Vulnerability Report

THICK CLIENT VULNERABILITY & PENETRATION TEST ASSESSMENT

Ebaneck Andui Claude Atuh
MSc. Computer Science
EPITA,France

Table of Contents

Synthesis

This report discusses the critical vulnerabilities and corresponding risks in a two-tier thick client application along with the measures to mitigate risks. Thick client is defined as an application client that processes data in addition to rendering. An example of thick client application can be a Visual Basic, JAVA or VB.NET application that communicates with a database.

The risks observed in thick client applications generally include information disclosure, unauthorized access, authentication bypass, application crash, unauthorized execution of high privilege transactions or privilege escalation etc.

This report is an audit of a command line application which is built for users to be able to perform the following functions.

- Search for articles (shoes, sweets, hats etc).
- Find products present
- Check for connectivity and much more

Category of Vulnerabilities

Number	**Vulnerabilities**	**Count**
1	Critical data in files (config file has secret information)	01
2	SQL Injection	01

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

Our vulnerability assessment was carried out from the **14 - 16 of June 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 Thick Client.

We have discussed a few critical vulnerabilities associated with a thick client application. To address these issues secure application architecture should be designed. The following security best practices can be considered during development.

- Application UI to Application Server and Application Server to Database server traffic should be protected to maintain confidentiality and integrity. Confidentiality can be maintained using encryption algorithm like DES or 3DES and integrity by SHA3.
- Application should validate all user inputs for length, special characters & code.
- Application should have provision to enforce proper authorization level to users on a need to know basis.
- Adequate audit and logging feature should be present and enabled in the application to log all transaction events including login attempts.
- Application should not store sensitive information like user password in computer memory, files, registry or database in clear text format. It should rather be stored in one-way hash format. Critical file and registry permission must be secured. Application should clear the data in the memory after its usage.
- Application should have the provision to enforce strong password policy to the users.
- Application must use strong authentication technique. In this 3-tier structure all the user credentials can be verified at application server itself. It must not pass any credential like actual password to UI.
- Application should uniquely identify and maintain sessions. Session IDs used should be random and unbreakable.
- Application should handle the errors without disclosing critical system information.
- Database should not run on a default port. A non-guessable port should be used to connect the database.
- Database, application server and OS should be configured with secure setting and updated with the latest software or patch levels.
- Database should accept the connections only from the application server.

Summary

The risks observed in thick client applications generally include information disclosure, unauthorized access, authentication bypass, application crash, unauthorized execution of high privilege transactions or privilege escalation.

In this report, we shall discuss in detail few of the critical vulnerabilities we discovered in thick client applications including unvalidated input, weak authentication method, sensitive data in memory, critical data in files & registry and impersonating (stealing account of) a high privilege user.

Initial exploration resulted in the discovery of the underlined Database used (MySQL). 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, 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. Encrypting the contents of this file on the server can potentially solve this issue.

Vulnerability Sheet

First Vulnerability: Critical data in files & registry

Business Risk	:	Critical
Ease of Exploitation	:	Easy
Ease of Correction	:	Medium (store connection username and password as hashes)

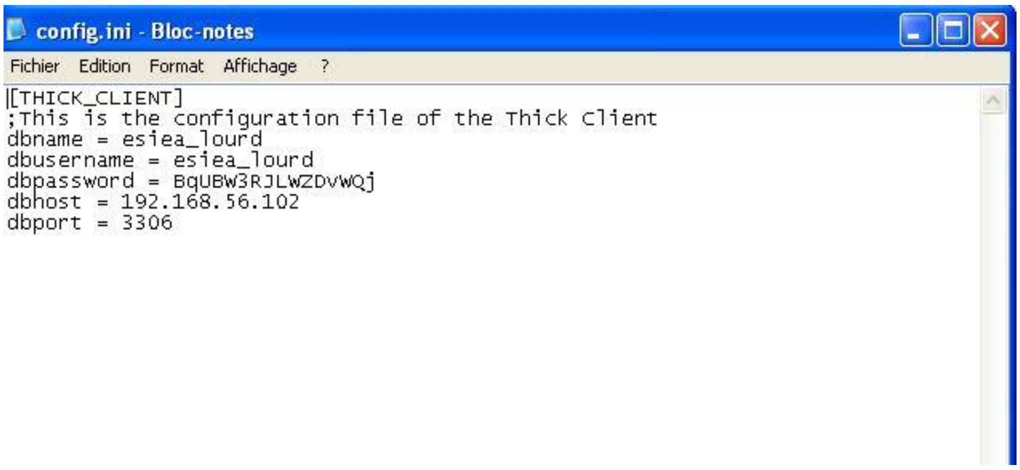
Application stores critical information in files and registry. These files generally contain cryptographic keys, username, password of application and database. If these files are not secured, a malicious user can delete, modify or read data from the files. Assume that the application fetches the data from a file to connect to a remote database using service name, IP address, port name, user ID and password as our case. Now, if the permission on this file is not secured, anybody can access the file to get user ID and password of the database. Again only an access control list (ACL) cannot prevent a clear text stored password from disclosure. A malicious user can use username and password to connect the database directly with admin privileges.

Description:

An attacker can retrieve user passwords and cryptographic keys from files and registry. Attacker can use the same username and password, gain access to the application and access sensitive data in the database. – Database credential in file

Exploitation

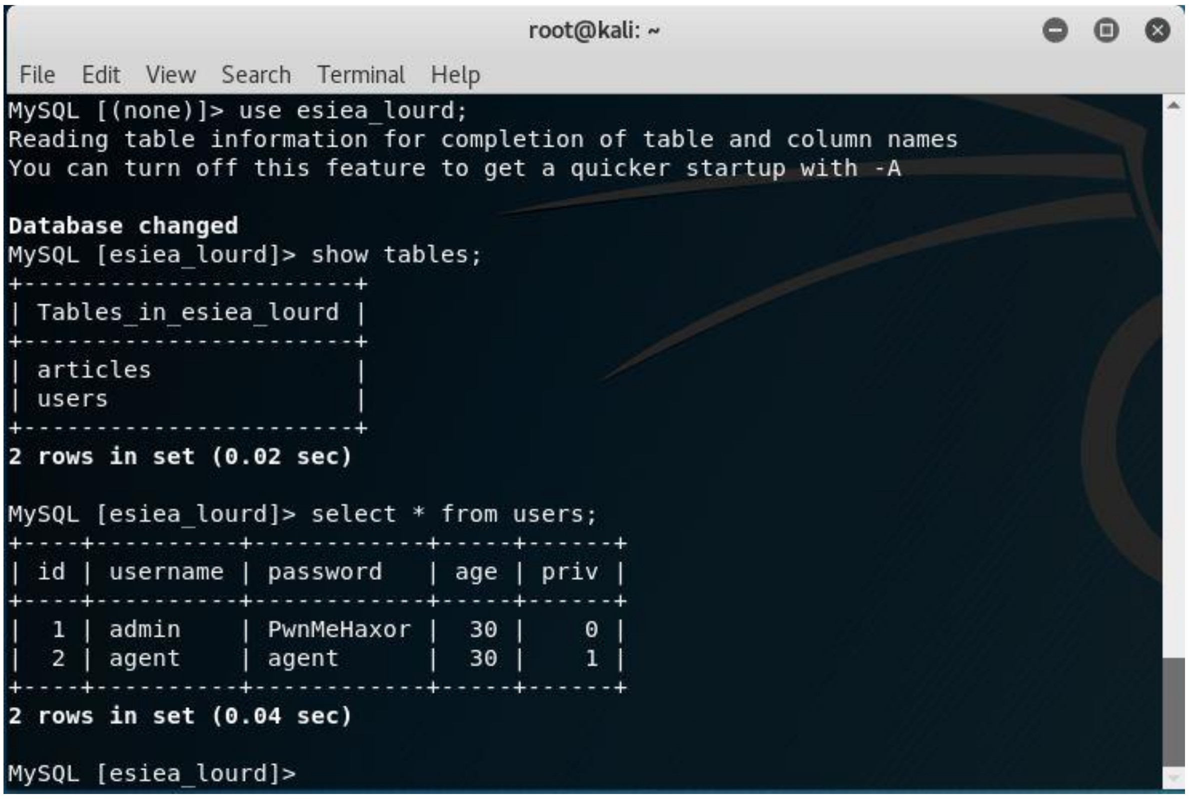
An example of database credential is stored in a file in clear text format is shown below.



```
[THICK_CLIENT]
;This is the configuration file of the Thick Client
dbname = esiea_lourd
dbusername = esiea_lourd
dbpassword = BqUBW3RJLWZDVWQj
dbhost = 192.168.56.102
dbport = 3306
```

Figure 1: Database credential stored in clear text format

With the database servers and database names identified, we attempted to list the tables and columns within the database. We found the following tables and columns. This provided us with a listing of tables and associated columns, which could be used to further target the client and eventually obtain the admin login credentials. (Figure 2):



```
MySQL [(none)]> use esiea_lourd;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MySQL [esiea_lourd]> show tables;
+-----------------------+
| Tables_in_esiea_lourd |
+-----------------------+
| articles              |
| users                 |
+-----------------------+
2 rows in set (0.02 sec)

MySQL [esiea_lourd]> select * from users;
+----+----------+------------+-----+------+
| id | username | password   | age | priv |
+----+----------+------------+-----+------+
|  1 | admin    | PwnMeHaxor |  30 |    0 |
|  2 | agent    | agent      |  30 |    1 |
+----+----------+------------+-----+------+
2 rows in set (0.04 sec)

MySQL [esiea_lourd]>
```

Figure 2: Using information in config file to connect to database remotely

Recommendations

There are various solutions for these problems:

- Secure the permission on application program files. Give full control permission only to application service account and administrator account. Other users and groups are restricted with no permissions to read this file.

- Restrict the permission on user and password file (if present). Store the password in one-way hash format (e.g. SHA3) in the file.

- Restrict the permission on database credential file. If database username and password are stored in this file, then store them in encrypted or one-way hashed format (e.g. SHA3).

Second Vulnerability: SQL Injection

Business Risk	:	High
Ease of Exploitation	:	Easy
Ease of Correction	:	Medium

Description

Unvalidated Input: User requests are not validated before being used by the application. Attackers can use this flaw to launch SQL injection attack. SQL injection attack is possible when application does not validate user input for special characters and codes. An attacker can exploit this vulnerability by supplying specially crafted SQL statement to the application.

Exploitation

This attack can result in authentication bypass, viewing unauthorized data, application malfunction, data deletion or malicious command injection including database shutdown.

The application uses the following SQL query to validate user credentials.

```
SELECT * FROM USERS WHERE USERNAME=' ' AND PASSWORD=' ';
```

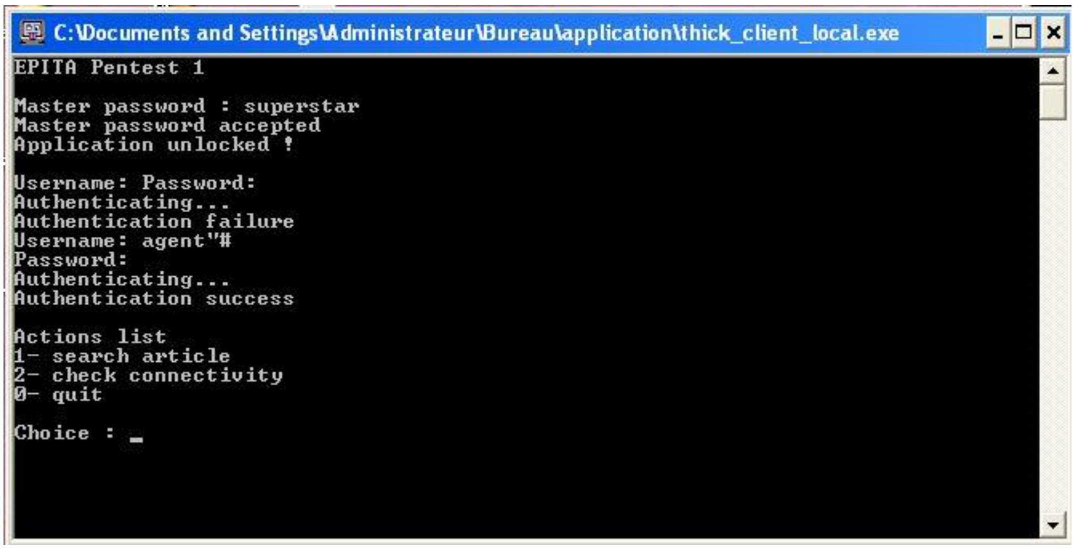
Application will authenticate a user if the database returns a row of data against this query.
A valid user supplies the following credential to the application: USERNAME: admin PASSWORD: agent Then the query will be:

```
SELECT * FROM USERS WHERE USERNAME='admin' AND
PASSWORD='admin';
```

The attacker knows the USERNAME i.e. ‘admin’ and he crafts the query:
USERNAME: admin”#

This query will comment out password checking logic and database will execute only uncommented query portion. This will return entire row of user ADMIN. Application assumes it as valid user and will grant the access.

The above exploitation can lead to **Database Shutdown using Command Injection**



C:\Documents and Settings\Administrateur\Bureau\application\thick_client_local.exe

```
EPITA Pentest 1

Master password : superstar
Master password accepted
Application unlocked !

Username: Password:
Authenticating...
Authentication failure
Username: agent"#
Password:
Authenticating...
Authentication success

Actions list
1- search article
2- check connectivity
0- quit

Choice : _
```

Figure 3: Account verification and authentication is bypassed.

The attacker only knows the USERNAME i.e. ADMIN and he crafts the query:

USERNAME: ADMIN"#; SHUTDOWN WITH NO WAIT; --

SELECT * FROM USER WHERE USERNAME ='ADMIN '; SHUTDOWN WITH NO WAIT; -- ' AND PASSWORD=' ';

SELECT * FROM USER WHERE USERNAME ='ADMIN '; SHUTDOWN WITH NO WAIT; -- ' AND PASSWORD=' ';

This query will comment out password checking logic and database will execute only uncommented query portion. After executing the SELECT query database will move to next query and execute the statement i.e. **SHUTDOWN WITH NO WAIT** and will immediately shutdown the database.

Recommendation:

The solution to this problem is input validation. The application should check for any special character and code in the user input. In case any special character like ' or “ (single/double quote) or command code is present; application should ignore the input and throw appropriate error message to the user.