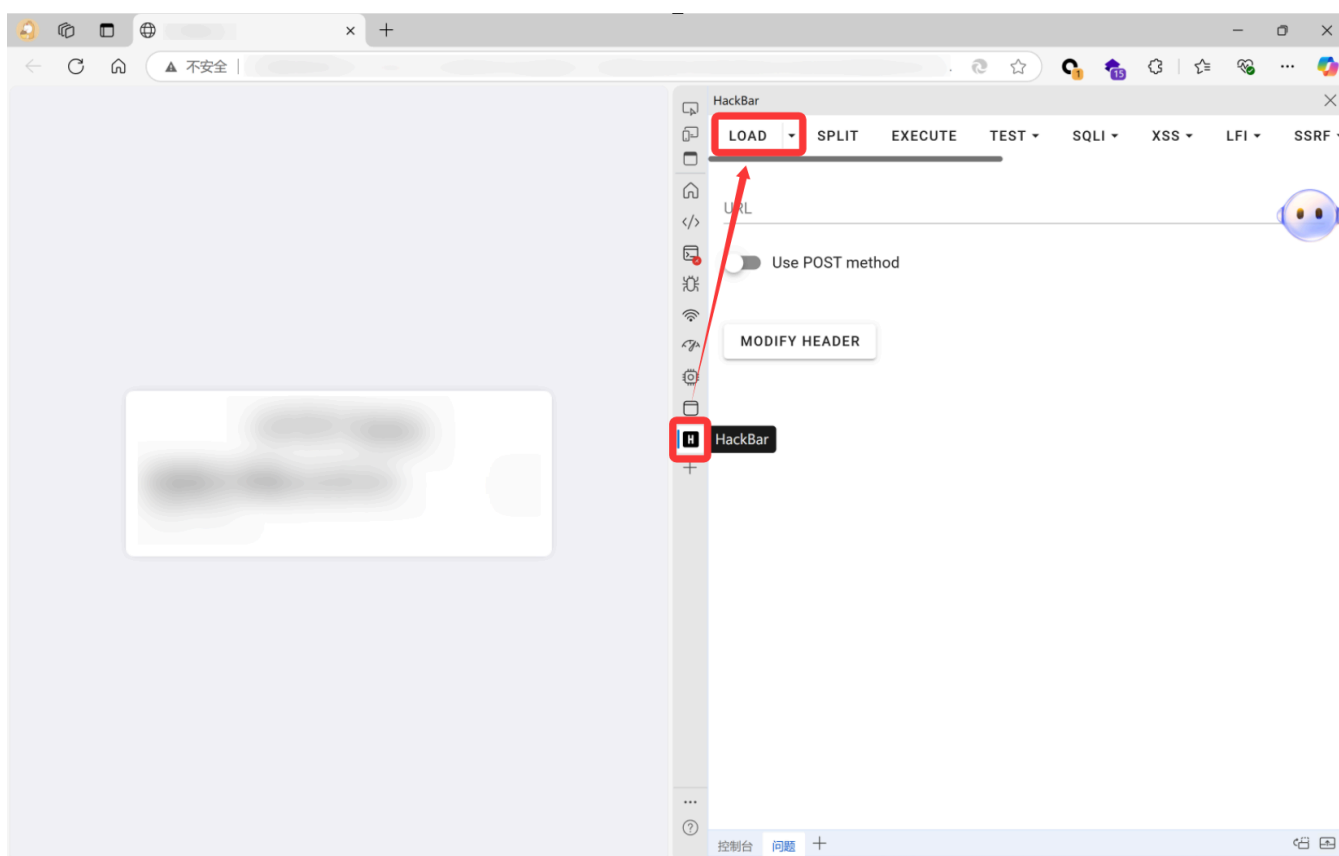


然后我们按下 F12 打开开发者工具，或者按下图所示方式手动打开：



然后找到 Hackbar 这一栏，如果找不到，可以点击旁边的加号（更多工具）按钮然后在里面找。

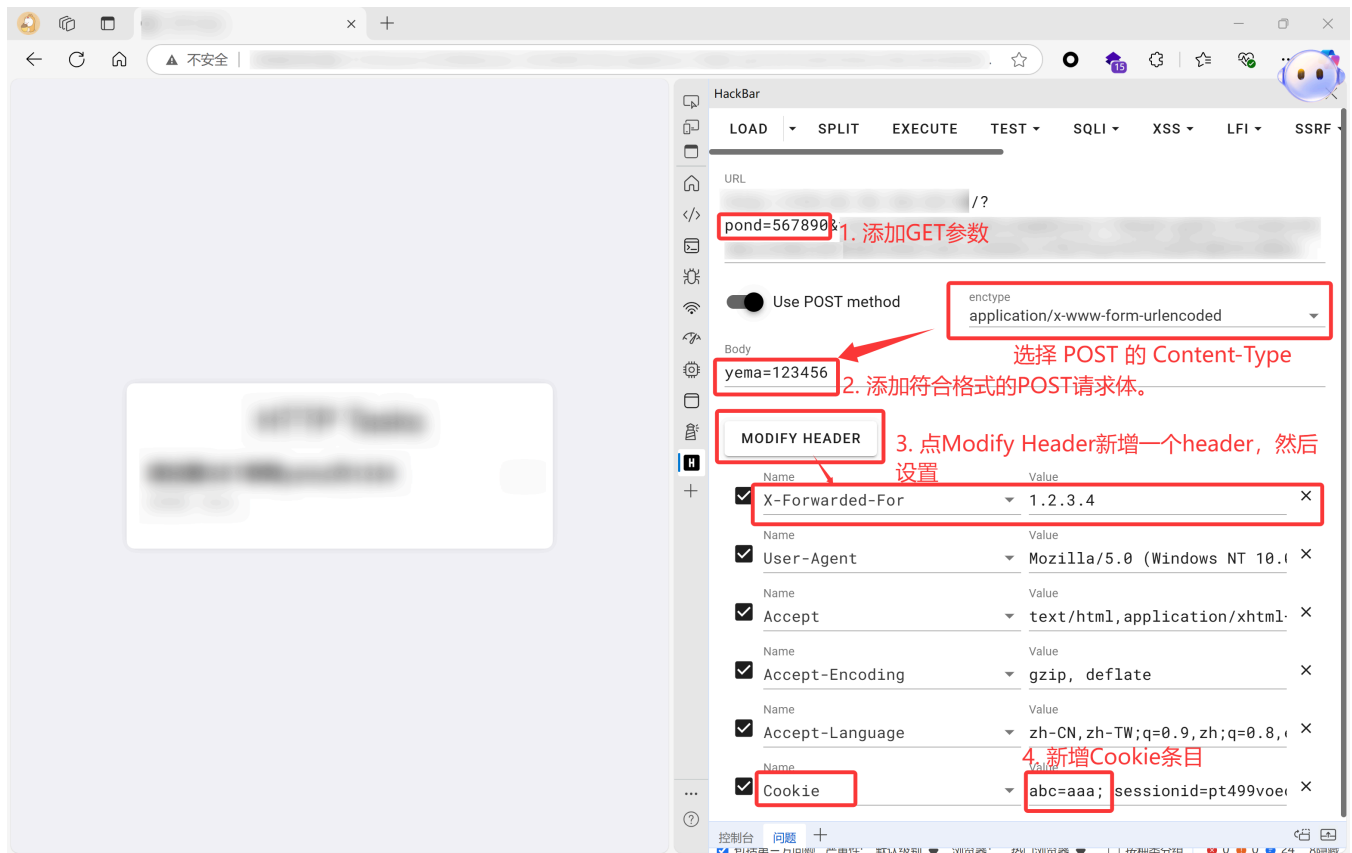
点开 Hackbar 之后，点击 LOAD 按钮就可以将当前网页的请求信息加载出来，包括请求的方式（是否为 POST），URL，和 Headers。



GET 的参数可以在 URL 中修改。开启 Use POST method 之后，可以选择请求的参数格式（Content-Type）和请求体（Body）。

点击 Modify Header 按钮可以新增 Header 条目。

比如，下图演示了添加 GET 参数 `pond=123456`，POST 参数 `yema=567890`，并修改 Cookie `abc=aaa`，添加 Header 头：`X-Forwarded-For: 1.2.3.4`



## 4. Postman

Postman 是一个优秀的图形化 HTTP 测试工具。下面我们将介绍使用 Postman 来发送和接受 HTTP 请求：

### 4.1 安装 Postman

访问 Postman 官网即可下载：[Download Postman | Get Started for Free](#)

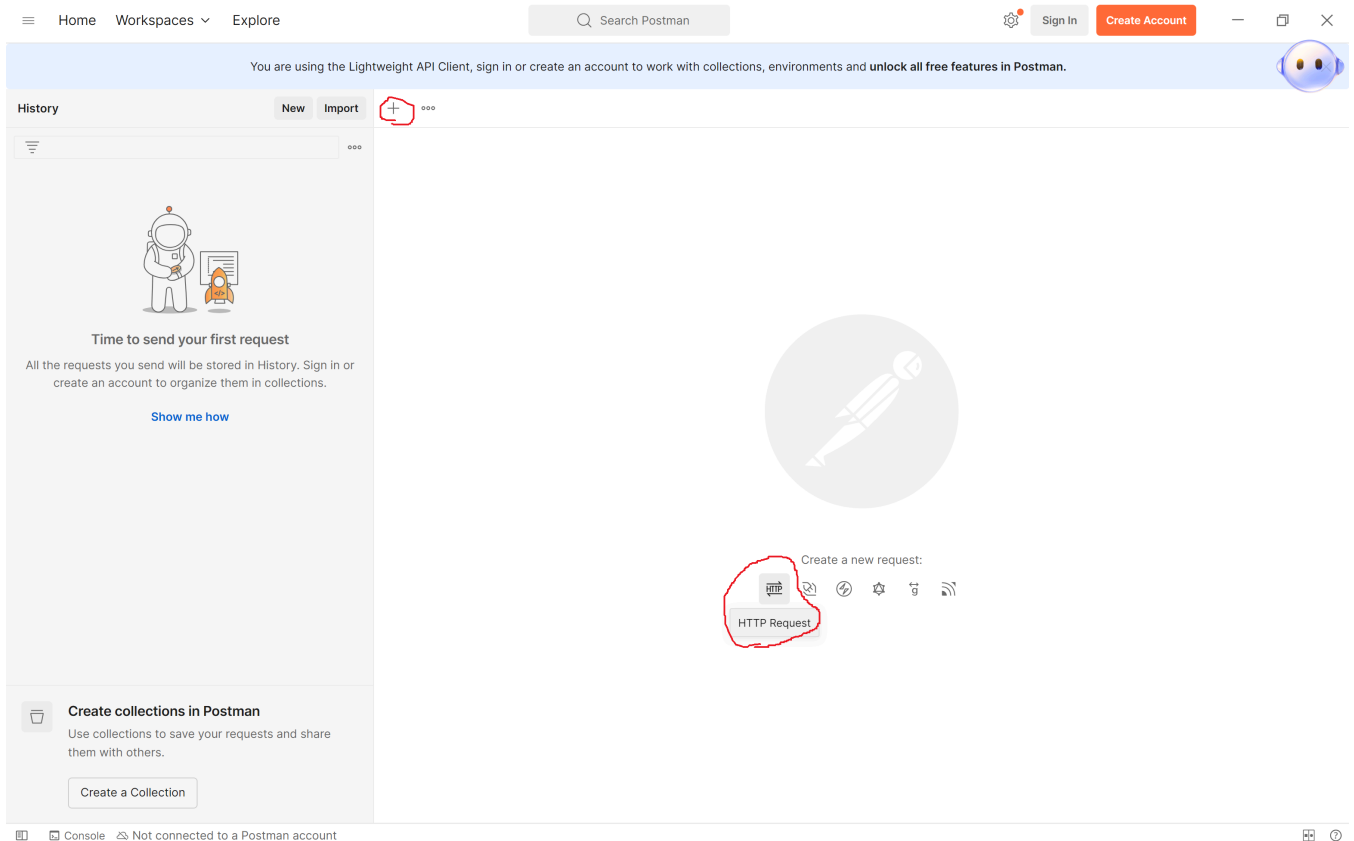
当然 Postman 也提供了在线版本，注册后即可使用。

### 4.2 使用示例

打开 Postman。初次使用可能会提示登录，但是可以跳过登录这一步使用 Lightweight API Client。

然后可以点击 + 来新建一个 HTTP 请求，也可以点击 HTTP 按钮来新建一个 HTTP 请求。

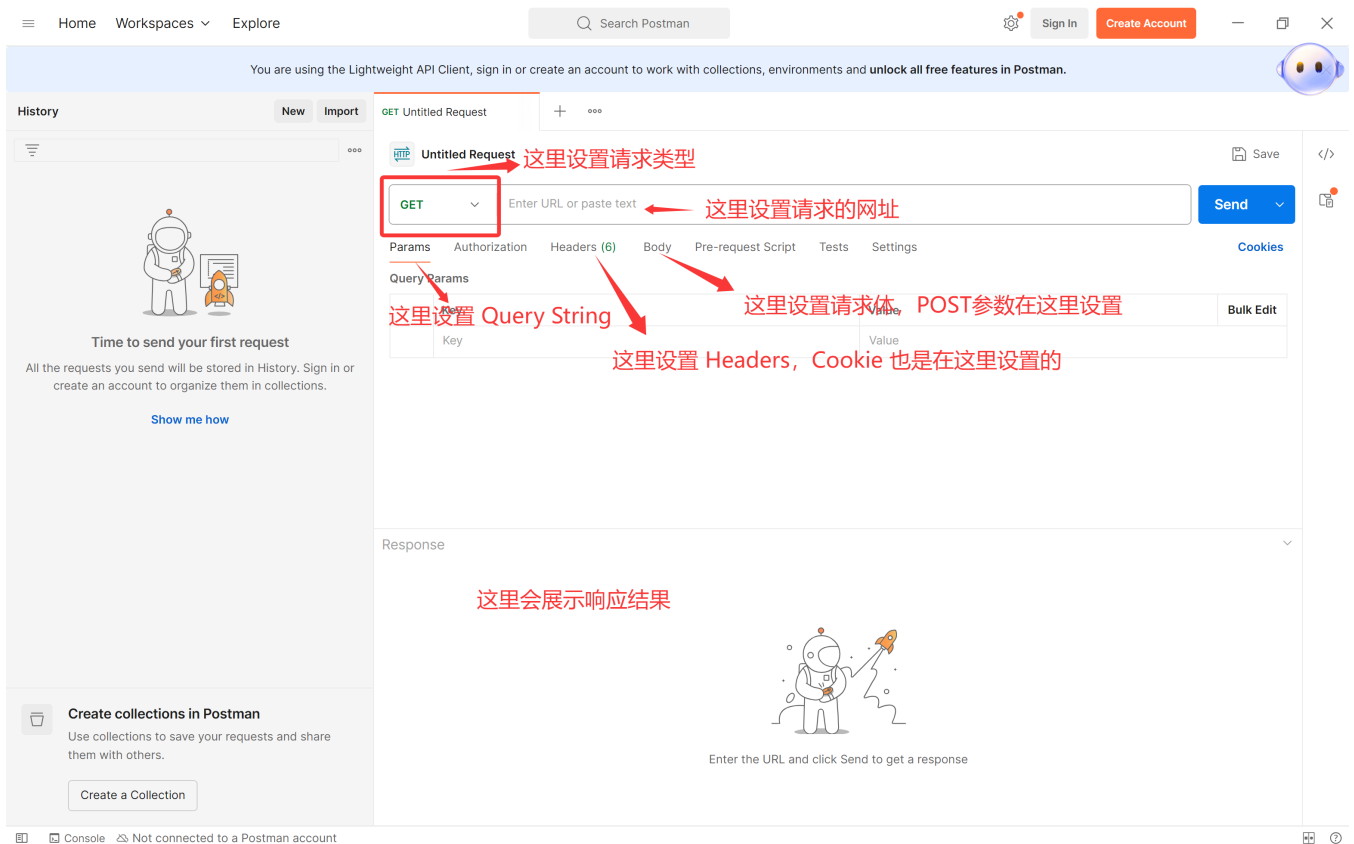




然后，就到了请求的配置界面，下面讲解基本参数的配置说明：

首先是 请求类型的选择 和 请求网址的设置。

下一行的 Tab 里，Params 是 Get 参数的设置，Authorization 是 HTTP 认证设置，Headers 是请求头设置，Body 是请求体设置。



全部配置完之后，点击 Send 就可以发送请求了。

例如，下面我们对 <http://example.com> 发送一个 POST 请求，Query 参数为 yema=111，POST 参数为 pond=666，设置 Cookie: abc=ddd，并设置 X-Forwarded-For 为 1.2.3.4：

GET

http://example.com

填写请求地址

Send

GET

POST

PUT

PATCH

DELETE

HEAD

OPTIONS

Type a new method

选择POST请求

Headers (6)

Body

Pre-request Script

Tests

Settings

Cookies

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

Key	Value	Bulk Edit
<input checked="" type="checkbox"/> yema	111	
Key	Value	

设置 Query参数 yema=111

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

Cookies

☒ Host

☒ Cookie

☒ X-Forwarded-For

Key

<calculated when request is sent>

abc=ddd

1.2.3.4

Value

设置 Cookie 和其他请求头

Params

Authorization

Headers (10)

Body

Pre-request Script

Tests

Settings

Cookies

none

form-data

☒ x-www-form-urlencoded

raw

binary

1. 先选择 Content-Type

Key	Value	Bulk Edit
<input checked="" type="checkbox"/> pond	666	
Key	Value	

2. 设置 POST 参数

然后点击发送即可看到请求结果：

Body

Cookies (1)

Headers (7)

Test Results

Body是响应体，Cookie是服务端设置的Cookie

Save Response

Pretty

Raw

Preview

Visualize

HTML

Headers是响应头

1

2

3

4

5

6

7

8

9

10

11

12

```
!DOCTYPE html
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>HTTP Tasks</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
    }
  </style>
</head>
<body>
</body>
</html>
```

响应体输出中，Pretty是美化的HTML代码，Raw是原始响应体  
Preview是解析HTML代码的结果

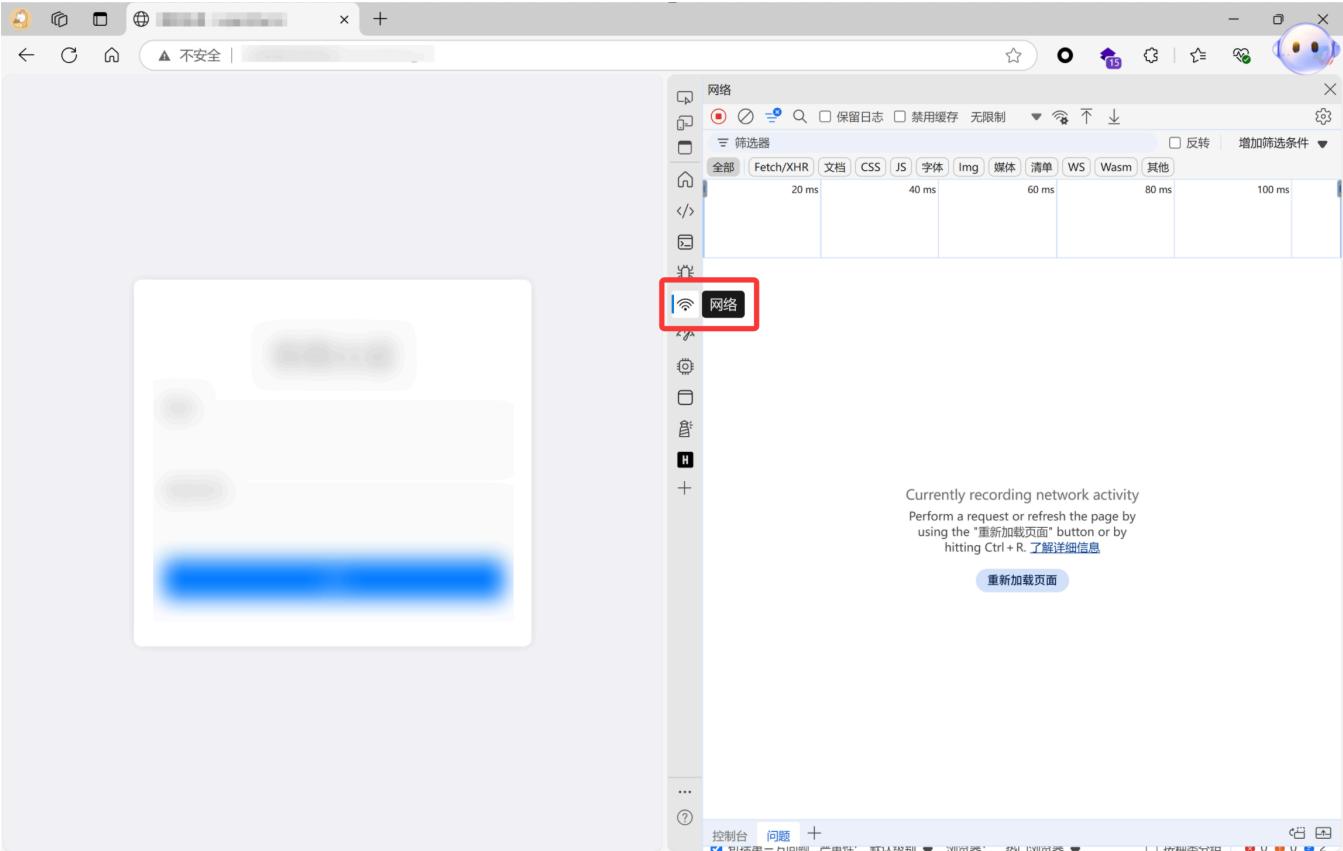
另外，JavaScript 中的 [fetch](#) 函数，命令行的 [curl](#) 工具也都是不错的工具，有兴趣的同学可以在网上搜索相关资料学习相关用法。

### 三、抓包的工具

#### 1. 使用浏览器进行简单的抓包

浏览器的开发者工具具有简单的抓包能力，下面将进行简单介绍：

首先打开需要抓包的页面，打开“开发人员工具”，选择“网络（Network）”选项卡：



然后根据提示按下 “Ctrl + R” 或者点击 “重新加载页面” 按钮进行抓包，右侧列表页面就获取了当前页面发送的全部的 HTTP 请求包的内容，包括 名称、状态码 等。

我们在左侧页面中随便输入一些信息，然后点击按钮，列表中新出现的项就是点击按钮后新发的包。

网络

筛选器

全部 Fetch/XHR 文档 CSS JS 字体 Img 媒体 清单 WS Wasm 其他

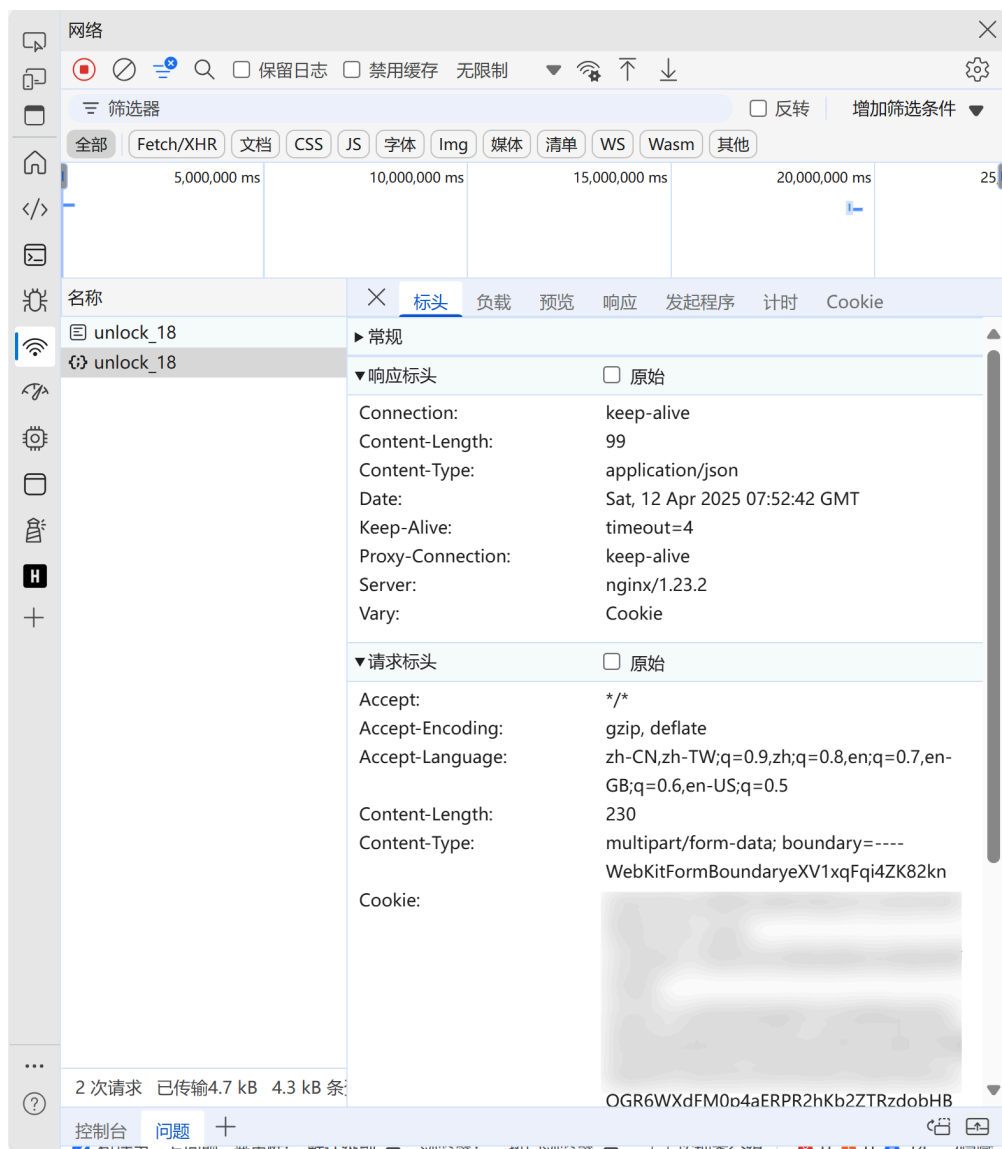
名称	状态	类型	发起程序	大小	时间	履行者
unlock_18	200	docum...	其他	4.4 kB	10 ms	
unlock_18	200	fetch	unlock_18:132	320 B	19 ms	

2 次请求 已传输 4.7 kB 4.3 kB 条资源 完成: 5.4 小时 DOMContentLoaded: 23 ms 加载: 46 ms

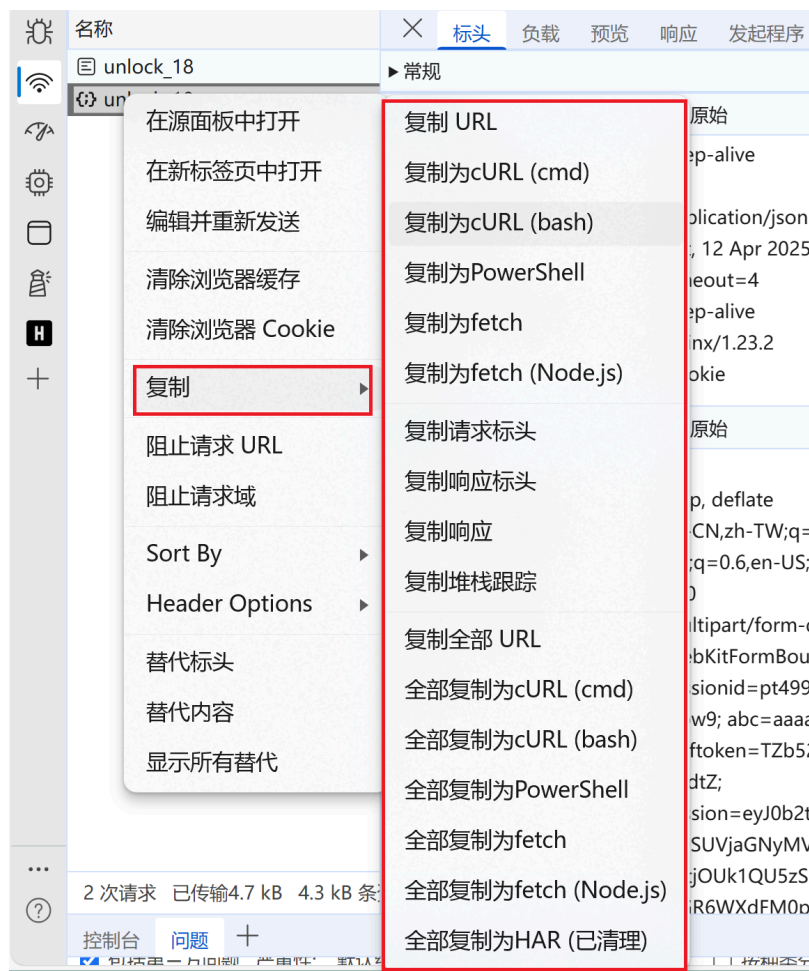
控制台 问题 +

点击按钮后，页面新发送了一个请求  
右边出现了被记录下来的请求

我们点击这一行新出的请求，可以查看请求的详细信息：



右键点击对应的请求，可以将对应的请求转化为代码（这里可以选 cURL、PowerShell、fetch 代码），也可以编辑并重新发送（比较鸡肋，建议还是用第 2 节介绍的 BurpSuite 抓包发包）



这里介绍一个将 cURL 命令转化为 Python 代码的方法，方便我们写脚本。我们首先点击“复制为 cURL(bash)”，把请求复制为 cURL 代码。之后我们在搜索引擎上搜索“cURL转Python”，可以找到进行在线的网站：[在线curl命令转代码](#)，将我们的代码复制进去：



这样子，我们就可以把浏览器的请求转化为对应的 Python 脚本，方便我们进一步操作。

## 2. BurpSuite 介绍

这个没什么好说的，从网上复制一段介绍过来就行。

BurpSuite 是一款广泛应用于 Web 应用程序安全测试的集成化平台，由 PortSwigger 公司开发。它集多种功能于一体，为安全测试人员和渗透测试人员提供了强大的工具，用于发现、分析和利用 Web 应用程序中的安全漏洞。该工具既适合初学者快速上手 Web 安全测试，也能满足专业安全人员在复杂场景下的深度测试需求。

这里用到的主要是 BurpSuite 的 Proxy、Intruder 和 Repeater 功能。

### 3. BurpSuite 专业版和 Proxy SwitchyOmega 安装

BurpSuite 官方的下载地址是：[Professional / Community 2025.2.4 | Releases](#)，但是由于专业版使用要钱，所以这里找到了可以使用的学习版本供大家学习使用：[【抓包神器】BurpSuite Professional v2025.2 中文汉化版 - 极核GetShell](#)

按照链接提示，可以下载汉化过的，直接使用的 BurpSuite 专业版，请照着文章中的教程进行破解配置。仅供学习使用！！

因为 BurpSuite 的 Proxy 功能会将使用 `http://localhost:8080` 的请求给拦截下来，所以我们需要配置浏览器的代理。Proxy SwitchyOmega 是一个浏览器插件，能够方便的修改浏览器的代理配置。

- Edge 安装地址（非官方，慎用，推荐从 Google 商店或 Github 安装）：[Proxy SwitchyOmega 3 \(ZeroOmega\) - Microsoft Edge Addons](#)
- Chrome 插件安装地址：[Proxy SwitchyOmega - Chrome Web Store](#)
- Firefox 安装地址：[Proxy SwitchyOmega - Get this Extension for 🦊 Firefox \(en-US\)](#)
- 也可以在官方 Github 仓库 [FelisCatus/SwitchyOmega: No longer maintained, see pinned issues](#) 的 Releases 中下载。

（从中根据自己的浏览器选择对应的插件下载并安装就行，Edge 也可以下载 Chrome 的插件）

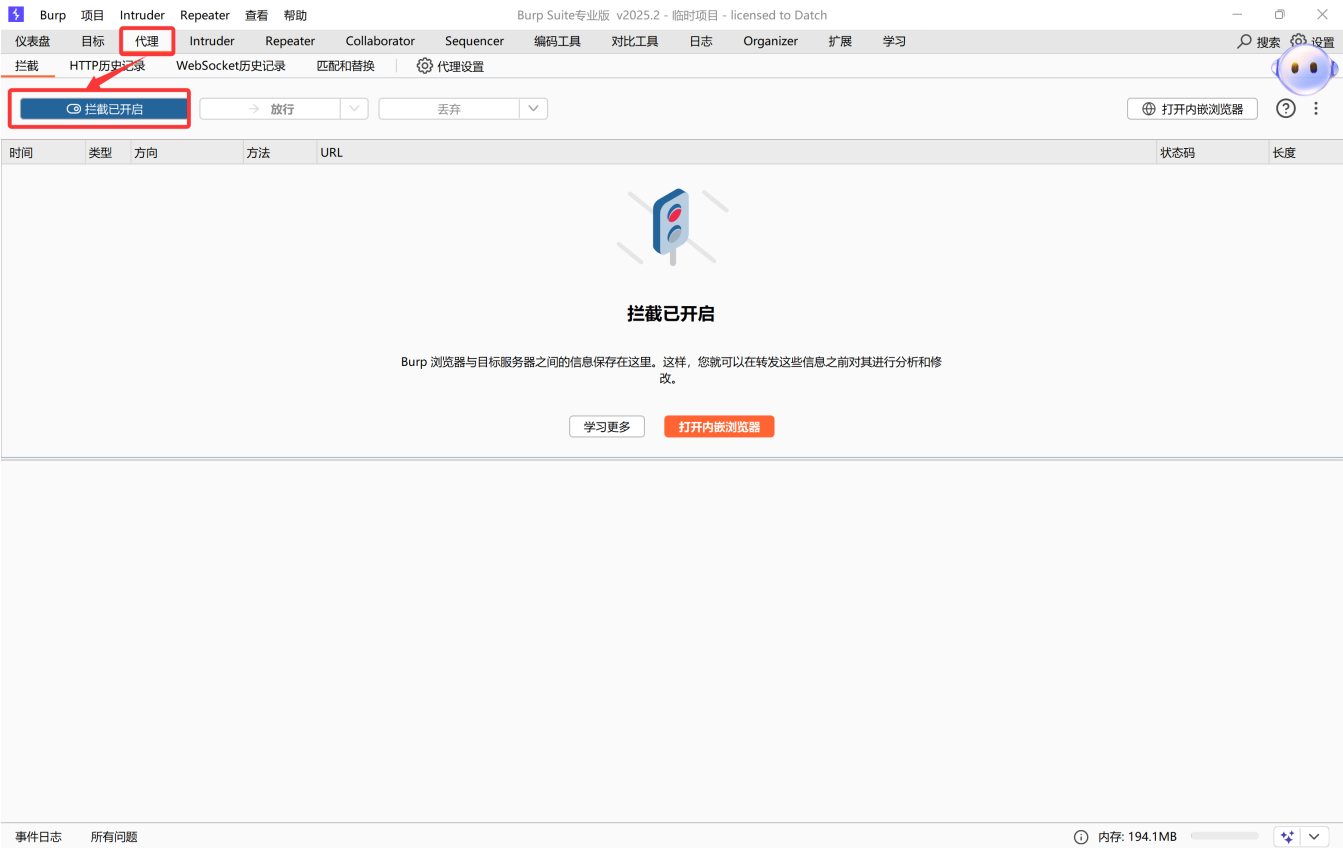
安装插件后，点击插件图标，点击选项，设置 Proxy 代理为 `localhost:8080`，便于用 BurpSuite 抓包：



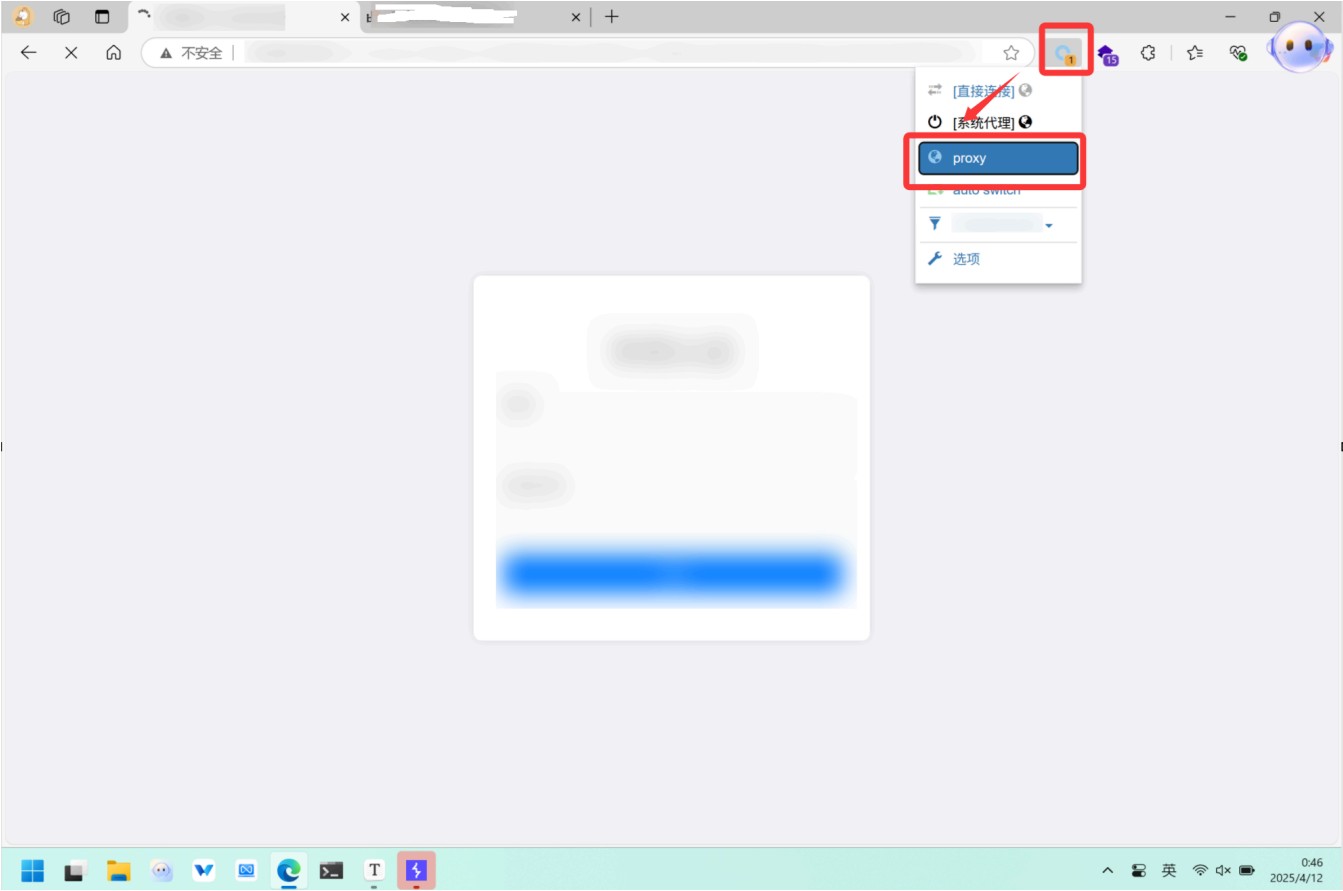
之后，点击插件图标，选择 proxy 就可以用 BurpSuite 的 proxy 抓包，点击系统代理就可以返回重新用系统设定的代理链接。

## 4. 基本抓包和发包的使用

我们首先打开 BurpSuite，点击 Proxy（代理）标签页，然后打开拦截（Intercept）

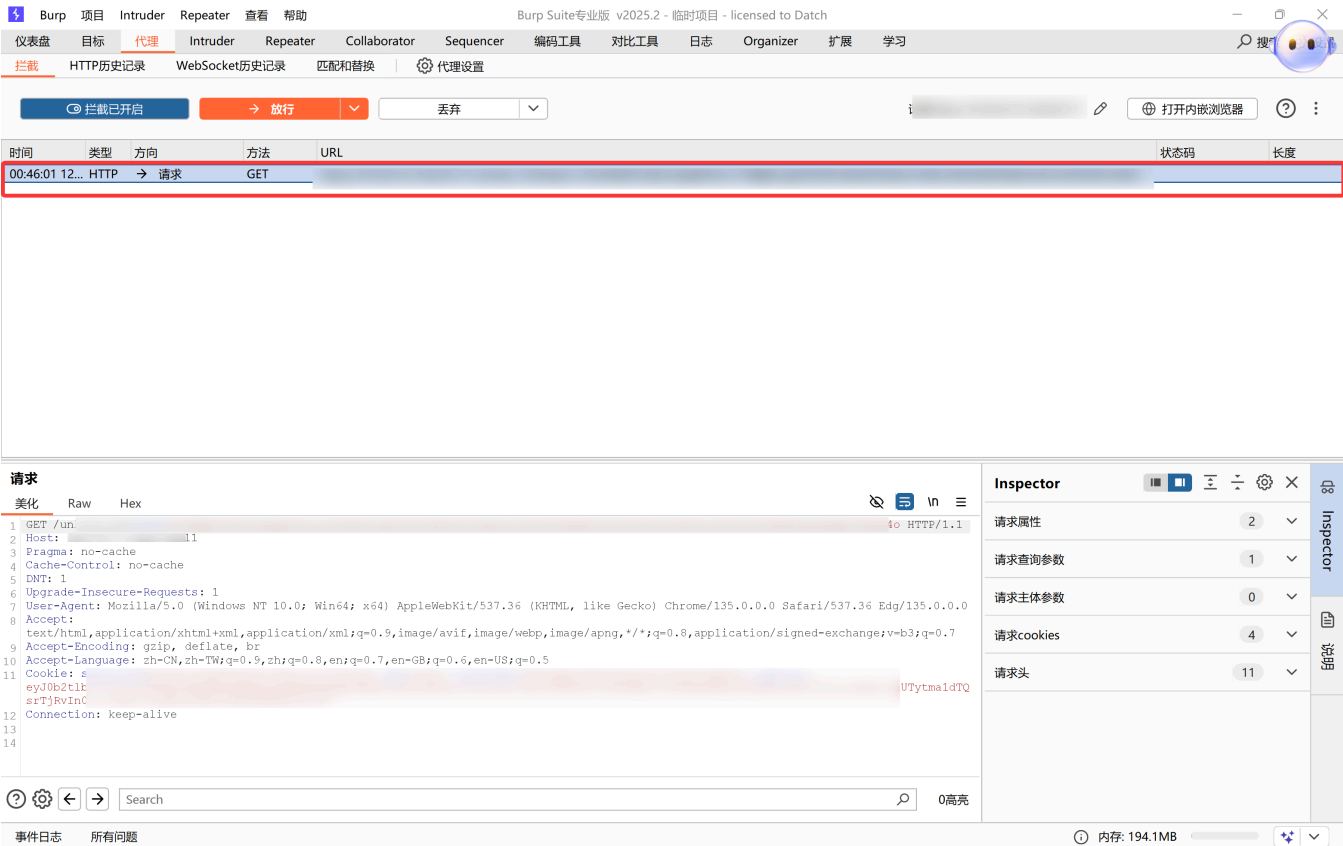


接着我们在浏览器打开需要被拦截的请求页面，使用 Proxy SwitchyOmega 打开 proxy 代理：

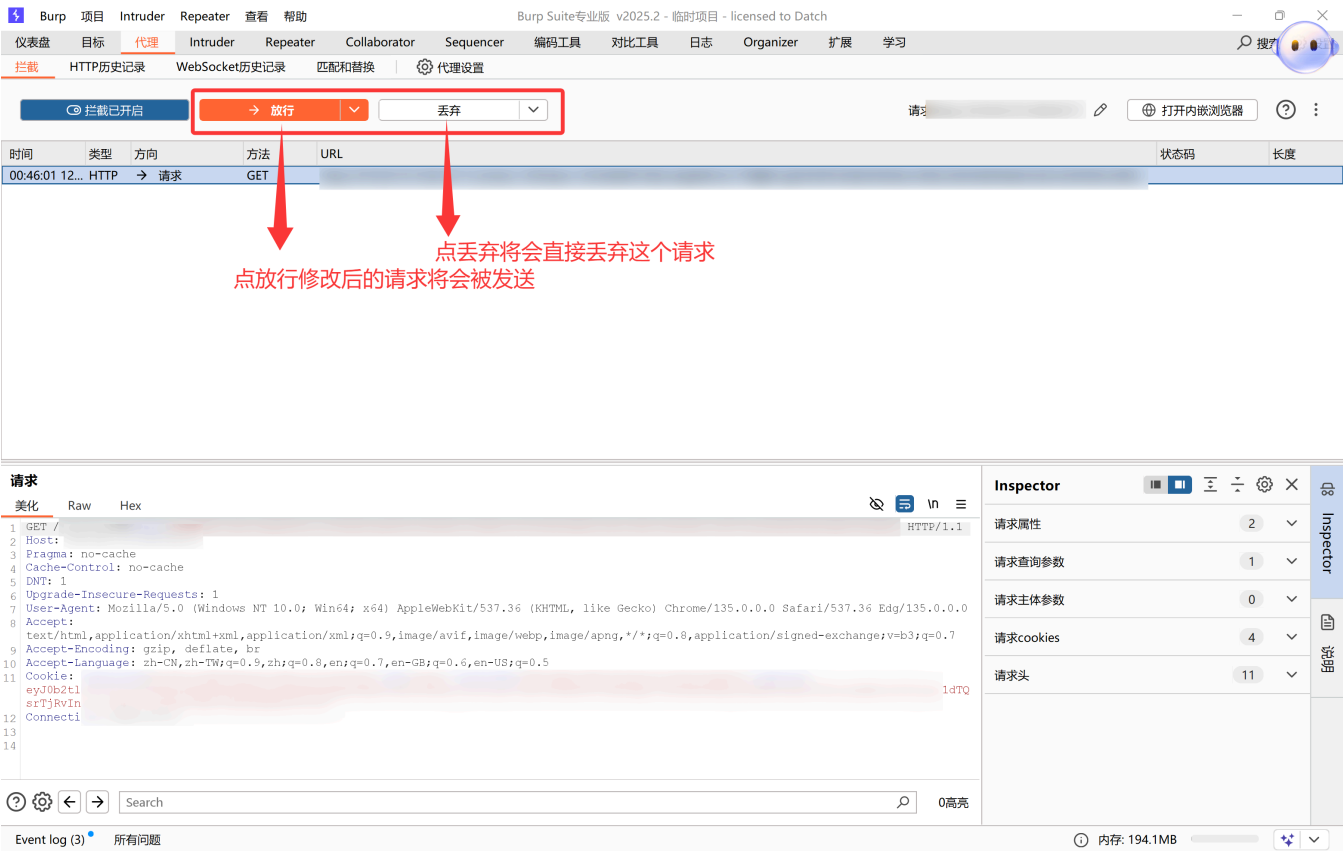




此时我们发现底部 BurpSuite 图标闪烁，打开后发现成功拦截到请求包，BurpSuite 展示了完整的 HTTP 请求包的内容：



此时我们可以直接编辑 HTTP 请求。然后点击上面的放行（Forward）按钮，就可以继续发送我们修改完成的 HTTP 请求了，丢弃（Drop）则是直接不要。请求得到的响应将会送回浏览器。



另外，修改后的请求可以用右键，选择“发送到Intruder”或者“发送到Repeater”进行进一步操作。其中，Intruder用于暴力破解，Repeater用于多次发包测试。关于 Intruder 的具体用法将在第五部分口令爆破中详细讲到。

## 四、目录爆破

### 1. 目录爆破简介

对于一些涉及敏感操作的网址，如网站备份或管理员页面，出于安全考虑，通常不会直接暴露。目录爆破是一种网络攻击技术，通常用于尝试发现目标网站或服务器上未公开的目录和文件。

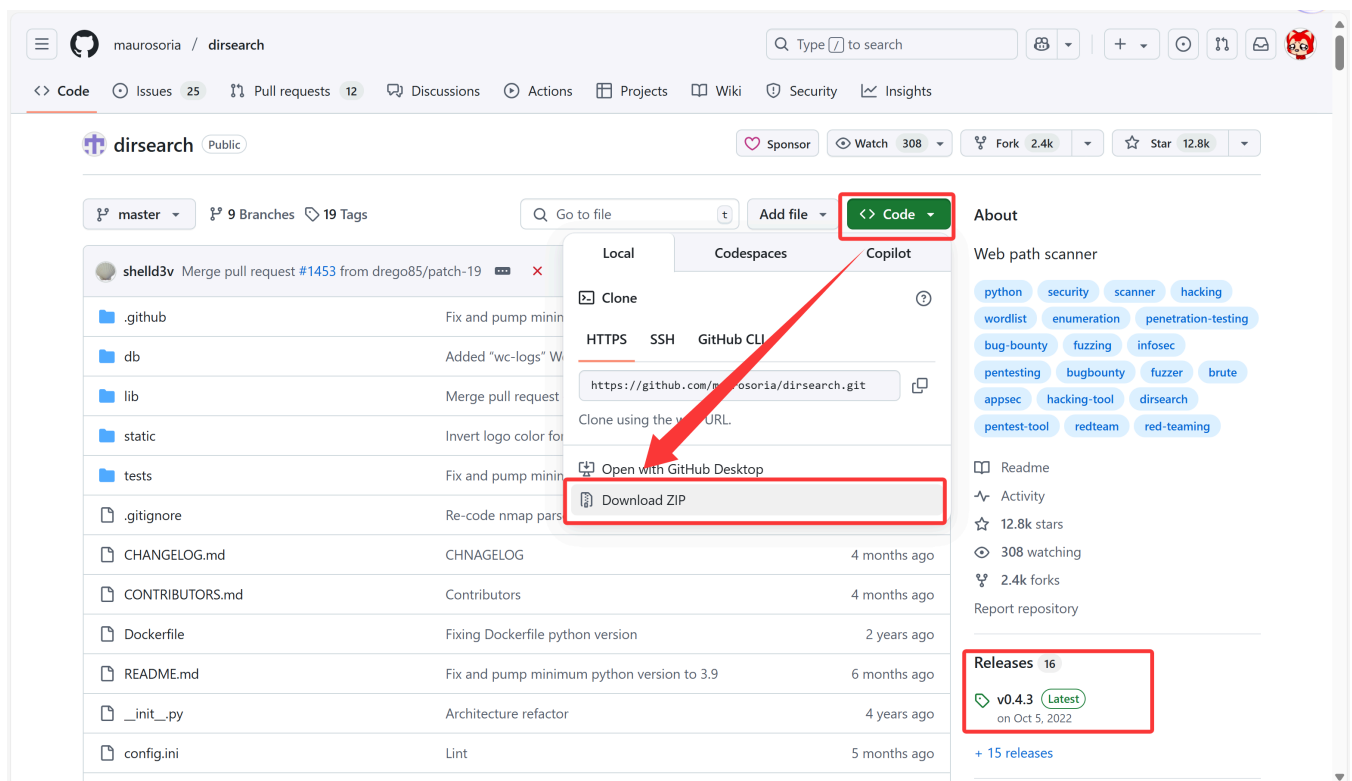
攻击者利用工具或脚本，向目标服务器发送大量包含不同目录和文件名的请求，通过分析服务器返回的响应状态码、内容等信息，来判断哪些目录或文件是存在的，哪些是不存在的。例如，当服务器返回状态码为 200 OK 时，通常表示请求的目录或文件存在；而返回 404 Not Found 等状态码，则表示未找到相应内容。

关于目录爆破的工具，主要有 dirsearch、dirbuster、BurpSuite 等。在这里，我们主要介绍 dirsearch 这一工具。

### 2. dirsearch 安装

dirsearch 需要 Python 来运行。首先，我们得准备好 Python 环境。

之后，我们访问 dirsearch 的官方 Github 仓库下载源码：[maurosoria/dirsearch: Web path scanner](https://github.com/maurosoria/dirsearch)。也可以在 Releases 页面，下载正式版本的源代码：



我们首先安装依赖包：

```
pip install -r requirements.txt
```

之后就可以使用了。

### 3. dirsearch 的使用

dirsearch 的基本使用方式是：

```
python dirsearch.py -u 链接 [其他选项]
```

例如，我们要爆破 `https://example.com` 的目录，我们就可以使用：

```
python dirsearch.py -u https://example.com
```

接下来，我们简单描述一下 dirsearch 其他的选项：

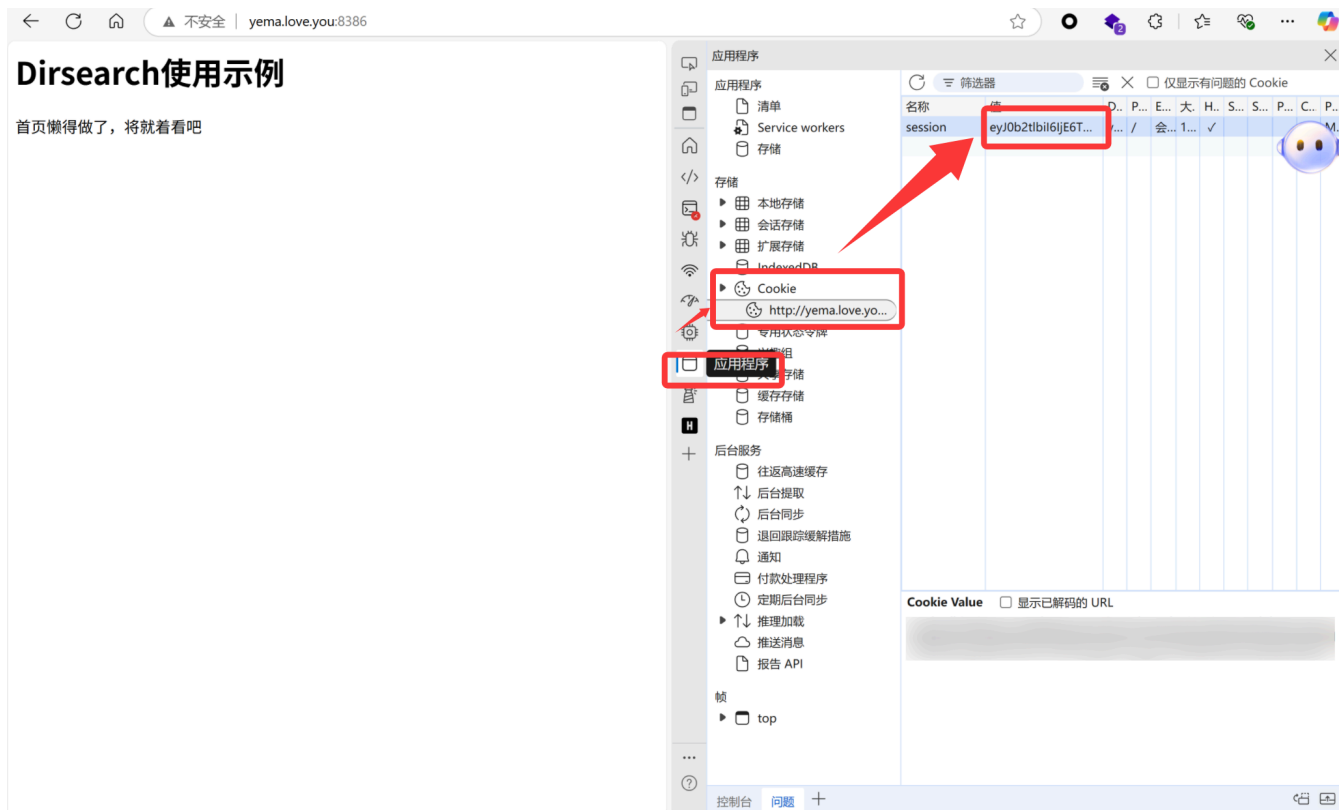
选项	描述	示例	示例说明
<code>-e EXTENSIONS</code> 或 <code>--extensions=EXTENSIONS</code>	指定要扫描的文件扩展名，多个扩展名之间用逗号分隔。	<code>-e php,asp,jsp</code>	扫描时会尝试查找以 <code>.php</code> 、 <code>.asp</code> 和 <code>.jsp</code> 为扩展名的文件和目录。
<code>--cookie=COOKIE</code>	设置请求时使用的 Cookie，用于模拟已登录状态或传递必要的会话信息。	<code>--cookie="session_id=12345; user=test"</code>	发送请求时会携带指定的 Cookie 信息， <code>session_id</code> 为 <code>12345</code> ， <code>user</code> 为 <code>test</code> 。
<code>-w WORDLIST</code> 或 <code>--wordlist=WORDLIST</code>	指定用于扫描的字典文件路径，字典文件中包含了要尝试的目录和文件名。	<code>-w /path/to/wordlist.txt</code>	使用 <code>/path/to/wordlist.txt</code> 文件中的内容作为扫描的字典。
<code>-i CODES</code> 或 <code>--include-status=CODES</code> <code>-x CODES</code> 或 <code>--exclude-status=CODES</code>	只包含指定状态码的扫描结果或排除指定状态码的扫描结果，多个状态码之间用逗号分隔。	<code>-i 200,300-399</code> <code>-x 301,500-599</code>	分别为：扫描结果只会显示返回状态码为 200 和 300-399 的结果；扫描结果不会显示状态码为 301 和 500-599 的结果。
<code>-r</code> 或 <code>--recursive</code> <code>-R RECURSIVE_LEVEL_MAX</code> 或 <code>-recursive-level-max=RECURSIVE_LEVEL_MAX</code>	<code>-r</code> 或 <code>--recursive</code> ：开启递归扫描模式，会在发现的目录下继续进行扫描； <code>-R</code> 或 <code>--recursive-level-max</code> ：指定递归扫描的最大深度。	<code>-r</code> <code>-R 2</code>	<code>-r</code> ：开启递归扫描，会不断深入发现的目录进行扫描； <code>-R 2</code> ：开启递归扫描，且最多深入到第二层目录。

另外，也可以设置 `-t THREADCOUNT` 来设置爆破线程数，用 `--random-user-agents` 使用随机的 User-Agent。这里只讲解了常用的配置，更多的参数设置请参考 dirsearch 的官方文档：[maurosoria/dirsearch: Web path scanner](#)。

### 4. dirsearch 的使用示例

我们以一个本地搭建的服务网站为例，讲解如何使用 dirsearch 来爆破网站的备份文件和后台地址。我们的网站地址是：`http://yema.love.you:8386`。我们先手动访问 <http://yema.love.you:8386>，被提示需要输入 token 才可以继续访问，所以在爆破的时候我们得用 Cookie 来伪造身份。

我们首先输入正确的 token，之后打开开发者工具，选择“应用程序”标签页，打开网页的 Cookie，将 session 的值复制。



然后，我们进行爆破，在爆破时，用 `-c` 指定 Cookie，将 xxx 替换为刚刚复制的 session 的值。关于其他的操作，我们保持默认即可（也可以根据具体情况调整后缀的爆破以及显示状态码的设置）。因为。所以，最后用如下命令

```
python dirsearch.py -u http://yema.love.you:8386 -c "session=xxx"
```

```
cliff azcliff v0.4.3
Extensions: php, asp, aspx, jsp, html, htm | HTTP method: GET | Threads: 25 | Wordlist size: 12289
Target: http://yema.love.you:8386/

[19:04:59] Scanning:
[19:05:03] 401 - 19B - /admin-serv/
[19:05:03] 308 - 261B - /admin-serv -> http://yema.love.you:8386/admin-serv/
[19:05:06] 308 - 253B - /backup -> http://yema.love.you:8386/backup/
[19:05:07] 403 - 27B - /backup/

Task Completed
```

我们发现，成功爆出了 `/admin-serv` 和 `/backup` 两个链接，根据名字可以猜测，`/admin-serv` 和 `/backup` 分别是管理员后台和备份目录。

对于 `/admin-serv/` 链接，返回码是 401 Unauthorized，这提示我们需要进一步 HTTP 认证。对于 `/backup/` 链接，返回码是 403 Forbidden，说明这很有可能是一个目录，需要我们进一步爆破，因此，用如下命令对 `/backup/` 目录进一步爆破：

```
python dirsearch.py -u http://yema.love.you:8386/backup -c "session=xxx"
```

成功扫到 `/backup/archive.zip`，下载即可得到网站的压缩文件：

```

c_l|_i_~_5_ c7_c_l|_i_~_c_l|_~_ v0.4.3

Extensions: php, asp, aspx, jsp, html, htm | HTTP method: GET | Threads: 25 | Wordlist size: 12289
Target: http://yema.love.you:8386/

[19:03:36] Scanning: backup/
[19:03:43] 200 - 18KB - /backup/archive.zip

Task Completed

```

## 五、口令爆破

想成为带骇客，你肯定要学会破解别人的用户名和密码。以登录为例，你在登录框输入用户名和密码之后，浏览器会把你的用户名和密码发送给服务器进行验证，并返回结果。

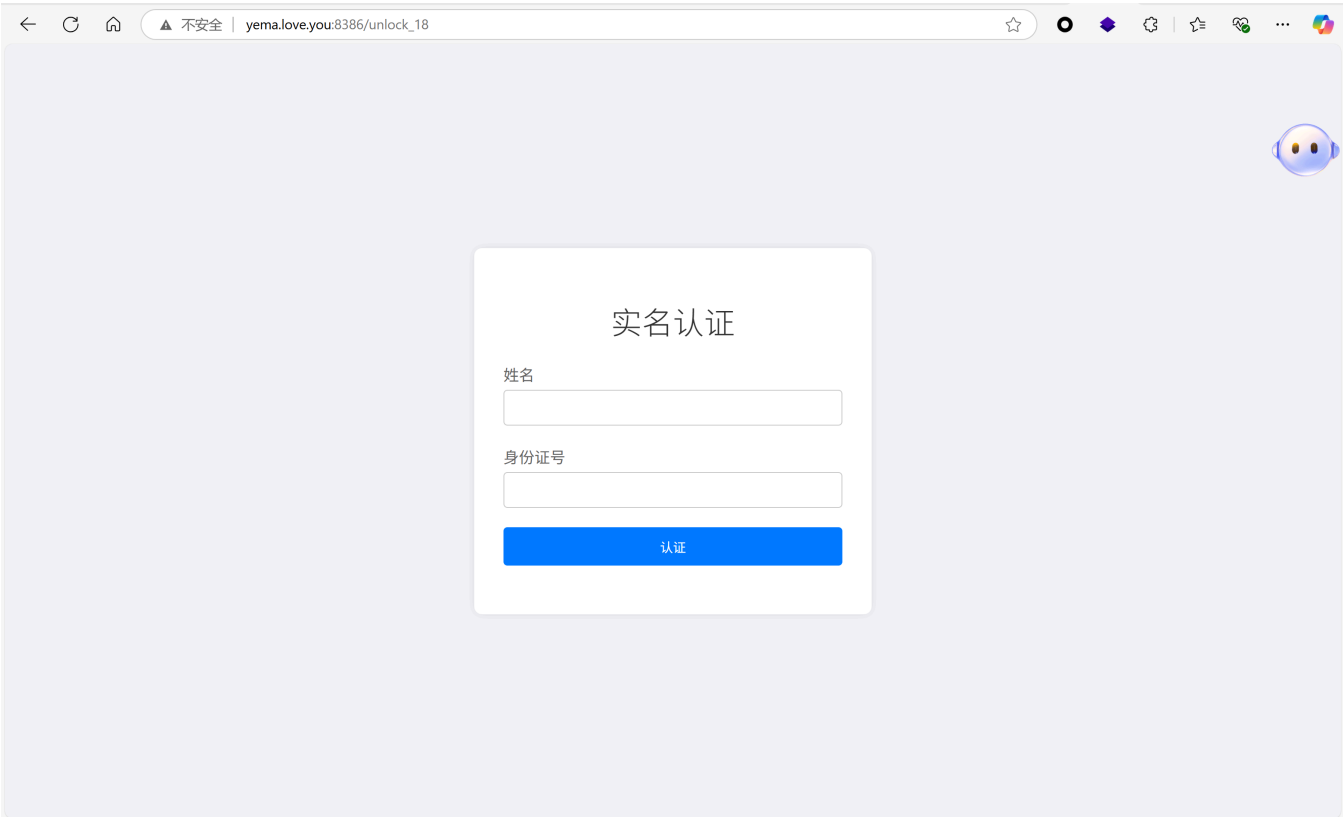
这和目录爆破的原理颇为相似。我们可以预先准备一个存有大量用户名和密码组合的字典文件，然后借助专门的软件按照字典里的内容逐个尝试登录。软件会根据服务器返回的响应情况，判断当前尝试的用户名和密码是否准确。经过不断尝试，最终便能找出正确的账号和密码，这一系列操作就是所谓的口令爆破。

口令爆破的基本操作手法是：1. 随便输入点信息提交，并使用抓包软件抓包；2. 确定 HTTP 包里面需要爆破的位置，设置好爆破的 Payload 3. 开始爆破。

### 1. 明文验证爆破

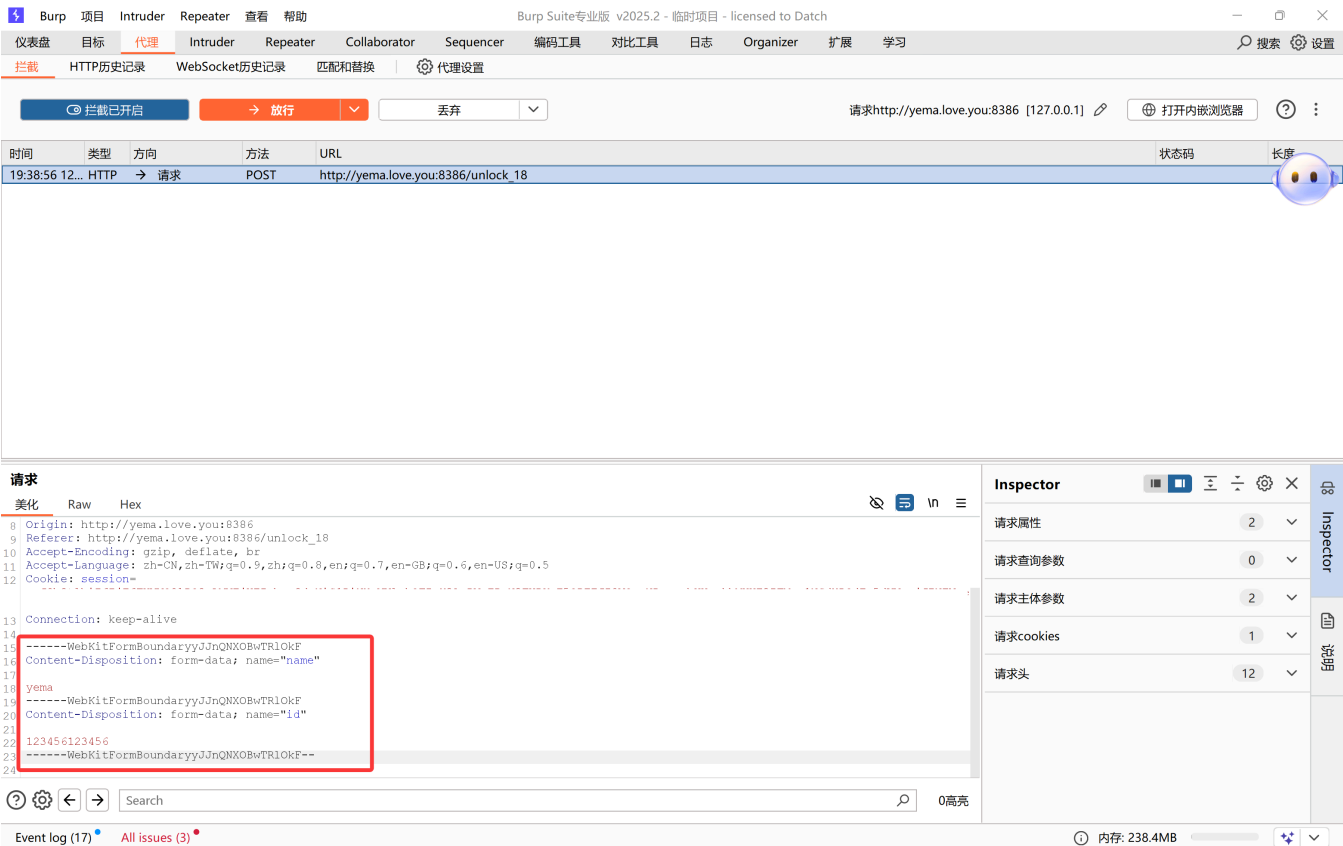
我们以一个模拟实名认证的网站，来讲解如何用 BurpSuite 对明文验证的网站进行爆破。

测试网站网址是：`http://yema.love.you:8386/unlock_18`：



这里面我们假设已知姓名是 `yema`，并且身份证号是 `12345620050418xxxx`，现在需要对最后的 4 位数字进行爆破。我们先用 BurpSuite 进行抓包。首先开启 BurpSuite，在 Proxy 页面设置拦截（Intercept）开启，浏览器设置代理为 Proxy（即 `http://localhost:8080`），随便输入姓名和身份证号，点击认证。

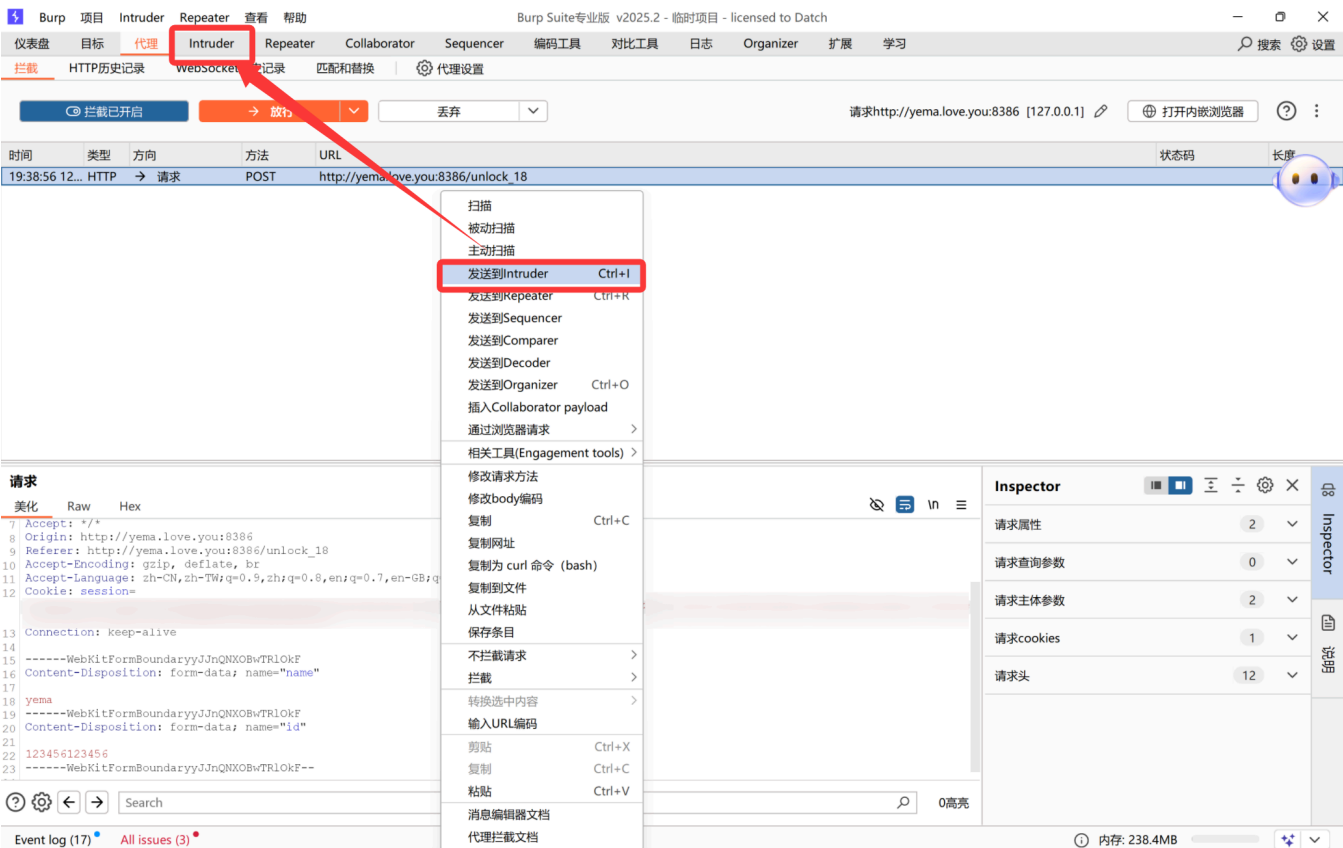
然后可以在 BurpSuite 看到请求包：



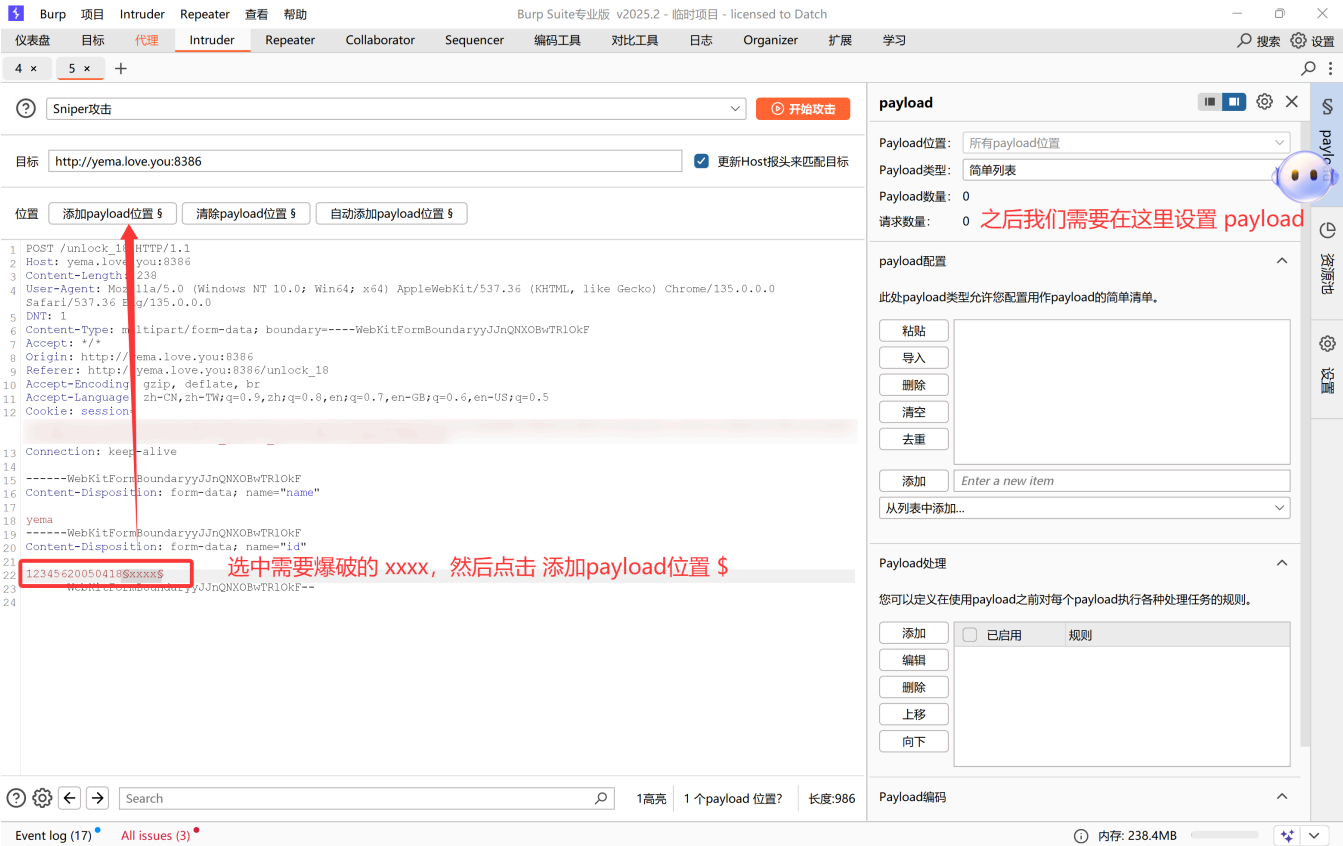
**Note**

注意需要在包中看到我们输入的姓名和身份证号。

然后我们右键 HTTP 包，点击发送到 Intruder，然后打开 Intruder 标签页：



在 Intruder 中，我们得先设置 Payload 位置。我们先将身份证号的位置改成我们需要爆破的字符串：  
12345620050418xxxx，然后选中需要爆破的后四位，也就是 xxxx，然后点击“添加 Payload 位置\$”，此时它变成  
12345620050418\$xxxx\$。



因为我们是爆破 4 位数字，因此设置 Payload 类型为“数值”，数字范围改成 0000~9999，并设置整数最小位数为 4（防止前导 0 被舍去），然后点击开始爆破：

payload

Payload位置: 所有payload位置

Payload类型: 数值

Payload数量: 10,000

请求数量: 10,000

先设置 数值

payload配置

生成给定范围内指定格式的有效数值内容。

数字范围

类型: ☒ 顺序 ☐ 随机

从: 0000

到(To): 9999

间隔: 1

设置数字范围0000-9999  
间隔为1

数量:

数值格式

基数(Base): ☒ 十进制 ☐ Hex

整数最小位数: 4

整数最大位数: 4

小数最小位数: 0

小数最大位数: 0

示例

0001

4321

因为成功和失败的返回结果肯定有不同，所以我们在这里对“长度”进行排序，找出长度不一致的一项，这一项的 Payload 就是正确的结果：

攻击 保存

9. Intruder attack of http://yema.love.you:8386

攻击 保存 暂停 帮助

结果 位置

捕获过滤: 捕捉所有项目

视图过滤: 显示所有条目

应用捕捉过滤器

请求	payload	状态码	接收到响应	错误	超时	长度	注释
1235	1234	200	1			154	
0		200	5			184	
1	0000	200	5			184	
3	0002	200	2			184	
4	0003	200	5			184	
5	0004	200	3			184	
8	0007	200	5			184	
9	0008	200	4			184	

长度154，和其他的184明显不一样

因此我们成功爆破出最后的四位数字是 1234。

## 2. HTTP 认证爆破

常见的 HTTP 认证方式有基本认证（Basic Authentication）和摘要认证（Digest Authentication）。基本认证是将用户名和密码进行 Base64 编码后放在请求头中发送给服务器，服务器对其进行解码和验证。摘要认证则是通过对用户名、密码、请求方法、请求 URI 等信息进行哈希运算，生成一个摘要值发送给服务器进行验证，相对基本认证更安全一些。在这里，我们只考虑基本认证，摘要认证的爆破方式类似。

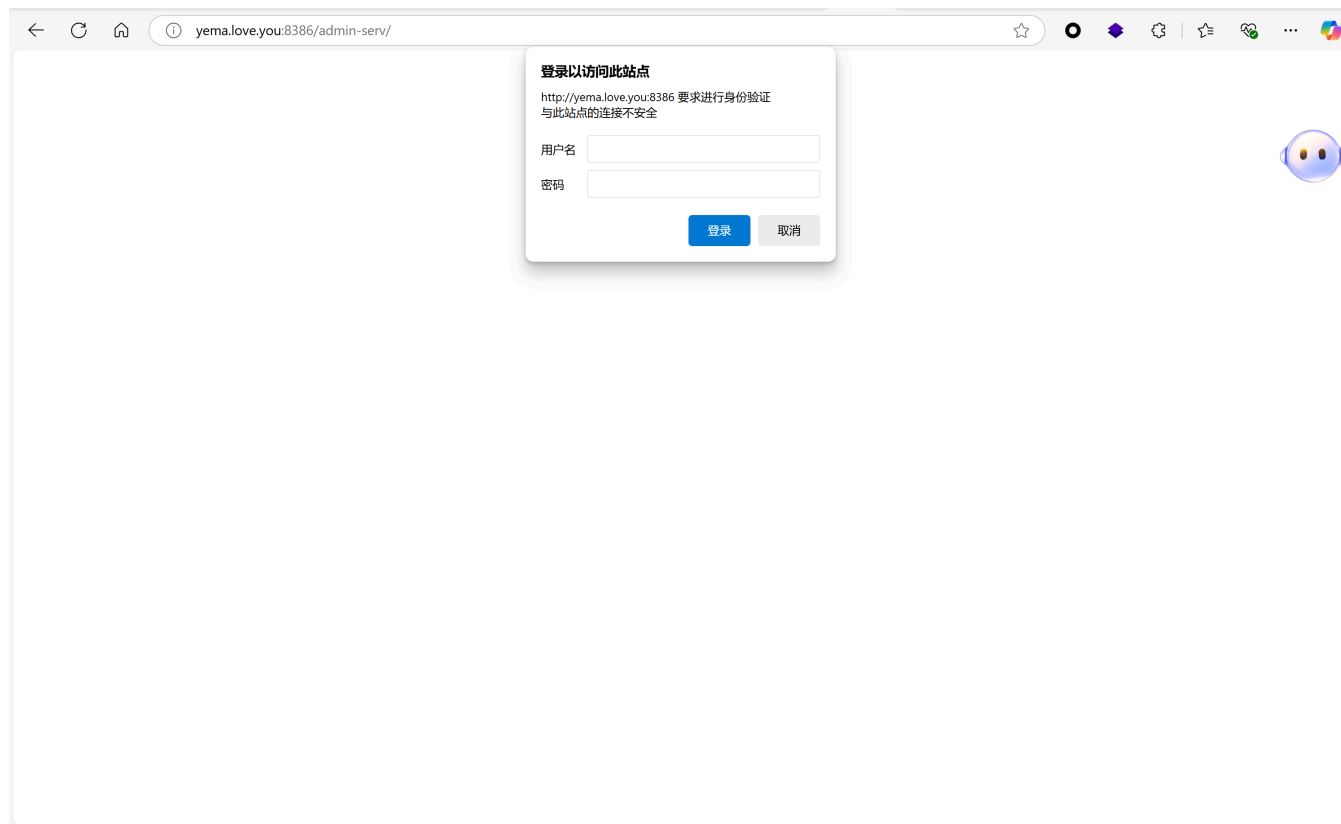
在请求时往往会带上一个 Authorization 的头：该头的格式为 `Authorization: Basic <Base64编码后的用户名:密码>`。



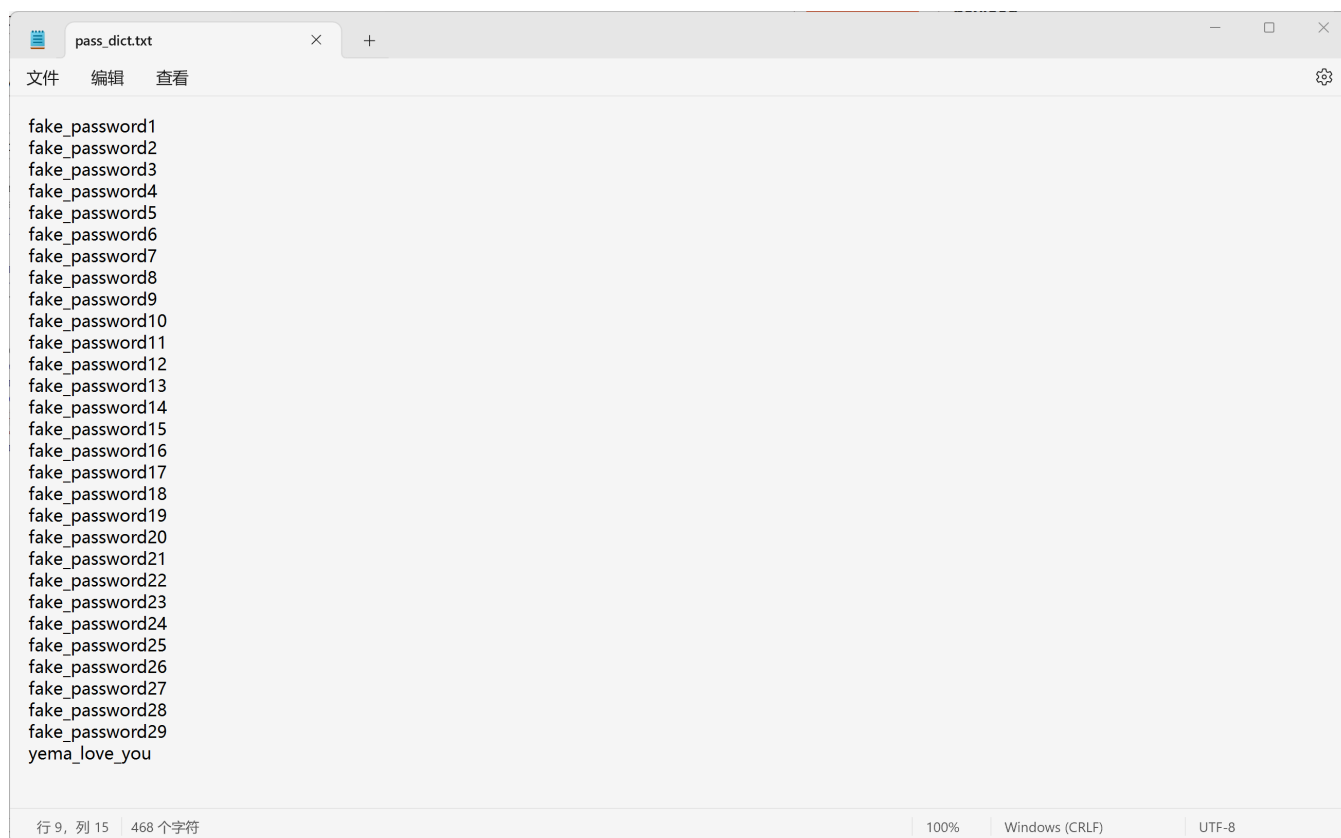
## Note

其实是想以此说明如何爆破前端对数据进行初步处理的情况。

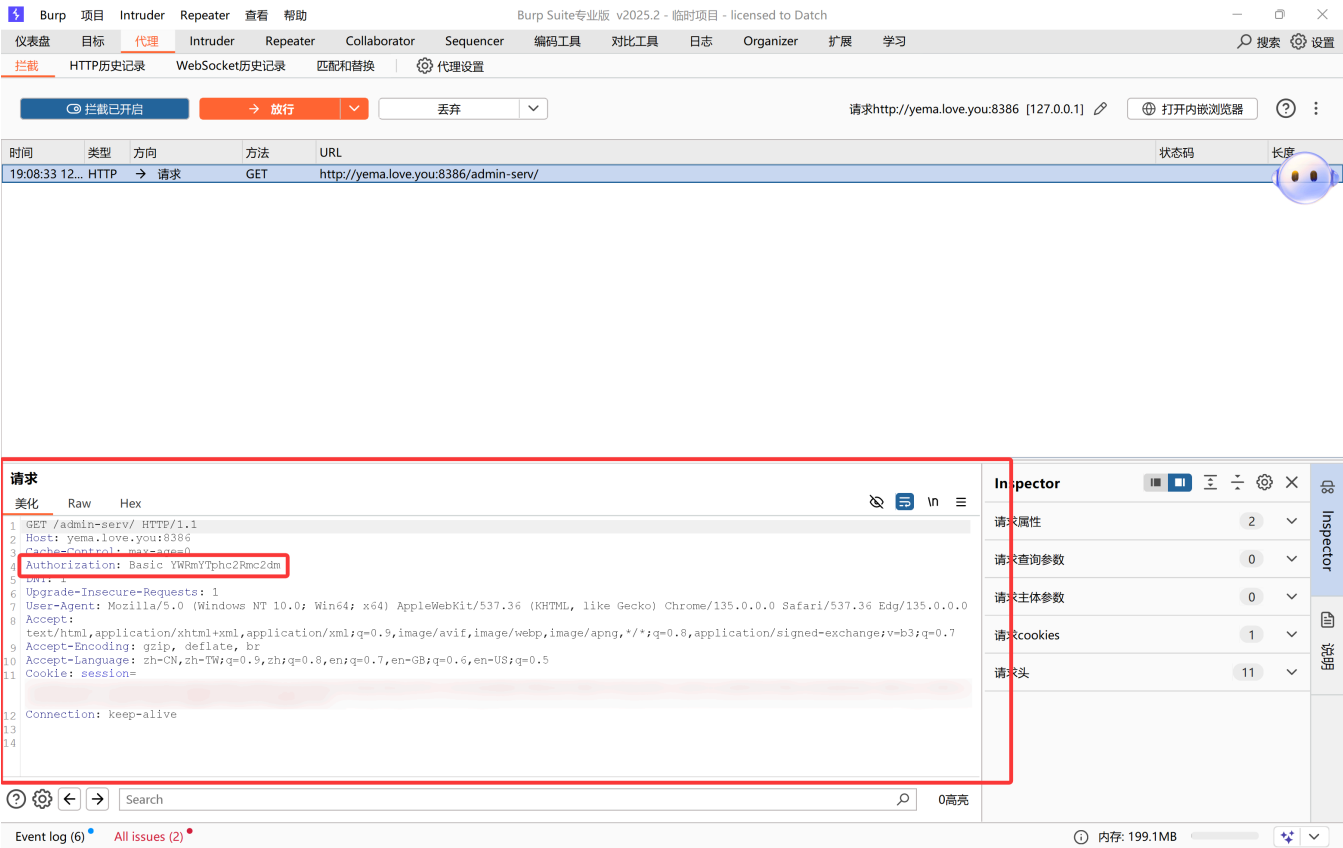
以网站 `http://yema.love.you:8386/admin-serv/` 为例，该网站提示需要进行 HTTP 认证：



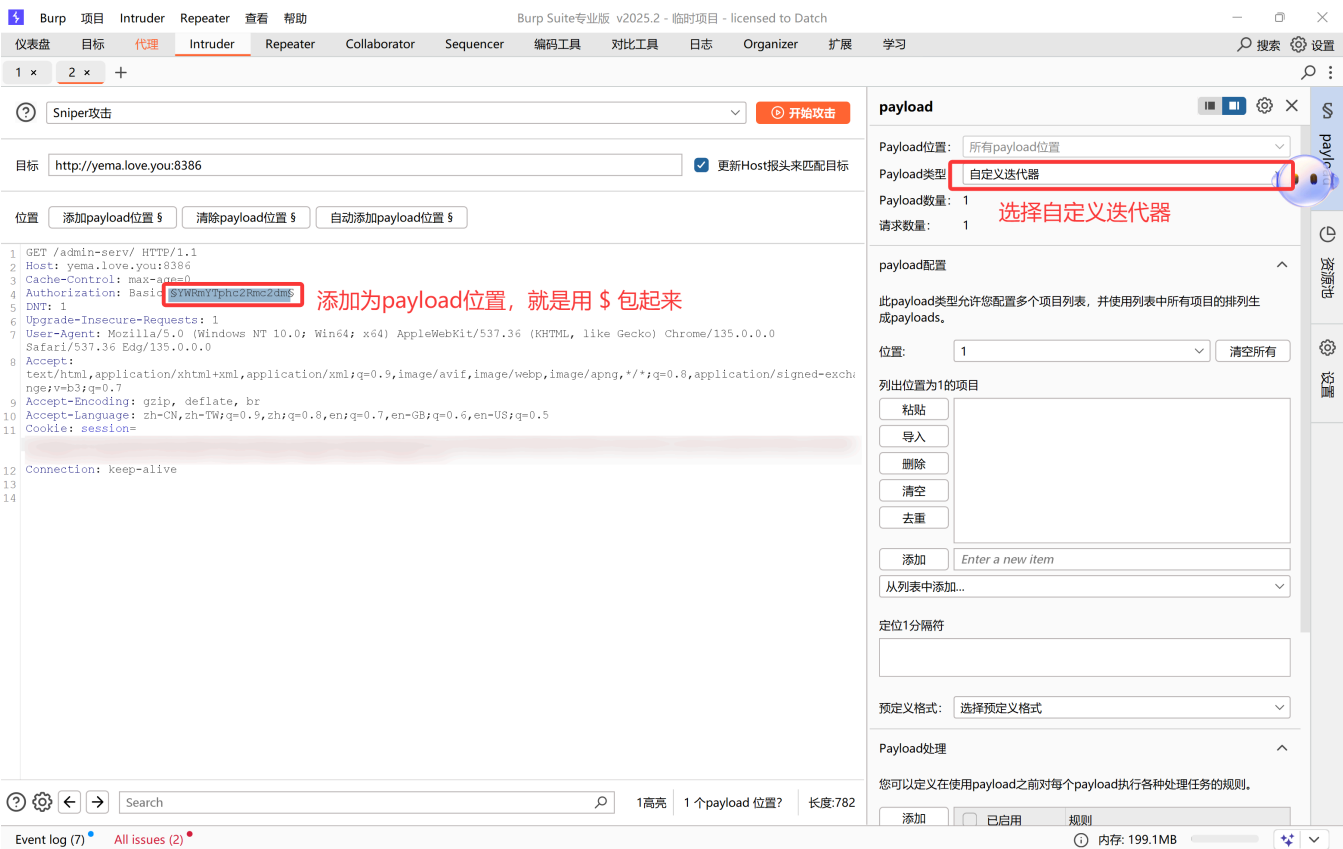
关于这个网站，我们假定我们已经获取用户名为 `yema` 了，并且获得了密码字典 `pass_dict.txt`：



和明文爆破类似，我们先用 BurpSuite 抓取请求包。开启 Proxy，然后随便输入点用户名和密码，点击登录。即可在 BurpSuite 抓到请求包：



我们发现，在请求头中有 Authorization 头，并且采用基础认证方式。图中 YWRmYTphc2Rmc2dm 即我们输入的用户名:密码 经过 base64 编码的结果。我们也是对这一部分进行爆破。我们先把请求发送到 Intruder。



首先将 YWRmYTphc2Rmc2dm 添加 Payload 配置。然后在右侧，设置：Payload 类型为“自定义迭代器”。因为我们已知用户名是 yema，所以最后我们的 Payload 是 yema:密码 的 base64 编码。

在下面的 Payload 配置中，首先选择位置 1，输入 yema: 并添加（注意冒号不能缺少，因为最后我们的 Payload 是 yema:密码 的 base64 编码）。然后选择位置 2，点击导入，将 pass\_dict.txt 的内容导入进去：

payload

Payload位置: 所有payload位置

Payload类型: 自定义迭代器

Payload数量: 1

请求数量: 1

payload配置

此payload类型允许您配置多个项目列表，并使用列表中所有项目的排列生成payloads。

位置: 1

清空所有

列出位置为1的项目

粘贴

导入

删除

清空

去重

添加

yema:

从列表中添加...

定位1分隔符

预定义格式: 选择预定义格式

Payload处理

您可以定义在使用payload之前对每个payload执行各种处理任务的规则。

添加

已启用

规则

位置: 1

清空所有

列出位置为1的项目 (1)

粘贴

导入

删除

清空

去重

添加

yema:

Enter a new item

从列表中添加...

位置: 2 清空所有

列出位置为2的项目 (30)

<span style="border: 1px solid #ccc; padding: 2px;">粘贴</span>	fake password1
<span style="border: 2px solid red; padding: 2px;">导入</span>	fake password2
<span style="border: 1px solid #ccc; padding: 2px;">删除</span>	fake password3
<span style="border: 1px solid #ccc; padding: 2px;">清空</span>	fake password4
<span style="border: 1px solid #ccc; padding: 2px;">去重</span>	fake password5
	fake password6
	fake password7
	fake password8
<span style="border: 1px solid #ccc; padding: 2px;">添加</span>	<input type="text" value="Enter a new item"/>
<span style="border: 1px solid #ccc; padding: 2px;">从列表中添加...</span>	

选择位置2，点击导入，  
打开准备好的密码文件

接着，在下面的 Payload 处理中，点击“添加”，在弹出的对话框中，上面一行选择“编码”，下面一行选择“Base64-encode”，即配置 base64 编码。并且取消掉下面的 Payload 编码：

The screenshot shows the Burp Suite interface. A dialog box titled '添加payload处理规则' (Add payload processing rule) is open, showing '编码' (Encoding) selected in the first dropdown and 'Base64-encode' in the second. A red arrow points from the 'Base64-encode' option to the '添加' (Add) button in the 'Payload处理' (Payload processing) panel on the right. Another red arrow points from the 'Base64-encode' option to the '确认' (Confirm) button in the dialog. The 'Payload处理' panel shows the '添加' button highlighted. Below it, the 'Payload编码' (Payload encoding) section has the 'URL编码' (URL encoding) checkbox unchecked, with a red arrow pointing to it and the text '这里不要勾选!' (Don't check here!).

然后我们点击开始攻击即可开始攻击：

攻击 保存

8. Intruder attack of http://yema.love.you:8386

攻击 ▼ 保存 ▼

结果 位置

捕获过滤: 捕捉所有项目

视图过滤: 显示所有条目

请求	payload	状态码	接收到响应	错误	超时	长度	注释
30	eWVtYTp5ZW1hX2xvdmVfeW91	200	11			4208	
0		401	4			179	
1	eWVtYTp5ZW1hX2xvdmVfeW91	401	5			179	
2	eWVtYTp5ZW1hX2xvdmVfeW91	401	5			179	
3	eWVtYTp5ZW1hX2xvdmVfeW91	401	5			179	
4	eWVtYTp5ZW1hX2xvdmVfeW91	401	6			179	
5	eWVtYTp5ZW1hX2xvdmVfeW91	401	5			179	
8	eWVtYTp5ZW1hX2xvdmVfeW91	401	2			179	

未认证的请求状态码是 401 Unauthorized，而成功认证的则不是。所以我们对状态码进行排序，发现 `ewvtYTp5ZW1hX2xvdmVfeW91` 这个状态码为 200，所以我们对其 Base64 解码之后即可得到密码：

```
$ echo eWVtYTp5ZW1hX2xvdmVfeW91 | base64 -d  
yema:yema_love_you
```

因此用户名是 `yema`，密码是 `yema_love_you`。

#### Note

有兴趣的同学也可以尝试用浏览器抓包之后，转为 Python 代码。用 Python 代码进行爆破。

## 3. Hash 爆破

前面我们介绍的都是对后端某个 API 进行爆破。不过某些时候，我们可以得到系统的判定规则，并且需要爆破出符合判定规则的字符串，例如知道密文的 md5 值反求出密文等。接下来，我们讲解运用 Python 脚本来爆破这种情况。

以如下 JavaScript 判定代码为例。首先，我们需要明白判定代码的意义，并将其转换为 Python 代码：

```
function checkActivationCode(code) {  
    const format = /^[a-zA-Z0-9]{4}$/;  
    if (!format.test(code)) {  
        return false;  
    }  
  
    const key = md5(code);  
    const int_pos = [2, 4, 5, 6, 8, 9, 12, 13, 14, 15, 17, 18, 19, 20, 21, 24, 27, 28, 29,  
31];  
    for (let i = 0; i < key.length; i++) {  
        if (int_pos.includes(i)) {  
            if (!/\d/.test(key[i])) {  
                return false;  
            }  
        } else {  
            if (!/[a-zA-Z]/.test(key[i])) {  
                return false;  
            }  
        }  
    }  
    if (key[4] == key[29] && key[9] == key[21] && key[10] == key[11] && key[26] == key[30])  
{  
        return true;  
    } else {  
        return false;  
    }  
}
```

可以看出，这个函数首先将接受的 `code` 计算 md5 值并存储到 `key`，之后定义了一个 `int_pos` 指明了 `key` 中数字的位置，并判定 `key` 中部分位置是否相同。

根据这个，我们写一个 Python 版的代码（简单的代码也可交给 AI 完成）：

```
import re  
import hashlib
```

```
def check_activation_code(code):
    # 激活码只能包含字母和数字，并且长度为4位
    format_pattern = re.compile(r'^[a-zA-Z0-9]{4}$')
    if not format_pattern.match(code):
        return False

    # 激活码的 md5 值必须满足某些条件
    key = hashlib.md5(code.encode()).hexdigest()
    int_pos = [2, 4, 5, 6, 8, 9, 12, 13, 14, 15, 17, 18, 19, 20, 21, 24, 27, 28, 29, 31]
    for i in range(len(key)):
        if i in int_pos:
            if not re.match(r'\d', key[i]):
                return False
        else:
            if not re.match(r'[a-zA-Z]', key[i]):
                return False

    if key[4] == key[29] and key[9] == key[21] and key[10] == key[11] and key[26] == key[30]:
        return True
    return False
```

因为我们知道激活码是 4 位的，所以我们暴力枚举 4 位的激活码，判定 check\_activation\_code 的返回值是否为 True 即可：

```
import tqdm

dic = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
total_iterations = len(dic) ** 4
progress_bar = tqdm.tqdm(total=total_iterations)

for a in dic:
    for b in dic:
        for c in dic:
            for d in dic:
                code = a + b + c + d
                if check_activation_code(code):
                    print(f"code: {code}")
                    progress_bar.close()
                    exit(0)
            progress_bar.update(1)
progress_bar.close()
```

最后成功得到结果 code: 10v3。

```
19%|██████████| 2803705/14776336 [00:13<01:11, 166526.19it/s]c
ode: 10V3
19%|██████████| 2824465/14776336 [00:13<00:55, 215584.33it/s]
```