Vulnerability Report

e-commune vulnerability & penetration test assessment

**Ebaneck Andui Claude ATUH**

**MSc. Computer Science**

**EPITA, Paris-France**

**Bismark Atta FRIMPONG**

**MSc. Computer Science**

**EPITA, Paris-France**

**Table of Contents**

[Synthesis 2](#_Toc515134089)

[Category of Vulnerabilities 2](#_Toc515134090)

[General Recommendation 2](#_Toc515134091)

[Summary 4](#_Toc515134092)

[Vulnerability Sheet 5](#_Toc515134093)

[First Vulnerability: Remote Exploit via SQL Injection 5](#_Toc515134094)

[Description: 5](#_Toc515134095)

[Exploitation 5](#_Toc515134096)

[Recommendation 8](#_Toc515134097)

[Second Vulnerability: Compromise of Website Admin Interface 10](#_Toc515134098)

[Description 10](#_Toc515134099)

[Exploitation 10](#_Toc515134100)

[Recommendation: 13](#_Toc515134101)

# Synthesis

e-Commune is a web application that has been developed to be deployed as an e-government framework for each town of France. The application is separated into different profiles with privileges as below:

* Visitor: no privileges, all visitors without authentication are part of this profile
* Agent: town's agent that can communicate by in-mail with citizens’ resident of the town, manage their documents.
* Chef: town's resident that can communicate by in-mail with agent, pay his bills, etc.
* Admin: websites administrator that can add pages, edit them, and much more.

# Category of Vulnerabilities

|  |  |  |
| --- | --- | --- |
| **Number** | **Vulnerabilities** | **Count** |
| 1 | SQL Injection | 01 |
| 2 | Comprise of the Website Admin Interface | 01 |

Team **BisCla** was contracted to conduct a penetration test in order to determine its exposure to a targeted attack. All activities were conducted in a manner that simulated a malicious actor engaged in a targeted attack against e-commune.org website with the goals of:

* Identifying if a remote attacker could penetrate e-Commune’s defences.
* Determining the impact of a security breach on.
* Confidentiality of the website’s private data.

Efforts were placed on the identification and exploitation of security weaknesses that could allow a remote attacker to gain unauthorized access to website data regardless of the source code. The attacks were conducted with the level of access that a general Internet user would have.

Our vulnerability assessment was carried out from the 23 - 25 of May 2018 and we have performed this audit in accordance to the generally accepted auditing standards defined by our Supervisor (Teacher).

# General Recommendation

The underlying problems we have spotted indicates the need of a well-planned and thoroughly implemented security architecture for the E-commune website. One has to be reminded that all the vulnerabilities found might turn out to becoming a privileged escalation or remote code execution when it is discovered and exploited.

We set out in our research to test whether the E-commune website application is resistant to casual attack and have proven that it is vulnerable to current attacks. As previously discussed many of the vulnerabilities found will need to be suﬃciently patched.

# Summary

Initial exploration/scan of the e-Commune web application resulted in the discovery of the underlined Database used (MySQL). This misconfigured database server allowed remote alteration of user information. The results provided us with a listing of the specific database name and version to target for this assessment. An examination of these information revealed the specific tables and columns present. After a few tries using information present on E-commune’s website, we were able to gain access to the database and uncover user password present within the database.

An examination of the files and folders on the server revealed the exposure of sensitive information. This compromise was escalated thanks to the presence of a database configuration details present in one of the files within this directory. A removal of this file from the server will solve this issue.

# Vulnerability Sheet

# 

## First Vulnerability: Remote Exploit via SQL Injection

Business Risk : Critical

Ease of Exploitation : Easy

Ease of Correction : Medium (have to re-write all SQL statements)

Upon having a closer look at the sources used for the web application login, it was found that the code is plagued by several SQL injection vulnerabilities. Those would enable an attacker to get entry to the admin area, even if the password would have been changed earlier. The attack can be exploited **quite easily** by simply entering the following strings into the form-fields for username and password, respectively.

This issue can addressed within few days hence easiness of correction is of medium scale as per the E-commune application.

### Description:

As the name suggests, an SQL injection vulnerability allows an attacker to inject malicious input into an SQL statement.

### Exploitation

We took a closer look at all the web pages within the E-commune website and we noticed how the URLs are being defined; e.g. *http://www.e-commune.org/index.php?ncat=52*

*Adding any arbitrary information after the url resulted into an error message with a* ***leak*** *of the database used, and web server running the website with the necessary parameters which needed to be passed.*

*Warning: mysql\_fetch\_array(): supplied argument is not a valid MySQL result resource in*

*Sample url for error return:*

[*http://www.e-commune.org/index.php?cat=citoyen&ncat=52*](http://www.e-commune.org/index.php?cat=citoyen&ncat=52) *UNION SELECT 1, 2, 3*

With the underlying database and web server identified, we attempted to conduct and confirm the database server on the website. We found that e-commune.org was using MYSQL database and was vulnerable to a full SQL injection. This provided us with a listing of the database used, the name of the database, tables within the database, columns, which could be used to further target the organization. (Figure 1) reveals some sensitive information about: Database type/kind, web server type and version. This information can provide attackers with detailed information about the capabilities of the website. It can potentially also leak information about the users contained with the website.

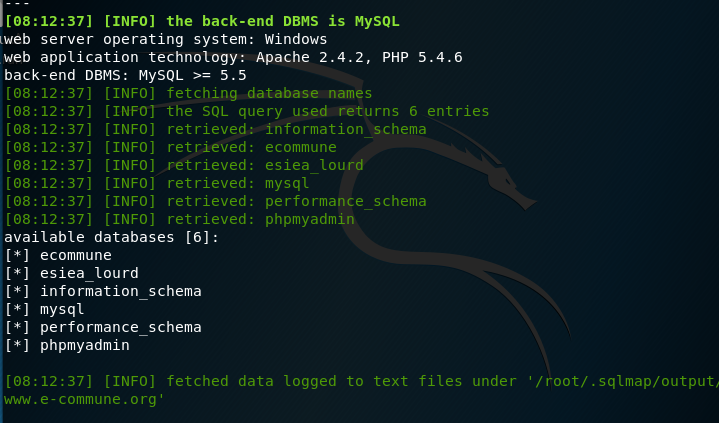


Figure 1: Leak of information on website technologies used and databases available.

With the database servers and database names identified, we attempted to list the tables and columns within the database. We found that E-commune has the following tables and columns. This provided us with a listing of tables and associated columns, which could be used to further target the website. (Figure 2):

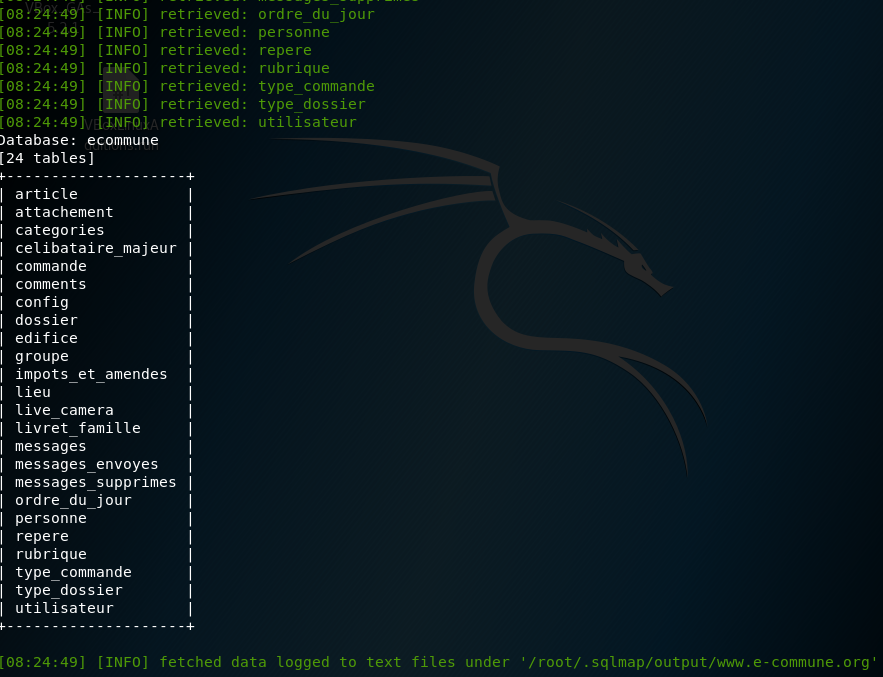


Figure 2: Listing of all the tables present within the website database (e-commune)

Based on this gathered information, we went ahead to look at the contents of all the tables and especially the table named “**utilisateurs**” which provided us with most sensitive information.

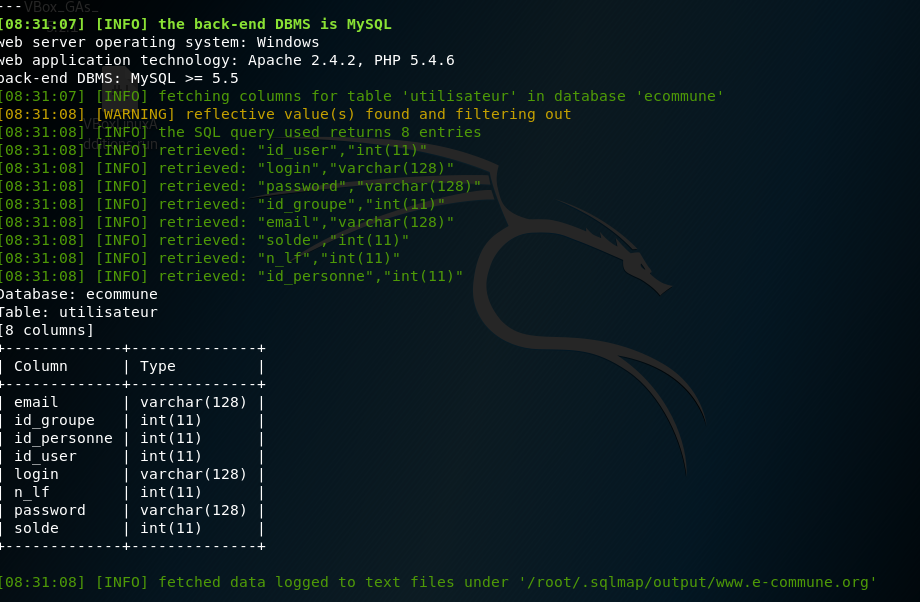


Figure 3: Display of the columns present with the user table (sensitive info)

With this information, it was only logical for us to try query the login and password column in other to obtain login details for the admin user.

We successfully obtain the list of users present in the login table in the figure below.



Figure 4: Query listing all the users present with the users’ column

We also query the password column and we obtain password but these password are all stored as hashes.

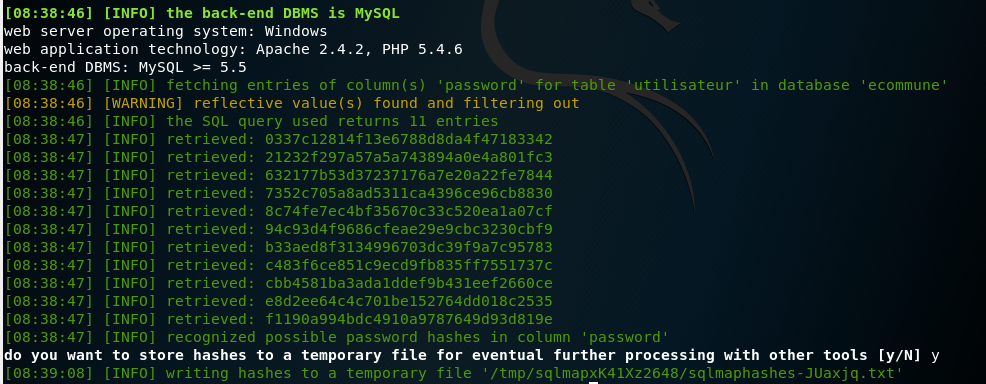


Figure 5: Extraction of user stored password present in the password column

Because the extracted passwords were not in plain text, we used a brute force attack with a dictionary file to attack the passwords stored as hashes. The outcome was even great.

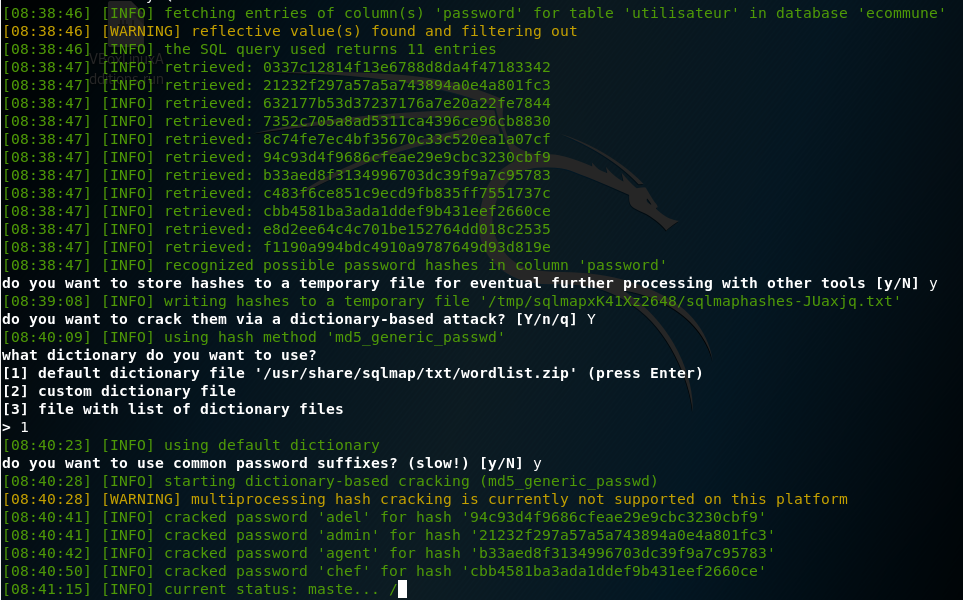


Figure 6: Successful Dictionary attack to crack the md5 password hashes

### Recommendation

A quick fix to solve the issue of MySQL injection will be to allow the database to be able to distinguish between the SQL command and data input by a user. This can be accomplished by using parameterized queries. This queries will force Developers to define the SQL query beforehand, and use placeholders for the user-provided variables within the query.

You can then pass-in each parameter to the query after the SQL statement is defined. If SQL commands are input by an attacker, the parameterized query would treat these as untrusted input, and the injected SQL commands will never get executed.

## Second Vulnerability: Compromise of Website Admin Interface

Business Risk : High

Ease of Exploitation : Easy

Ease of Correction : Medium

### Description

This section shows a huge exposure of sensitive technical information which in business terms is of high risk since this can lead to potential exposure of user credentials of current users of the website. Such exposures can potentially lead to closure of the entire business since customers/users will lose trust in the entire system. This vulnerability can be corrected.

### Exploitation

The e-Commune webserver was found to be running an Apache web server in the previous exploitation. Accessing the root URL of the website resulted in the display of an index page. We next conducted a quick enumeration scan of the system with the intention of looking for common directories and files (Figure 7 below).

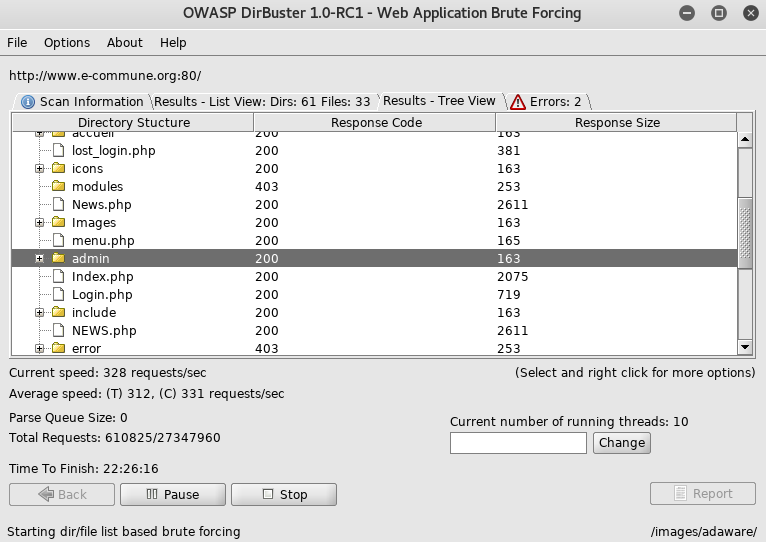


Figure 7: website directory and file discovery

The scan results revealed that along with common Apache default files, we identified an **“/admin”** directory that was accessible without authentication.

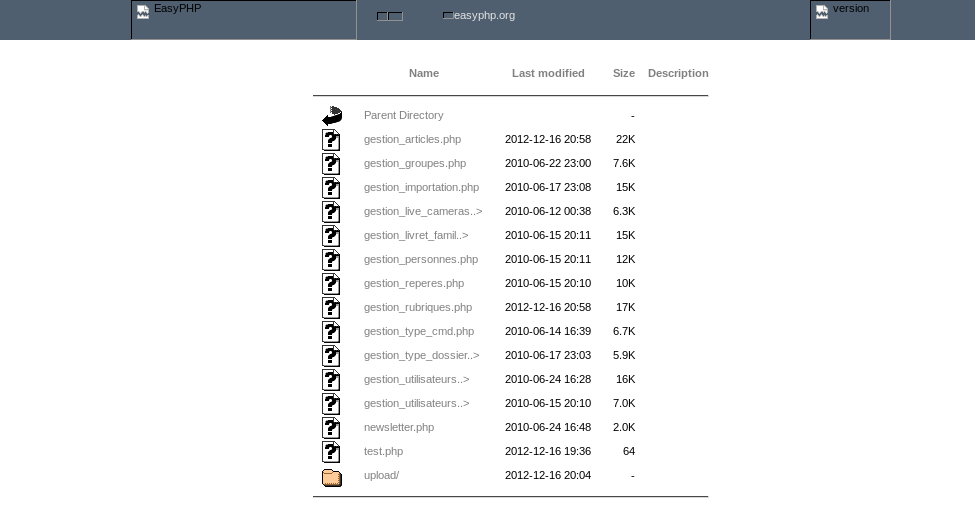


Figure 8: Access to the “admin” directory is not protected

We went through the files in the /admin directory and nothing useful was found. So we started searching the subdirectory called “/admin/upload” and finally we came across a file with image extension but the contents were different.

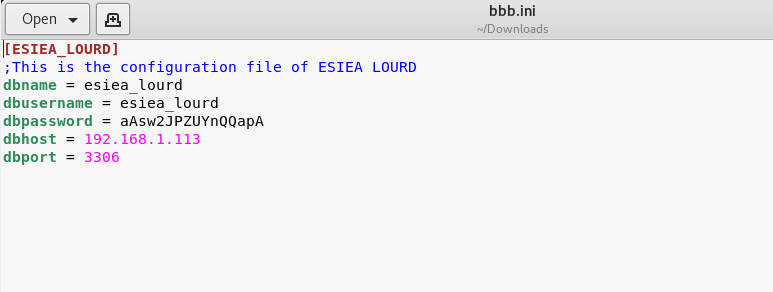


Figure 9: File contains database configuration but was named differently.

We access to this database configuration files, we tried to remotely connect to the database and the database server allowed us authentication using the above username and password.

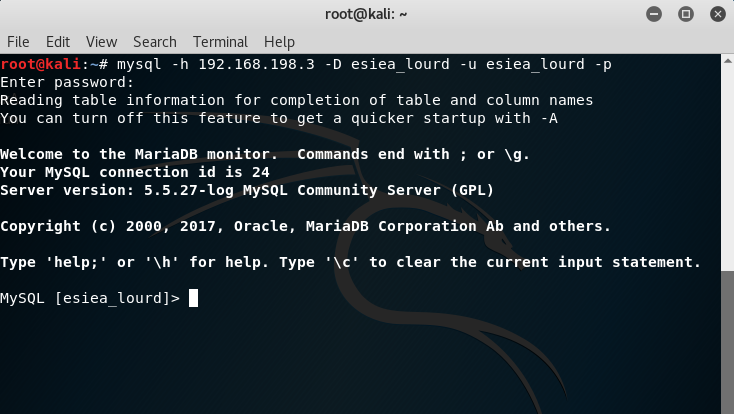


Figure 10: Remote connection established using the found username and password.

Upon searching through the tables and columns present in the database, We found out the following sensitive information. This could potential be the login information for the admin panel into the database dashboard.

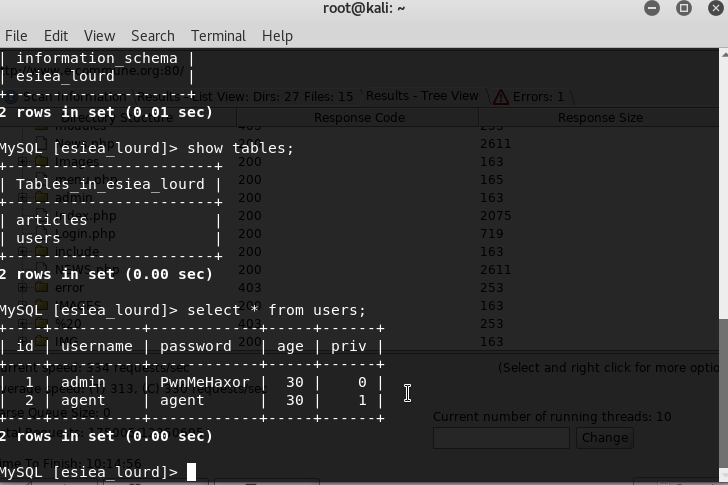


Figure 11: Show database columns and tables while listing contents in user table.

The information gotten above could potentially be the login details into the admin panel for phpMyAdmin found on the server.

### Recommendation:

Firstly, the “/admin” directory should be password protected to restrict access to this directory and hence minimise risk of exposing sensitive data to the public.

Secondly, the database configuration information which is even more sensitive should be remove completely from the directory or server. This is technical information and it should not be disclosed.