**Team4 Penetration Test Report**

**Team 3**

**About This Document**

Document Information

|  |  |
| --- | --- |
| **Issuing authority** | Team 3 |

Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Verion** | **Date** | **Comment** | **Author** | **Approver** |
| 2.0 | 2021-07-01 | Second Release(English) | Team 3 |  |
| 1.0 | 2021-06-30 | Initial Release(Korean) | Team 3 |  |
|  |  |  |  |  |

Acronyms / Glossary

|  |  |
| --- | --- |
| **Acronym** | **Description** |
| CCTV | Server application, which connected with camera |
| Monitoring System | Client application, which display the image |
| User Register | Auxinaly application, which register new face image |

Table of Contents

[**Executive Summary**](#_67cdizv7fa99) **4**

[Main Features](#_sme5j7kw6w82) 4

[**Target System Overview**](#_f60kw590u9lk) **4**

[Main System](#_tooox2uxzh1m) 4

[Users](#_32ffxpa17g5v) 4

[**Penetration Testing Steps**](#_kimx2ihy5ocu) **4**

[Penetration Processing](#_wjrafwkadai9) 4

[Plan](#_virfm6lhmjni) 5

[Information Gathering](#_oiv60kx8fd5l) 5

[Penetration Execution](#_cdrj6yg7sd6d) 5

[**Results of the Assessment**](#_e0q7xwxrmblm) **10**

[Main Features](#_1553was3i80p) 10

[**Recommendations**](#_m2jx212rpa5d) **10**

[Main Features](#_86wm37h7uyso) 10

[**Conclusion**](#_2ue1q774zgeq) **11**

[Main Features](#_byydqtt2zozo) 11

# **Executive Summary**

This document is a penetration test record of the Tartan Monitoring System made by Team4 “4tential”. Team3 “100% Certified” planned and analyzed the system to penetrate the system, found vulnerabilities, and actually attacked those vulnerabilities. The attack points of various groups were analyzed and the vulnerabilities discovered were described. In fact, some attacks were very effective, and we will check the points to be taken into account when developing and redistributing the system.

# **Target System Overview**

## **Main System**

The software provided by Team4 is as follows.

* CCTV: Camera image processing and face recognition program. It is a server application.
* Monitor System: A program that receives images from the network and displays them. It is a client application.
* User Register: User registration program. It is an auxiliary application of the server side and has no network input/output function.

## **Users**

* Authorized Person: A person who enters the server room and is subject to CCTV footage. To become an Authorized Person, you must apply for user registration to the Monitoring System Manager in advance.
* Security Agent: A person who is looking at the Monitoring System does not do anything other than looking at the screen.
* Monitoring System Manager: A person who registers users. Although the design document only state user registration is specified, in reality, server application operation, maintenance, management, etc. are all done.

# **Penetration Testing Steps**

## **Penetration Processing**

Penetration proceeds as follows.

Plan → Information gathering → Threat modeling → Vulnerability analysis → Exploitation

→ Post exploitation → Reporting

The whole process proceeds as above, in Chapter 3.2 deals with plan

in Chapter 3.3 deals with information gathering, in Chapter 3.4 deals with vulnerability analysis and exploitation, in Chapter 4 deals reporting, and in Chapter 5 deals Recommendation.

## **Plan**

Overall, design analysis, static analysis, and dynamic analysis were performed by each member of Team3, and in some cases, everyone gathered and analyzed together. And we decided to share the vulnerabilities found and directly try to attack some of them.

* Design document analysis
  + Design document/code matching
  + Checking security reflections in design (threat modeling analysis)
  + Security policy analysis
  + Program usage analysis
* Static analysis
  + Static analysis by tool
  + 3rd party open source vulnerability check
  + Security code review
* Dynamic analysis
  + Fuzz Testing: Zuff, AFL
  + Vulnerability Scanner: nmap, other tools
  + Data input/output analysis
  + Exploit
  + Test Case analysis

After exploit, we discuss about vulnerability about it again and record all the things in this document

## **Information Gathering**

In order to find vulnerabilities in commercial programs, we will use the scanner program first after reviewing the manual, but in this project, design documents and source codes are provided, so we proceeded at the same time. The following tools, methods and sites were used.

* Open source known vulnerabilities identification: dependency library, cve site search
* Vulnerability scanner: Nessus, nmap
* Fuzz Testing: Zuff, AFL
* Security focused testing: Encryption, Network data analysis
* Static analysis: RATs, SonarCloud, Code X-ray
* Design/code review: Review all output from Team4

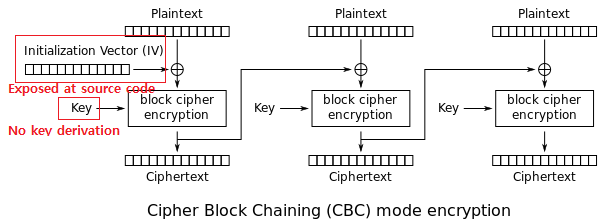
## **Penetration Execution**

Based on the data obtained through information gathering, vulnerabilities and weak points were analyzed, and exploitation was attempted if possible. Some cases are difficult to implement in practice, but there are unstable factors, so we have dealt with them.

* + 1. Common attack point

This is the whole system related part.

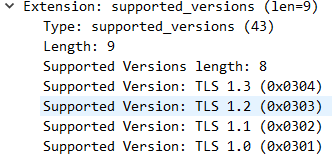
* Encryption Key: Since the AES-CBC-128 algorithm is used for block encryption, it is practically difficult to hack. However, the Initialization Vector used for CBC is exposed in the source code, and PBKDF (password-based key derivation functions, NIST SP800-132) suggested by NIST is not applied. These don't directly lead to data leak, but hackers can reduce the Iteration effect of AES so the attack can be made faster.



|  |
| --- |
| $ strings LgFaceRecDemoTCP\_Jetson\_NanoV2 | grep 1234  1234567887654321 |

The Initialization Vector is also exposed by the strings command at linux shell.

* TLS version control: TLS, which is an encrypted communication, is difficult to intercept data in the middle. However, several mistakes were discovered during the code review process. When using the OpenSSL API, if the previous version of the API is used and the ciphersuite is not restricted, the actual connection is negotiated with one of TLS 1.0, TLS 1.1, TLS 1.2, and TLS 1.3. Each system of the server and client supports all TLS, but fortunately, the actual connection takes precedence over the higher version, so it is connected with TLS 1.3. The security design document specifies that TLS 1.2 is used.



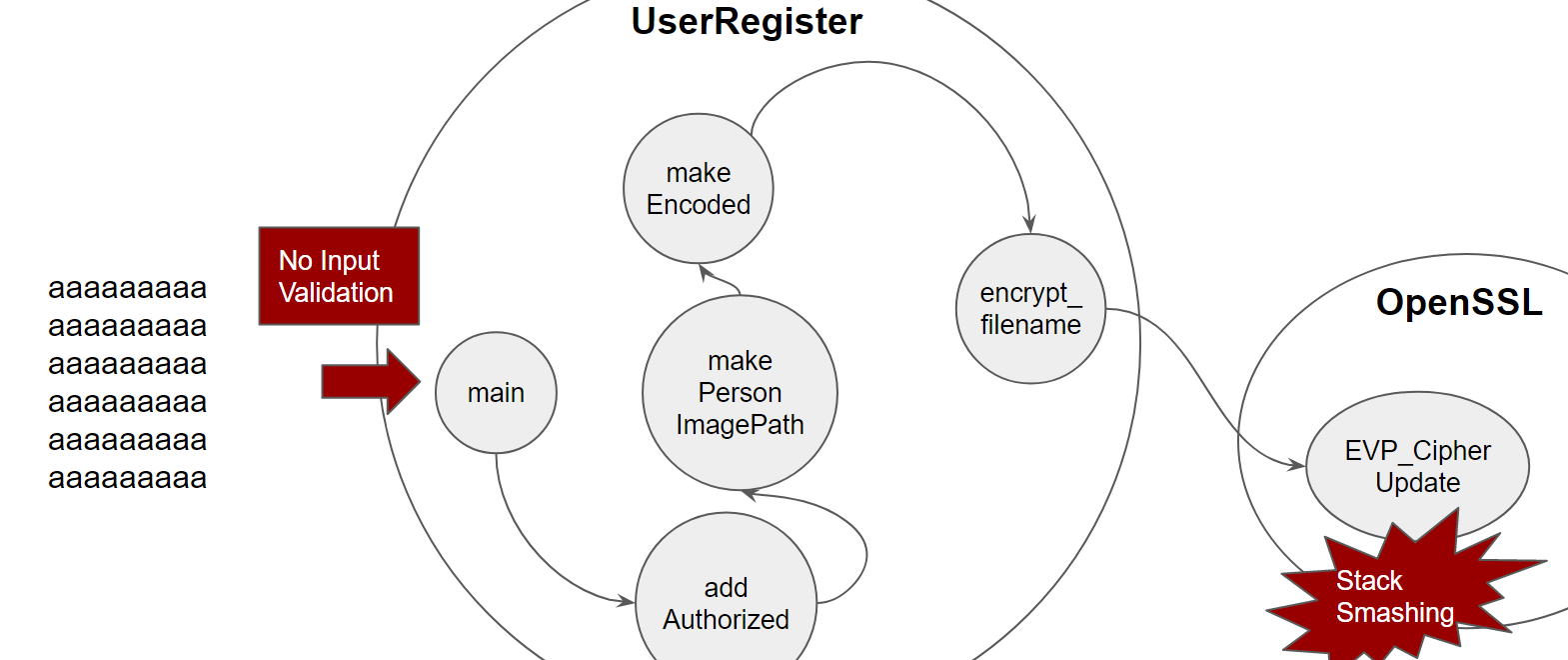
* Deployment process attack: When distributing the program, it is distributed as source code rather than as an executable file, so you have to compile it arbitrarily to create an executable file. This means that when a user uses an executable file, the user cannot know whether it is a legitimate file or not. This is related to the program's attempt to use the RootCA file in a strange location(..\..\Certificate\rootca.crt). Therefore, when distributing the program, the signed executable file or executable file and the hash value that can confirm it must be provided together. In addition, since Release mode is not defined during binary build, it is compiled in Debug mode. In this case, it is not optimized and all debug symbols are exposed, increasing the possibility of disassembler analysis.



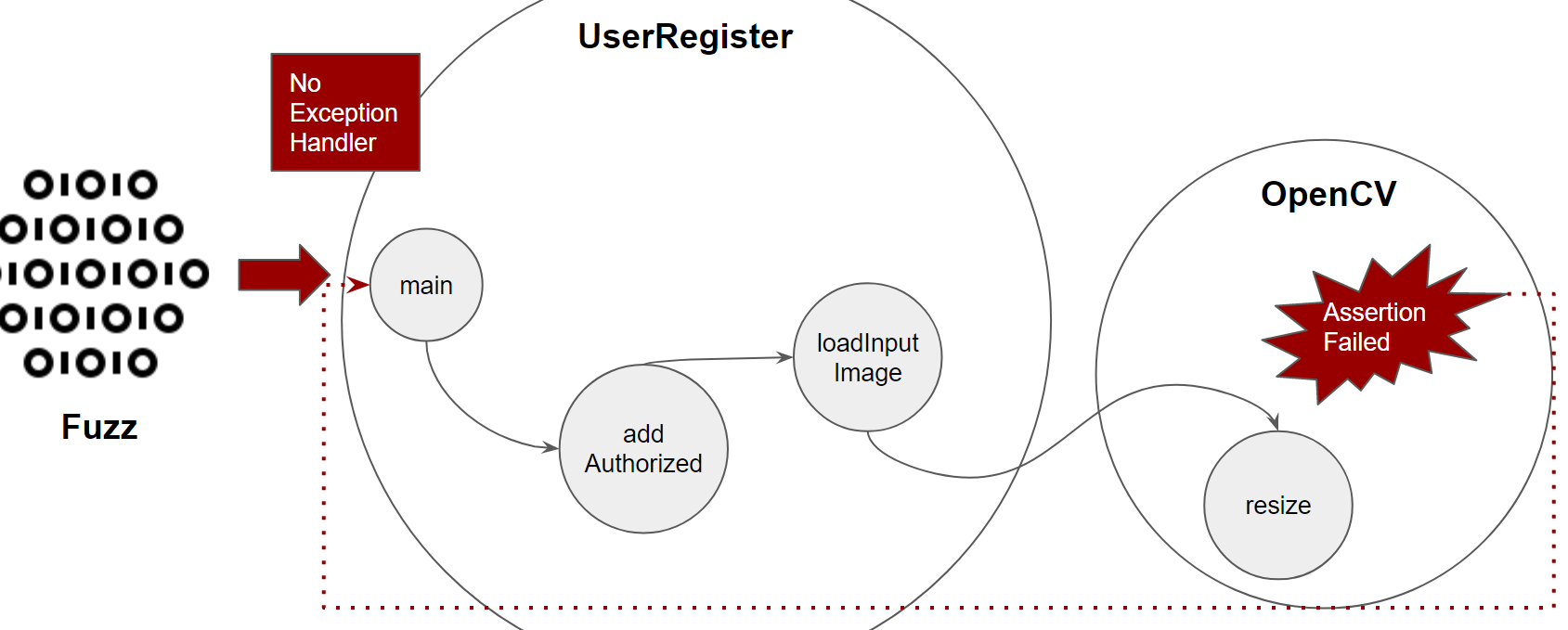
* + 1. Input data attack point

This is the data input related part.

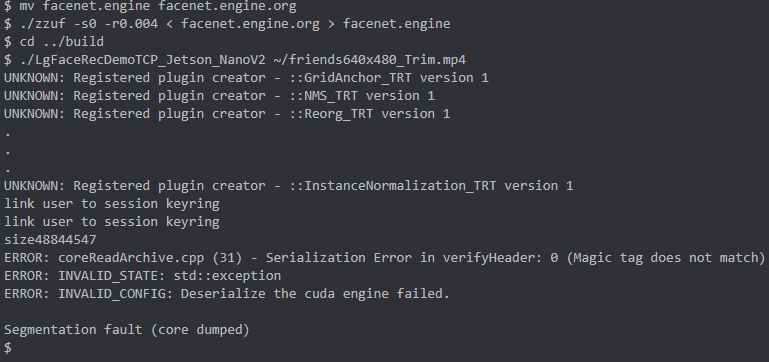
* When registering a user, the name is received as a command argument, and this was tested. Although a 1024-byte buffer is used, a stack smash occurs when a long name is entered.



* Fuzz image data: When registering a user, a registration test was conducted using an image created with zuff. We made about 10,000 images using Zuff and tried to register as a user. The wrong image is processed as assertion failed in OpenCV, but the program ends abnormally because there is no exception handler in the main program.



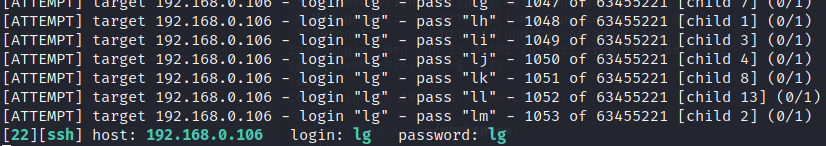
* Engine data file modification: Check if detection is possible for the modified facenet and mtcnn engine files. Since it is very difficult to directly manipulate the engine file, we changed it using zzuf and started the program. The program ended with a segmentation fault.

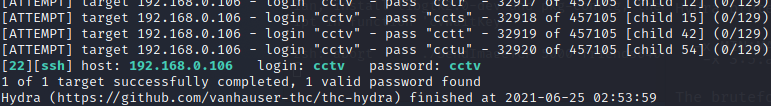


* + 1. Outer attack point

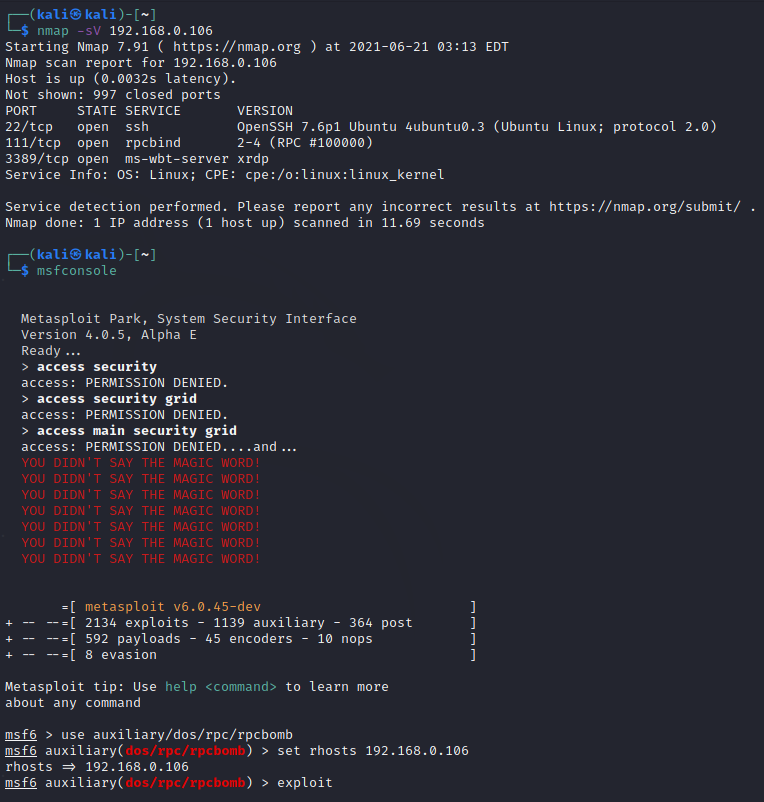
This is the outer attack related part.

* SSH account login attack: In Team4 document, it is required creating cctv and manager linux accounts. A brute force attack with hydra in Kali Linux. As a result of the test, there was no rule to prevent brute-force during the login process, so the lg/lg account succeeded after 1053 attempts, and the cctv/cctv account logged in after 32920 attempts. The manager/manager account failed the attack because the password was long.





* Rpcbind DoS attack: It attacks the port for Rpcbind (nfs) to cause a denial of service. The system has been identified as stopped. But not crashed



* Face recognition error

1. No face(Make face invisible)
2. Printed image
3. Displayed image(Replay image with pad or smartphone)
4. Glasses(Wearing glasses to look like a different person)
5. Dust mask
6. Rigid mask(mask made of hard material)
7. Flexible paper mask(like rubber face mask)
8. Fake head



Actual attacks were case 1, 2, 3, 4, 5. In case 1, 4, and 5, the modified face could not be recognized as a face. These attacks make no log(like, in/stay/out) on the system. In case 2 and 3, it was recognized as another person's face.

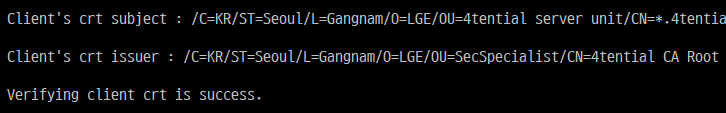
* + 1. Inner attack point

This is the part that occurs when there is more intrusion from the inside.

* Mutual authentication attack: In the code handling the certificate, there is developer code remaining, which does not check certificate name. And the purpose of the certificate is so wide. We tried to test to change the client certificate to a server certificate.

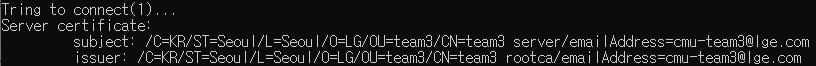
|  |
| --- |
| >openssl x509 -in server.crt -purpose -noout  Certificate purposes:  SSL client : Yes  SSL client CA : No  SSL server : Yes  SSL server CA : No ... |

This is the server's certificate, and it can be used on the client because both client/server are set to Yes for purpose.

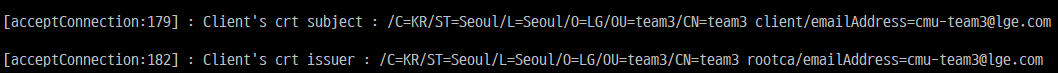


This is the client's certificate (actually the server's certificate) received from the server.

* RootCA certificate change attack: Check that there is no part to check the rootca certificate, change the rootca certificate to another one and test it. Also, assuming that the CCTV keyring can be replaced by a hacker. We changed all the certificates for both server and client. Even though both certificates were replaced and used, the program did not notice the result at all.



Installed a malicious certificate on the server and a malicious rootca certificate on the client. communication is normal.

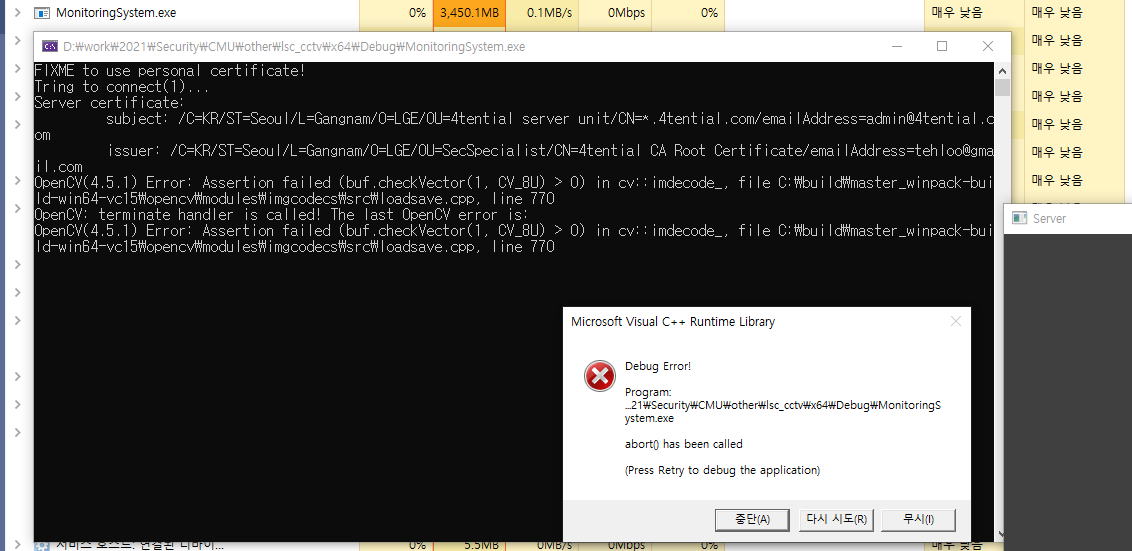


Installed a malicious certificate on the client and a malicious rootca certificate on the server. communication is normal.

* Protocol robustness testing: This is to test the robustness of the network. Data comes only from the server's module, and the network uses TLS, so there is no way to verify the input data in the client because I think it will not be forged or forged in the middle. However, in order to reuse the program code or change the program specification in the future, it seems that proper input verification is required. So, we ran the test below.

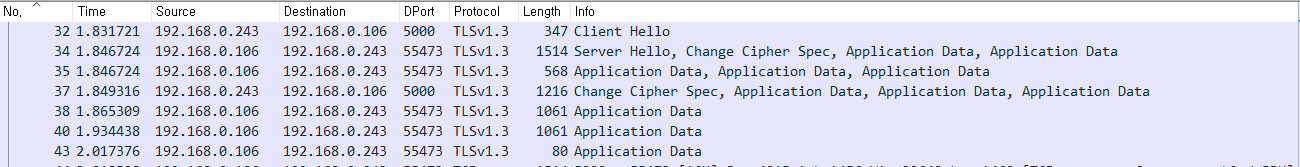
Create input2 through x2.py file and python as follows.

|  |
| --- |
| > type x2.py import sys sys.stdout.buffer.write(b'\xFF'\* 4 + b'\xFF'\* 4294967295) #Actually, Python does not recognize 4294967295, so It need to write it as multiple parts. #The image size is FFFFFFFF bytes, and the dummy data is prepared #as much as the corresponding size.  > py x2.py > input2  > type input2 | openssl s\_server -cert client.crt -key client.key -CAfile rootca.crt -port 5000 Using default temp DH parameters ACCEPT ← At this time, run client application ERROR shutting down SSL CONNECTION CLOSED |

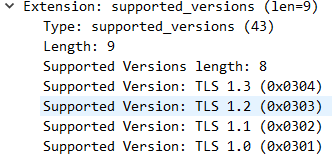


The client allocates 4GB of memory, but it passed safely because it is a 64bit application environment, but the program is stopped for a long time and assert fails in opencv

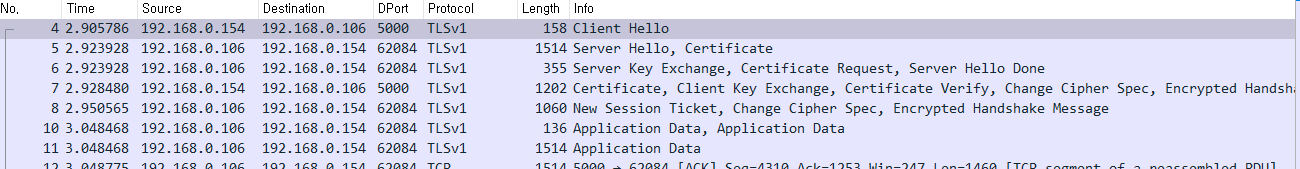
* MITM attack: A monitoring system (client) certificate was exported from the PC and a man-in-the-middle attack was attempted using this. The screen sent by the server can be intercepted by a man in the middle, and the man in the middle can send another screen to the monitoring system. It is possible for an intermediary to use the client certificate without permission and to connect with the client by using it as if it were a server certificate. In addition, this is related to TLS version control, and it leads to a TLS 1.0 connection from the middleman to both Server/Client, exposing the risk on the network.



Network packet capture for default connections



It is designed to support various TLS versions, but among them, it is connected to TLS1.3.

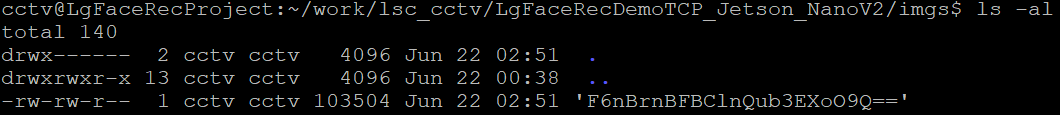


Create a man-in-the-middle and establish a connection to the server and client. And force both connections to TLS1.0. Data is being exposed from intermediaries and network security is not working as expected.

* Same person name attack: If access to the server is possible, pay attention to the file created after user registration on the server. The file name itself is encrypted, so it cannot be used arbitrarily. However, the original encrypted file name is the name used in the user registration process, and if another picture is registered with the same name, the same file is created. Therefore, if one user is normally registered and a second user is registered with the same name, when the second user enters, it is determined that the first user entered.



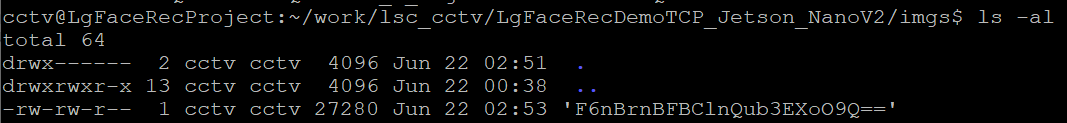
Register the first user(Mike) as Mike



Mike.jpg→ “F6nBrnBFBC1n!ub3EXoO9Q==”



Register the second user(John) as mike



Mike.jpg → “F6nBrnBFBC1n!ub3EXoO9Q==”

Afterwards, when the second user (John) accesses the server room, “F6nBrnBFBC1n!ub3EXoO9Q==” is decrypted by the first user (Mike), so it comes out as the one accessed by the first user (Mike).

* Simple network protocol: This part is not directly related to vulnerability. Since this system sends a JPEG image as soon as it is connected to the server, you can easily obtain a jpeg image without a monitoring system as long as you have a certificate.

|  |
| --- |
| openssl s\_client -connect 192.168.0.106:5000 -cert client.crt -key client.key -CAfile rootca.crt -noservername -quiet -verify\_quiet > output.bin |

output.bin has the following structure.

|  |
| --- |
| [SIZE1=4 BYTE][JPEG SIZE1][SIZE2=4 BYTE][JPEG SIZE2].... |

Therefore, if you add a simple parser that can extract size and jpeg, you can use almost all the functions of the monitoring system without the monitoring system.

# **Results of the Assessment**

The test results in Chapter 3 were classified according to severity.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Severity=High** | **Severity=Medium** | **Severity=Low** |
| **Common Attack** | TLS version control  Deployment process attack | Encryption Key |  |
| **Input Data Attack** | Engine data file modification |  | Weak input validation  Fuzz image data |
| **Outer Attack** | Ssh account login | Face recognition error | Rpcbind dos attack |
| **Inner Attack** | Mutual authentication attack  RootCA certificate change attack  MITM attack | Same person name attack | Protocol robustness testing |

# **Recommendations**

## **Things to improve first**

* Basic requirements not met: Since basic requirements are not implemented, the areas that can be evaluated are limited.
* Discomfort things: For security reasons, various convenience functions were not implemented. There are many restrictions on GUI or program interface, and there are parts that separate the SW program into three and the maintenance work is rather increased.
* Deployment and verify: When making a distribution, it is necessary to develop and test the release mode. And need a process that can verify the executable file.
* Too much task of monitoring system manager: In the security design document, the role of the monitoring system manager is that of a person who registers users, but in reality, there are many tasks such as system log check, cctv service restart, account authority control, ssh connection, scp connection, etc. Most of them are missing from threat modeling and documents. Because the security manager is not an IT technician, it is difficult to do these things without a GUI environment, and the authority seems to be concentrated.
* Remove developer code: Because the fixme code is not deleted, files other than Certificate Management provided by the OS are allowed.
* Remove C coding style: There are parts that adhere to the C coding style, and the compiler will handle them well, but you need to rewrite those parts. (true/1, nullptr/NULL)
* Missing certificate control: When using a certificate, the process of revoke and redistribution is required, but it is missing. Also, the process of authenticating the rootca certificate is missing. In addition, it is necessary to strengthen the part that authenticates the counterpart's certificate.
* Tls version control error: The design document says that 1.2 is supported, and the actual operation is set to 1.3, and TLS1.0, TLS1.1, TLS1.2, and TLS1.3 are all supported by program settings.
* Hard-coded IV of AES-CBC, and secure key expansion missing

# **Conclusion**

The basic encryption data storage and key management using the functions of the operating system are working well. We have listed many vulnerabilities, but only a few have a real chance of success. Such things do not cause any major problems in use, so if necessary, it seems that they can be solved through an update. However, it is necessary to reconsider the inconvenient interface. Although a minimal implementation may be good for security, very few people will want to write a program that is inconvenient to use.

# **Reference Images**

<https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation>

<https://blog.cloudflare.com/tls-1-3-overview-and-q-and-a/>

<https://www.researchgate.net/figure/Attack-categories-in-WMCA-dataset-only-RGB-images-are-shown-Print-and-Replay_fig4_343150267>