

# 使用HaE获取你的第一笔十万赏金

看到标题也许有人会觉得这是夸大，但笔者可以负责任的说HaE给我带来的价值远超万元。大多数人只是把HaE当作一个敏感信息发现的工具，而实际上如果你可以深入了解HaE，了解它每个规则，基于HaE所匹配到的大量数据进行关联，你会发现更多高危漏洞，获得更多的赏金。

漏洞积分 ∨ ⓘ	赏金 ∨	报告质量	漏洞星级	有效性
+83 (分)	+4000.00 (元)	良	★ ★ ★ ★ ★	确认漏洞
+83 (分)	+4000.00 (元)	良	★ ★ ★ ★ ★	确认漏洞
+83 (分)	+4000.00 (元)	良	★ ★ ★ ★ ★	确认漏洞
+83 (分)	+4000.00 (元)	良	★ ★ ★ ★ ★	确认漏洞
+83 (分)	+4000.00 (元)	良	★ ★ ★ ★ ★	确认漏洞


本文就分享一下笔者所掌握的HaE使用技巧和思路。

HaE项目地址：<https://github.com/gh0stkey/HaE>

## 规则

## Cloud Key

Cloud Key用于发现各类云的SecretKey和AccessKey。

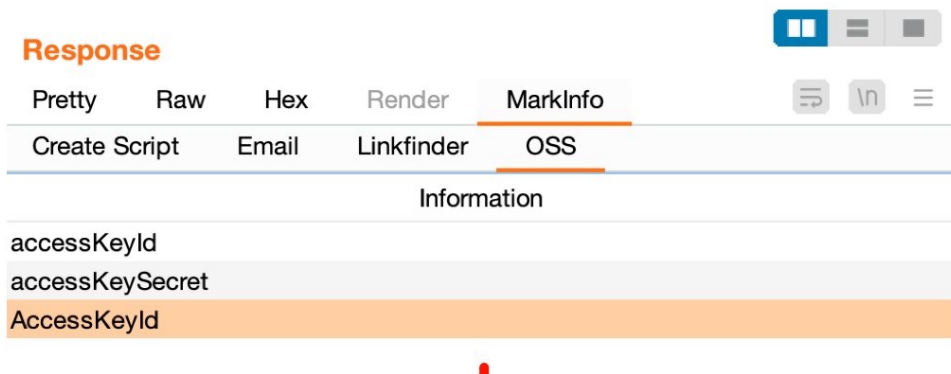


Name:	Cloud Key
F-Regex:	<code>((access)(- _)(key)(- _)(id secret)) (LTAI[a-z0-9]{12,20}))</code>
S-Regex:	
Format:	<code>{0}</code>
Scope:	any
Engine:	nfa
Color:	yellow
Sensitive:	false

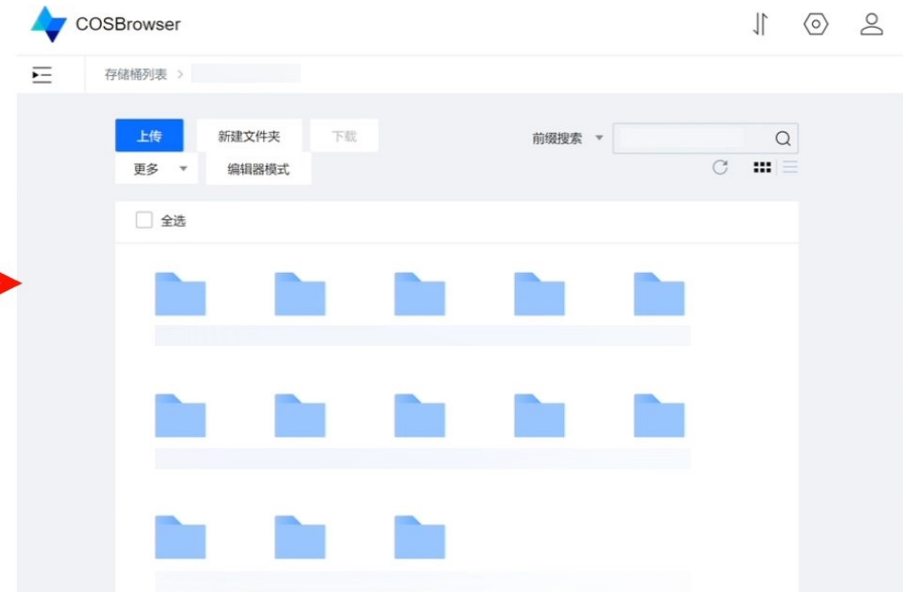
否(N)

是(Y)

这个规则就是纯帮助你一键找到高危漏洞了。用这个接管了很多存储桶，甚至有的可以直接供应链攻击，官网的安装包也在这个存储桶内，直接替换。当然也有一些具备VPS管理权限的，可以控机器。



```
GXLr:function(t,e,s){  
  "use strict";  
  var a=s("Zrlr"),i=s.n(a),n=s("wxAW"),o=s.n(n),r=s("/Z0X"),l  
  =s.n(r),c=function(){  
    function t(){  
      i()(this,t),this.client=new l.a({  
        region:"[redacted]",accessKeyId:  
        "[redacted]",accessKeySecret:  
        "[redacted]",bucket:"[redacted]"  
      })  
    }  
  }  
}
```



## Linkfinder

Linkfinder用于匹配返回报文中的请求路径信息（API）。

Edit Rule

?

Name:

Linkfinder

F-Regex:

3,}(?:[\\?|#][^"']{0,})|([a-zA-Z0-9\_\\-]{1,}\\.(?:\\w)(?:[\\?|#][^"']{0,})))(?:'|")

S-Regex:

Format:

{0}

Scope:

response body

▼

Engine:

nfa

▼

Color:

gray

▼

Sensitive:

true

▼

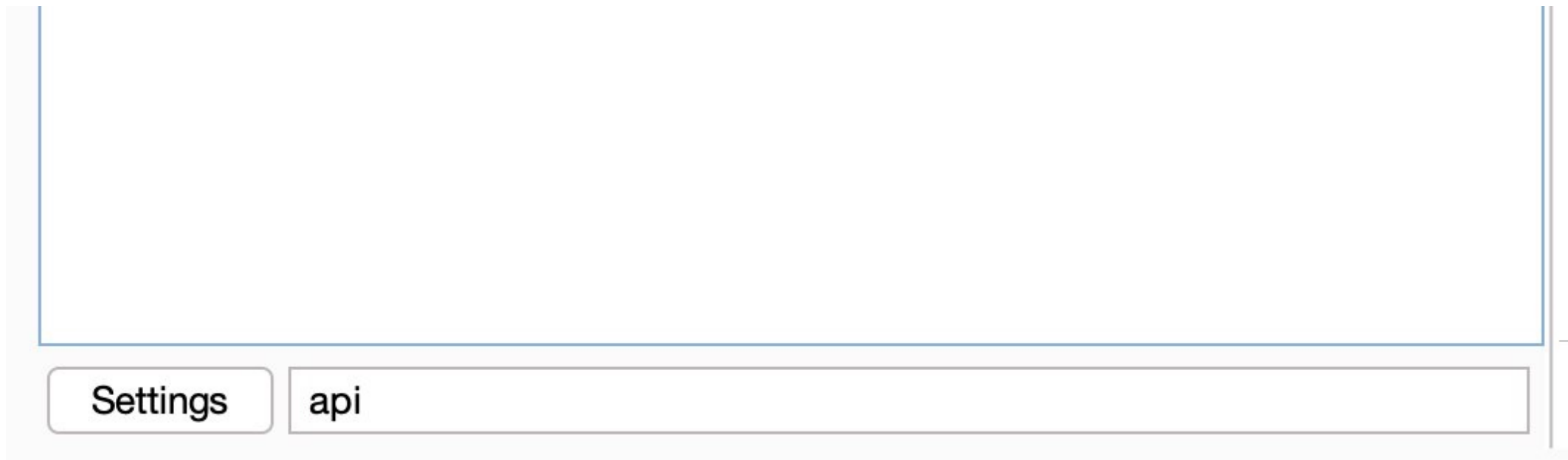
否(N)

是(Y)

我觉得这个规则应该是出洞率最高的规则，通常情况下使用它都可以发现未授权访问漏洞。但需要注意的是很多人拼接的路径不正确导致错失高危，我建议先找个登录点抓个包看看路径是否有一个基路径（一级目录），然后再拼接跑一遍。

Create Script (21)	All URL (25)	Linkfinder (48)	Upload Form (1)	> ▼
#	Information ^			

17	/api/v1/user/bind
16	/api/v1/user/bind_code
32	/api/v1/user/check_name
35	/api/v1/user/forgot_pwd
30	/api/v1/user/forgot_pwd_code
19	/api/v1/user/login
24	/api/v1/user/login_out
20	/api/v1/user/register
18	/api/v1/user/upd_pwd
34	/api/v1/user/user_info
29	/api/v1/verifycode



## Create Script

Create Script用于自动拼接前端JS中动态创建JS引用的JS路径，发现无法直接抓包获取到的JS文件路径信息。

Edit Rule

?

Name:

Create Script

F-Regex:

(\{[^}]\*\}\s\*\{[^s]\*\}\s\*\{+\s\*"[^s]\*\.js")

S-Regex:

"?([\w].\*?)"?:"(.\*)"

Format:

{0}.{1}

Scope:

response body

Engine:

nfa

Color:

green

Sensitive:

false

否(N)

是(Y)

如图所示的JS文件，以 `static/js/{...}.js` 作为JS文件地址赋值给到script标签的src属性值，也就表示这里有许多的JS文件，例如其中一个拼接出来的地址就是 `static/js/0.fbcd72b6763971b2e095.js`



```

!function(e){var n=window.webpackJsonp;window.webpackJsonp=function(r,o,a){for(var f,d,i,u=0,b=[];u<r.length;u++)d=r[u],t[d]&&b.push(t[d][0]),t[d]=0;for(f in o)Object.prototype.hasOwnProperty.call(o,f)&&(e[f]=o[f]);for(n&&n(r,o,a);b.length;)b.shift();if(a)for(u=0;u<a.length;u++)i=c(c.s=a[u]);return i};var r={},t={23:0};function c(n){if(r[n])return r[n].exports;c.e=function(e){var n=t[e];if(0===n)return new Promise(function(e){e()});if(n)return n[2];var r=new Promise(function(r,c){n=t[e]=[r,c]});n[2]=r;var o=document.getElementsByTagName("head")[0],a=document.createElement("script");a.type="text/javascript",a.charset="utf-8",a.async=!0,a.timeout=12e4,c.nc&&a.setAttribute("nonce",c.nc),a.src=c.p+"static/js/"+e+"."+{0:"fbcd72b6763971b2e095",1:"9c80a603c36dd26957a5",2:"0eb60ce6ea8f71c76763",3:"6292429c596b1afc35f1",4:"27ee0c68dd4665a93daa",5:"3cfc1bd67732b39bb6c4",6:"27cf835cb136ecd6c9a7",7:"be8331e079d540e2a26a",8:"bea40bc46f41e75c4b09",9:"03ab6577eb373d78b634",10:"6e2590486f4511d5805c",11:"dcd8693f198edd050b02",12:"8c82b4d027ef5450222f",13:"36d11021e39ada947c13",16:"3e291e5d1ec489dca048",17:"9c5c2133d733d9b404af",18:"6329847bbb6f5fa4e65d",19:"6d2fe2ea128c643a5a05",20:"888b122f16429428f79f",21:"240b1c14f5120dd82b5e",22:"4c54017d9388ca66e78b"}[e]+".js";var f=setTimeout(d,12e4);function d(){a.onerror=a.onload=null,clearTimeout(f);var n=t[e];0!==n&&(n&&n[1](new Error("Loading chunk "+e+" failed.")),t[e]=void 0)}return a.onerror=a.onload=d,o.appendChild(a),r},c.m=e,c.c=r,c.d=function(e,n,r){c.o(e,n)||Object.defineProperty(e,n,{configurable:!1,enumerable:!0,get:r})},c.n=function(e){var n=e&&e.__esModule?function(){return e.default}:function(){return e};return c.d(n,"a",n),n},c.o=function(e,n){return Object.prototype.hasOwnProperty.call(e,n)},c.p="/",c.oe=function(e){throw console.error(e),e}}([]);

```

在HaE新版中，该规则支持对这类格式的文本信息进行格式化输出，如图所示。你只需要Ctrl/Command+A全选Ctrl/Command+C复制，选择一条JS文件请求放入Intruder，以路径作为变量，跑一遍复制的结果，就可以发现意外之喜（更多信息泄露、接口及其他信息）。





Target:

```
1 GET /static/js/$vue.min$.js HTTP/1.1
2 Host:
3 Accept-Language: zh-CN
4 User-Agent: Mozilla/5.0 (Windows NT 10.0;
5 Accept: */*
6 Referer:
7 Accept-Encoding: gzip, deflate, br
8 Connection: keep-alive
9
10
```

## Payload settings [Simple list]

This payload type lets you configure a simple list of strings that are used

Paste	19.6d2fe2ea128c643a5a05
Load ...	8.bea40bc46f41e75c4b09
Remove	18.6329847bbb6f5fa4e65d
Clear	16.3e291e5d1ec489dca048
Deduplicate	5.3cfc1bd67732b39bb6c4
	11.dcd8693f198edd050b02
	0.fbcd72b6763971b2e095
	4.27ee0c68dd4665a93daa
Add	<input type="text" value="Enter a new item"/>
Add from list ...	<input type="text" value=""/>

Request	Payload	Status code	Comment
13	10.6e2590486f4511d5805c	200	Sensitive Field (1), Router Push (1)
6	11.dcd8693f198edd050b02	200	Sensitive Field (1)
20	17.9c5c2133d733d9b404af	200	Linkfinder (8), Username Field (1), Router Push (1)
14	6.27cf835cb136ecd6c9a7	200	Linkfinder (4), Username Field (1), Sensitive Field (3), Password Field (1), Router Push (1)
2	8.bea40bc46f41e75c4b09	200	Linkfinder (4), Username Field (1), Sensitive Field (1), Password Field (2), Router Push (1)
16	7.be8331e079d540e2a26a	200	Linkfinder (3), Username Field (1), Router Push (1)
12	3.6292429c596b1afc35f1	200	Linkfinder (3), Router Push (1)
17	9.03ab6577eb373d78b634	200	Linkfinder (2), Username Field (2), Password Field (2), Router Push (1)
4	16.3e291e5d1ec489dca048	200	Linkfinder (1)
0		200	All URL (3), Linkfinder (4), Sensitive Field (4)
3	18.6329847bbb6f5fa4e65d	200	All URL (3), Linkfinder (13)
8	4.27ee0c68dd4665a93daa	200	All URL (1), Linkfinder (4), Username Field (1), Password Field (1), Router Push (1)
18	2.0eb60ce6ea8f71c76763	200	All URL (1), Linkfinder (3), Router Push (1)
15	1.9c80a603c36dd26957a5	200	All URL (1), Linkfinder (1), Sensitive Field (3), Password Field (2)
1	19.6d2fe2ea128c643a5a05	200	All URL (1), Email (1), Linkfinder (6), Password Field (1), Router Push (1)
5	5.3cfc1bd67732b39bb6c4	200	All URL (1), Email (1), Linkfinder (4), Router Push (1)
7	0.fbcd72b6763971b2e095	200	All URL (1), Email (1)

## Request URI

Request URI可以用来发现未授权漏洞，比如一个后台，你只需要登录进去点击所有的功能，然后在HaE里把Request URI提取的结果复制一下，不帶

凭证的跑一遍，就可以发现未授权了。

Edit Rule

?

Name:

Request URI

F-Regex:

((?!.\*\.js(?:.\*)?\$(.\*?[^.js\$]))

S-Regex:

Format:

{0}

Scope:

request line

Engine:

nfa

Color:

gray

Sensitive:

false

否(N)

是(Y)

Ueditor

Ueditor用于匹配页面中是否引用了Ueditor编辑器的JS文件。

找到引用的JS信息，访问之后就会发现有对应的Ueditor编辑器的上传路径（有时候业务中可能没有功能点）。接下来开始你的测试，迎接你的

RCE、SSRF等等。

Response

PrettyRawHexRenderMarkInfo

Ueditor (1)Linkfinder (29)

#	Information
1	ueditor.config.js

Request

PrettyRawHex

1GET / /ueditor.config.js HTTP/1.1

2

3

Response

PrettyRawHexRenderMarkInfo

All URL (2)Sensitive Field (1)Linkfinder (8)

#	Information ^
7	/myProject/ueditor/
4	/themes/iframe.css
2	/xxxx/xxxx/
6	1.5
5	http://bs.baidu.com/listicon/
3	net/controller.ashx
1	third-party/codemirror/codemirror.css
8	third-party/codemirror/codemirror.js

## URL Scheme

URL Scheme用于匹配非HTTP/HTTPS开头的URL（自定义伪协议的URL）。

这个主要用于在返回包中发现客户端（PC、安卓、iOS）注册的伪协议URL，一般放在Web业务功能点上的都是比较完整的，有路径、有参数。

Edit Rule

?

Name:

URL Schemes

F-Regex:

l{1,20}://[-A-Za-z0-9+&@#/%?~=~\_!|:.,;]+[-A-Za-z0-9+&@#/%=~\_!|])

S-Regex:

Format:

{0}

Scope:

response body

Engine:

nfa

Color:

yellow

Sensitive:

false

否(N)

是(Y)

找到它的意义是，可以直接发现高危漏洞。如发现一个伪协议URL为：xx://xx?url=，它是某安卓APP的，我们通过HaE发现之后，在安卓浏览器打开发现参数url所指向的地址会调起APP内置Webview去访问（携带凭证），直接获得1 Click用户凭证劫持。再深入拓展，可以逆向APP看下JSBridge的逻辑，1Click RCE也不是没有可能。

思路参考文章：<https://gh0st.cn/archives/2018-12-08/1>

# Password Field、Username Field

Password Field、Username Field用于发现返回包中的用户名、密码字段信息。

Edit Rule

?

Name: Username Field

F-Regex: |update)((d|r)(by|on|at))|(creator)))([w]{1,10})(['"])(:|=)([x](['"])(.\*?)(['"])(,))

S-Regex:

Format: {0}

Scope: response body

Engine: nfa

Color: green

Sensitive: false

否(N)

是(Y)

Edit Rule

?

Name: Password Field

F-Regex: ,10})([p](ass|wd|asswd|assword)))([w]{1,10})(['"])(:|=)([x](['"])(.\*?)(['"])(,))

S-Regex:

Format: {0}

Scope: response body

Engine: nfa

Color: yellow

Sensitive: false

否(N)

是(Y)

使用这个规则经常能在JS文件中发现测试用户和密码， 或者发现通用的初始化密码便于爆破。曾经用它直接进了某后台，然后在后台挖掘，继而RCE。

```
if (rootDomain !== curDomain) {
  window.MQTT = {
    host: '██████████',
    port: 8084,
    ssl: true,
    userName: 'admin',
    password: 'e██████████9V'
  }
}
```

Response

PrettyRawHexMarkInfo


Username FieldPassword Field

Information

userName: 'admin'

# Router Push

Router Push用于匹配前端注册的路由信息。



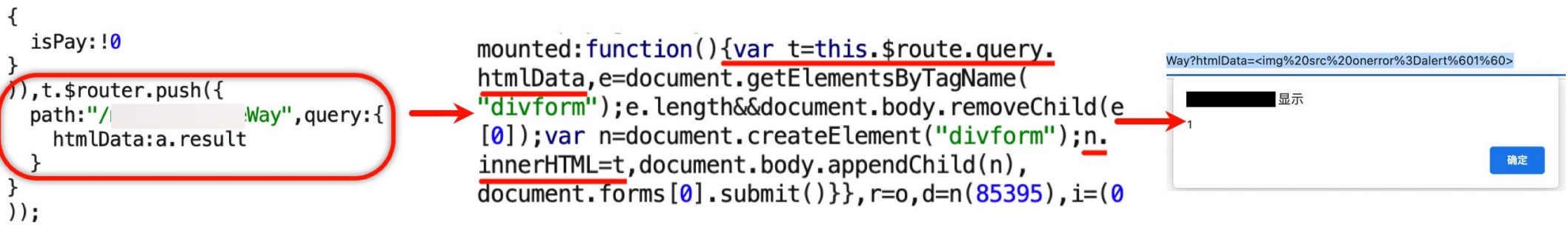
Name:	Router Push
F-Regex:	(\\${router}\.push)
S-Regex:	
Format:	{0}
Scope:	response body
Engine:	dfa
Color:	magenta
Sensitive:	false

否(N)

是(Y)

通常使用它可以来发现一些前端路由的传入参数，如图所示，通过HaE定位到具体逻辑，找到传入参数和处理逻辑成功发现DOM型XSS漏洞。





## ALL URL

ALL URL用于匹配返回包中的所有URL链接。

## Edit Rule

?

Name:

All URL

F-Regex:

[a-zA-Z0-9+&@#/%?=~\_|!:,.;\u4E00-\u9FFF]+[-A-Za-z0-9+&@#/%=~\_|])

S-Regex:

Format:

{0}

Scope:

response body

▼

Engine:

nfa

▼

Color:

gray

▼

Sensitive:

true

▼

否(N)

是(Y)

这个功能看似毫无意义但实际上却帮助我们发现了数十个高危漏洞。其主要围绕发现新资产（路径、地址）、凭证链接（Token、密码）。如图所示，就在一个JS文件中泄漏了Token凭证链接，访问即可接管账户。

< JSON Web Token (1) HTML Notes (2) Cloud Key (2) <u>URL (265)</u> Create Script (14... > ▼	
#	Information ^
214	111

214 http  
212 http  
206 http  
210 http  
219 http  
221 http  
207 http  
213 http  
216 http  
209 http  
254 http  
220 http  
211 http  
217 http  
252 http  
255 http  
218 http  
253 http  
205 http  
208 http



## 聚合查询

HaE 2.X版本开始支持聚合查询Databoard面板，这个是最喜欢的功能，也是我觉得HaE最不可能被替代的竞争能力。

每次做完测试，我都会回顾一下Databoard来看看有没有什么遗漏的地方，经常又能让我再发现几个高危漏洞。

我个人比较直接喜欢使用 \* 号来查询所有东西，这样看着很爽。然后双击左边的数据就会在右边给你符合数据的HTTP请求响应。

HaE

Highlighter and Extractor - Empower ethical hacker for efficient operations.

Rules

Databoard

Config

Host: \*

Action

< le Number (6) Create Script (4) All URL (328) Chinese IDCard (5) > v

#	Information
1	
2	
3	
4	
5	

Method	URL	Comment	Status	Length	Color
GET		All URL (4), Linkfinder (11), Username ...	200	30619	green
POST		Request URI (1)	200	477	gray
GET		All URL (4), Authorization Header (1)	200	4898	yellow
POST		Request URI (1)	200	478	gray
GET		All URL (12), Email (1), Linkfinder (22), ...	200	60196	orang
GET		Linkfinder (1)	200	14332	gray
GET		All URL (12), Email (1), Linkfinder (22), ...	200	60226	orang
GET		All URL (4), Linkfinder (16), URL Sche...	200	151835	orang
GET		All URL (11), Linkfinder (15)	200	25434	mage
GET		All URL (1)	200	3431	gray
GET		Sensitive Field (1)	200	55258	yellow
GET		Sensitive Field (2)	200	18814	yellow
GET		Linkfinder (4)	200	96639	gray
POST		Request URI (1), DoS Paramters (1)	200	630	cyan
GET		All URL (6), Linkfinder (19), URL Sche...	200	152819	orang
POST		Request URI (1)	200	5134	gray
POST		Request URI (1), URL As A Value (1)	200	5963	cyan
GET		Request URI (1)	200	5115	gray
POST		Request URI (1)	200	5117	gray
POST		All URL (3), Linkfinder (7), Request UR...	200	8836	mage

Request

Response

Pretty

Raw

Hex

1

Settings

Search

Second search

AI Empowered

?

⚙

⬅

➡

Search

🔍

0 matches

OK

## 二次搜索

HaE 3.3.3版本开始支持二次检索了，可以通过如下正则：

```
1 | remove|update|submit|save|put|clear|del|init|add|reset|restart|clean
```

来过滤一些涉及敏感关键词的数据，如下图所示，配合HaE的 Reverse search ，来过滤掉敏感操作的请求接口。过滤完成之后再通过二次检索功能基于上一结果再进行过滤。

Create Script (1402)

All URL (34)

Email (4)

Linkfinder (1508)

Request URI ... > ▾

#	Information ^
1044	/api/user/login
945	/api/user/loginUserName
85	/api/user/logout

☒ Reverse search

Settings

remove|update|submit|save|pi

/api/

AI Empowered