SWEN90006_2023_SM2
Security & Software Testing

Group 15

Junheng Chen 1049540
Ling Huang 1015103
Yingying Guo 1382000
Hanyu Wang 1347595

1. Experimental Environment and Tools

- Operating System: Ubuntu 18.04.6 LTS
- Software: Docker Desktop 4.22.1, VS code 1.82.0
- Fuzzer: AFLNet

2. Experimental Procedures and Results

2.1. Fuzz Testing Setup

2.1.1. Git Clone the Repository

```
git clone https://github.com/SWEN90006-2023/swen90006-assignment-2-group-15.git
```

2.1.2. Set Up Docker Image and Container

```
cd swen90006-assignment-2-group-15
#Create a image and its container, and then get into the container bash
docker build . -t swen90006-assignment2
docker run -it swen90006-assignment2
```

2.1.3. Compile Source Code of the Topstream Server

```
cd $WORKDIR/topstream
make all
```

2.1.4. Prepare A Fuzzing Setup

```
# Install clang (as required by AFL/AFLNet to enable llvm_mode)
sudo apt-get install clang
# Install graphviz development
sudo apt-get install graphviz-dev libcap-dev

# First, clone this AFLNet repository to a folder named aflnet
git clone https://github.com/aflnet/aflnet.git aflnet
# Then move to the source code folder
cd aflnet
make clean all
```

```
cd llvm_mode
# The following make command may not work if llvm-config cannot be found
# To fix this issue, just set the LLVM_CONFIG env. variable to the specific llvm-config version
on your machine
# On Ubuntu 18.04, it could be llvm-config-6.0 if you have installed clang using apt-get
make
# Move to AFLNet's parent folder
cd ../..
export AFLNET=$(pwd)/aflnet
export WORKDIR=$(pwd)

# Setup PATH environment variables
export PATH=$PATH:$AFLNET
export AFL_PATH=$AFLNET
```

2.1.5. Fuzzing

```
# Fuzzing
afl-fuzz -d -c $WORKDIR/topstream/restart.sh -i $WORKDIR/results/sample -o
$WORKDIR/results/test_out -N tcp://127.0.0.1/8888 -P TOPSTREAM -D 10000 -q 2 -s 2 -E -K
-R $WORKDIR/topstream/topstream-fuzz 127.0.0.1 8888 127.0.0.1 9999
```

2.2. Seed Design and Description

We have modified the DPIN to 1111 for smoother fuzz testing, so before the fuzzing testing and checking the coverage score, always make sure the generated PIN code will be 1111, otherwise, our seed corpus will not work properly.
All seeds are in results/seed_corpus. All seeds in dat format are stored in the dat folder, dat is more friendly to gcov. while all seeds in raw format are stored in the raw folder for fuzzing. The fuzzing folder Contains two seeds, tsUPDP.raw, and ListLoadPlay.raw. This is based on the seeds in the raw folder. Through fuzz testing, we found that these two seeds were most likely to cause the program to crash, so we separated them separately for separate fuzzing.

2.2.1. tsUSER.dat

- Try to trigger error520 with empty parameters:
    - USER
- Try triggering error400 to test logging in to an unregistered user:
    - USER wula
- Try to trigger error530 using USER while logged in:
    - USER admin
    - PASS admin
    - DPIN 1111
    - REGU test,testpass
    - USER test
    - LOGO
- Try to trigger success 210 username successfully:
    - USER admin

2.2.2. tsPASS.dat

- Try to trigger error530 and enter the PASS command in the login state:
    - USER admin
    - PASS admin
    - DPIN 1111
    - PASS admin
    - LOGO
- Try to trigger error520 with empty parameters.
    - USER admin
    - PASS
- Try to trigger error410 with wrong password:

    - USER admin
    - PASS admi
- Try to trigger success220 to log in to an account that does not require a PIN:
    - USER admin
    - PASS admin
    - DPIN 1111
    - REGU testForPass,testpass
    - LOGO
    - USER testForPass
    - PASS testpass
    - LOGO
- Try to trigger success290 and log in to an account with a PIN:
    - USER admin
    - PASS admin
    - DPIN 1111
    - LOGO

2.2.3. tsUPDP.dat

- Try to trigger error530 and change password when not logged in:
    - UPDP newpass,newpass
- Try to trigger error520 with empty parameters:
    - USER admin
    - PASS admin
    - DPIN 1111
    - UPDP
- Try to trigger error520 format error without "," or only pass in one parameter:
    - UPDP newpassnewpass
- Try to trigger error450 Password Inconsistency:
    - UPDP newpass,newpas
- Try to trigger success300:
    - UPDP newpass,newpass
    - UPDP admin,admin
    - REGU primary_user,primary_user
    - LOGO
- Try to trigger success300 in a different way, especially when the new passwords is long:
    - USER primary_user
    - PASS primary_user
    - UPDP codecodecodecodecodecodecodecodecodecodecodecodecodecode,codecodecodecodecodecodecodecodecodecodecodecodecodecodecodecodecodecodecode

    - LOGO
    - USER primary_user
    - PASS codecodecodecodecodecodecodecodecodeocdeocdecodecodecodecode
    - LIST
    - LOGO

2.2.4. tsDPIN.dat

- Try to trigger error530 using the DPIN command after the user logs in:
    - USER admin
    - PASS admin
    - DPIN 1111
    - DPIN 1111
    - LOGO
- Try to trigger error520 with empty parameters:
    - USER admin
    - PASS admin
    - DPIN
- Try triggering error440 by entering wrong PIN and try multiple times:
    - DPIN 1112
    - DPIN 1113
    - DPIN 1114
    - DPIN 1115
- Try triggering success220:
    - DPIN 1111
    - LOGO

2.2.5. tsREGU.dat

- Try to trigger error530 when not logged in using the REGU command:
    - REGU testForRegu,testpass
- Try to trigger error520 with empty parameters:
    - USER admin
    - PASS admin
    - DPIN 1111
    - REGU test,testpass
    - REGU
    - LOGO
- Try to trigger error430 and try the REGU command using a non-admin account:

    - USER test
    - PASS testpass
    - REGU testForRegu,testpass
    - LOGO
- Try to trigger error460 and try to register a user with the same name:
    - USER admin
    - PASS admin
    - DPIN 1111
    - REGU test,testpass1
- Try to trigger error451. The password input is too long when registering:
    - REGU testForRegu,1234567891011121314151617181920
- Try to trigger error520 REGU by passing in only one parameter:
    - REGU testForRegutestpass
- Try to trigger success230:
    - REGU testForRegu,testpass
    - LOGO

2.2.6. tsAMFA.dat

- Try to trigger error 530 set AMFA without logging in:
    - AMFA 0123456789
- Try to trigger error520 with empty parameters:
    - USER admin
    - PASS admin
    - DPIN 1111
    - REGU testForAmfa,testpass
    - AMFA
- Try to trigger error540 error format:
    - AMFA 012345
    - AMFA 012345aaaaaaa
    - AMFA 01234a6789
- Try triggering success280:
    - AMFA 0123456789
    - LOGO
    - USER testForAmfa
    - PASS testpass
    - AMFA 0123456789
    - LOGO

2.2.7. tsUPDA.dat

- Trying to use the UPDATE command while not logged in will not respond with the error code:
    - UPDA test,VIP
- Try to trigger error520 with empty parameters:
    - USER admin
    - PASS admin
    - DPIN 1111
    - REGU testForPass,testpass
    - UPDA
    - LOGO
- Try to trigger error 430 by entering the UPDATE command from a non-admin account:
    - USER testForPass
    - PASS testpass
    - UPDA testForPass,VIP
    - LOGO
- Trying to trigger error400 and error490, non-existent user, malformed parameters, and NULL:
    - USER admin
    - PASS admin
    - DPIN 1111
    - UPDA
    - UPDA testForPass,WULA
    - UPDA testForPasss,VIP
    - UPDA testForPasss
- Try to trigger success310:
    - UPDA testForPass,FREE
    - UPDA testForPass,BASIC
    - UPDA testForPass,VIP
    - UPDA testForPass,VIPPPP (VIPPPP can also successfully upgrade the account to VIP)
    - LOGO

2.2.8. ListLoadPlay.dat

This seed includes all the contents of tsLIST, tsPLAY, tsLOAD, tsLOGO, and tsQUIT.

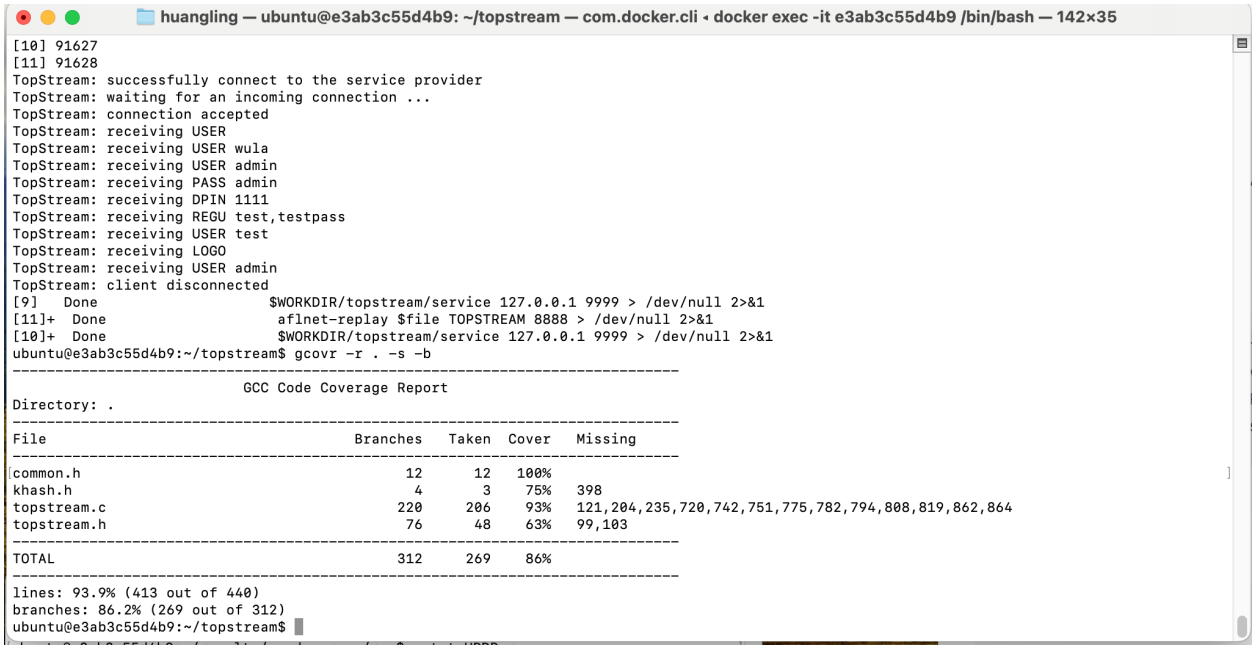
- Try to trigger error420 without loading any movies LIST:
    - LIST

- Try to trigger error420 and play when the movie has not been loaded:
    - LOAD Movies_by_group15.txt
    - LOAD bufferOverflow.txt
    - USER admin
    - PASS admin
    - DPIN 1111
    - REGU testForPass,testpass
    - PLAY 5
    - LOGO
- Try to trigger error530 LOAD when not logged in:
    - LOAD movies.txt
- Try to trigger error520 with empty parameters:
    - USER admin
    - PASS admin
    - DPIN 1111
    - LOAD
    - LOGO
- Try to trigger error430 LOAD under non-admin account:
    - USER testForPass
    - PASS testpass
    - LOAD movies.txt
    - LOGO
- Try to trigger error550 LOAD non-existent txt:
    - USER admin
    - PASS admin
    - DPIN 1111
    - LOAD movieeeeees.txt
- Try to trigger error 560 Error format in movies txt:
    - LOAD movies1ForError560.txt
    - LOAD movies2ForError560.txt
    - LOAD movies.txt
    - LOAD mix.txt
    - LOAD error_length_not_number.txt
    - LOAD error_length_negative.txt
    - LOAD error_type_not_number.txt
    - LOAD error_type_negative.txt
    - LOAD error_insufficient_data.txt
    - LOAD error_too_much_data.txt
    - LOGO
- Try to trigger successcode:

    - LIST
- Try to trigger error530 and enter the PLAY command when not logged in:
    - PLAY 5
- Try to trigger error520 with empty parameters:
    - USER admin
    - PASS admin
    - DPIN 1111
    - PLAY
- Try to trigger error470: Use FREE user to load a movie that he does not have permission to watch:
    - REGU freeaccount,freepass
    - UPDA freeaccount,FREE
    - LOGO
    - USER freeaccount
    - PASS freepass
    - PLAY 1
    - LOGO
- Try to trigger error480:PLAY a non-existent movie:
    - USER admin
    - PASS admin
    - DPIN 1111
    - PLAY 100
- Try to trigger error520: Enter non-numeric parameter:
    - PLAY ABC
- successcode, try to play some movies:
    - PLAY 0
    - PLAY 1
    - PLAY 2
    - PLAY 3
    - PLAY 4
    - PLAY 5
    - PLAY 6
    - PLAY 7
    - PLAY 8
    - PLAY 9
    - PLAY 10
    - PLAY 11
    - PLAY 12
    - PLAY 13
    - PLAY 14

    - PLAY 15
    - PLAY 16
    - PLAY 17
    - PLAY 18
    - PLAY 19
    - PLAY 20
- success260:
    - LOGO
- Try to trigger error 530 and enter the LOGO command when not logged in:
    - LOGO
- Enter a completely unknown command:
    - WULA 123
    - QUIT
- Try continuing to enter the command after QUIT:
    - USER admin
    - PASS admin
    - DPIN 1111
    - PLAY 100
    - PLAY ABC
    - PLAY 0
    - PLAY 1
    - PLAY 2

2.3. Code Coverage



```
huangling — ubuntu@e3ab3c55d4b9: ~/topstream — com.docker.cli ◂ docker exec -it e3ab3c55d4b9 /bin/bash — 142×35
[10] 91627
[11] 91628
TopStream: successfully connect to the service provider
TopStream: waiting for an incoming connection ...
TopStream: connection accepted
TopStream: receiving USER
TopStream: receiving USER wula
TopStream: receiving USER admin
TopStream: receiving PASS admin
TopStream: receiving DPIN 1111
TopStream: receiving REGU test,testpass
TopStream: receiving USER test
TopStream: receiving LOGO
TopStream: receiving USER admin
TopStream: client disconnected
[9]   Done                    $WORKDIR/topstream/service 127.0.0.1 9999 > /dev/null 2>&1
[11]+  Done                    aflnet-replay $file TOPSTREAM 8888 > /dev/null 2>&1
[10]+  Done                    $WORKDIR/topstream/service 127.0.0.1 9999 > /dev/null 2>&1
ubuntu@e3ab3c55d4b9:~/topstream$ gcovr -r . -s -b
------------------------------------------------------------------------------
                           GCC Code Coverage Report
Directory: .
------------------------------------------------------------------------------
File                                       Branches   Taken  Cover   Missing
------------------------------------------------------------------------------
common.h                                         12      12   100%
khash.h                                           4       3    75%   398
topstream.c                                     220     206    93%   121,204,235,720,742,751,775,782,794,808,819,862,864
topstream.h                                      76      48    63%   99,103
------------------------------------------------------------------------------
TOTAL                                           312     269    86%
------------------------------------------------------------------------------
lines: 93.9% (413 out of 440)
branches: 86.2% (269 out of 312)
ubuntu@e3ab3c55d4b9:~/topstream$
```

Fig.1 Code Coverage of Fuzzer Running

Lines coverage:93.9%
Branches coverage:86.2%

2.3.1. How to Reproduce Coverage

2.3.1.1. Replace File

The first major premise is to replace the topstream.c file and use the original topstream.c file to achieve the highest coverage, because some assertions are added to the current topstream.c for the next step of fuzz testing.

2.3.1.2. Change tsDPIN Function From Return Result to Return 1111

2.3.1.3. Compile New topstream.c

cd $WORKDIR/topstream
make all

2.3.1.4. Run All Seeds Under topstream Path

```
for file in $WORKDIR/results/seed_corpus/dat/*.dat; do
    $WORKDIR/topstream/service 127.0.0.1 9999 > /dev/null 2>&1 &
    aflnet-replay $file TOPSTREAM 8888 > /dev/null 2>&1 &
    $WORKDIR/topstream/topstream-gcov 127.0.0.1 8888 127.0.0.1 9999
```

done

2.3.1.5. Run All Queues Under topstream Path

Note: you may need to change /home/ubuntu to the different content based on the system and environment you are using

for f in $(echo /home/ubuntu/results/generated_corpus/replayable-queue/id*); do echo "Processing $f ..."; $WORKDIR/topstream/service 127.0.0.1 9999 >> /home/ubuntu/log.txt 2>&1 & aflnet-replay $f TOPSTREAM 8888 > /dev/null 2>&1 & $WORKDIR/topstream/topstream-gcov 127.0.0.1 8888 127.0.0.1 9999; done

2.3.1.6. View Coverage

gcovr -r . -s
gcovr -r . -s -b

2.4. Discovered Vulnerabilities

We have found 4 vulnerabilities in the topstream.c, including 3 Buffer overflows problems and 1 unauthorized access problem.
Furthermore, we have identified a series of memory leaks that were causing the program to crash. However, since the ED discussion made it clear that these were not considered as "vulnerabilities" for this assignment, we only wrote reproducible seeds for two of the memory leak issues. Seeds are stored in-home/ubuntu/results/others called REGU_memory_leak and UPDA_memory_leak(although they may not be considered as vulnerabilities).

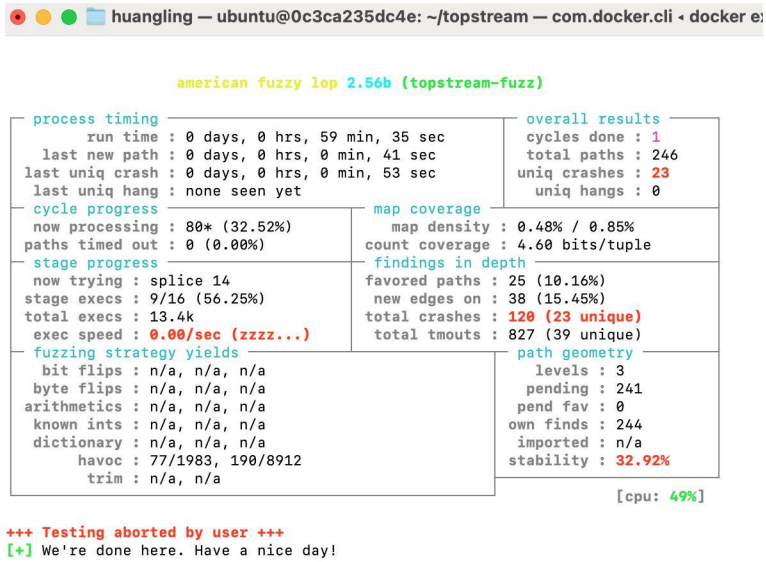


Fig.2 Results of Fuzzer Running

2.4.1. Assertion

Before using AFLNet for fuzz testing, we learned that Null pointer dereference problems are more likely to occur when the parameter is not confirmed to be NULL. Stack/Heap Buffer Overflow and Use-After-Free problems are more likely to occur when “malloc”, “free”, and Improper use of string.h library and some situations exceed the allocated memory range.
So we added assertions in advance to some places where vulnerabilities may occur.

```c
/**
 * Check if a username exists
 */
int isUser(const char* name) {
  assert(name != NULL);
  for (ki = kh_begin(users); ki != kh_end(users); ++ki) {
    if (kh_exist(users, ki)) {
      if (!strcmp(kh_key(users,ki), name)) return 1;
    }
  }
  return 0;
}
```

Fig.3 Assertion in isUser Function

```c
/**
 * Check if the given password is correct
 */
int isPasswordCorrect(const char* password) {
  assert(password != NULL);
  assert(active_user_name != NULL);
  for (ki = kh_begin(users); ki != kh_end(users); ++ki) {
    if (kh_exist(users, ki)) {
      if (!strcmp(kh_value(users,ki)->password, password) &&
          !strcmp(kh_key(users,ki), active_user_name))
      return 1;
    }
  }
  return 0;
}
```

Fig.4 Assertion in isPasswordCorrect Function

```c
/**
 * Check if the current user has MFA enabled
 */
int isMFAEnabled() {
  assert(active_user_name != NULL);
  ki = getUser(active_user_name);
  user_info_t *user = kh_value(users, ki);
  if (user->device_id != NULL) {
    return 1;
  }
  return 0;
}
```

Fig.5 Assertion in isMFAEnabled Function

```c
/**
 * Send a PIN to the service provider (e.g., a telco service)
 * so that it can be forwarded to the user device (e.g., a mobile phone)
 */
void sendPIN(int PIN) {
  char message[MSG_BUF_SIZE];

  ki = getUser(active_user_name);
  user_info_t *user = kh_value(users, ki);

  assert(user->device_id != NULL);

  sprintf(message, "Device-%s, PIN-%d\r\n", user->device_id, PIN);

  assert(strlen(message) < MSG_BUF_SIZE);

  if(send(service_sock, message, strlen(message) , 0) < 0)
  {
    fprintf(stderr,"[ERROR] TopStream cannot communicate with the service provider");
  }
}
```

Fig.6 Assertion in sendPIN Function

```c
/**
 * Handle user login
 * E.g. USER admin
 */
int tsUSER(char *params) {
  if (ts_state == INIT) {
    if (params == NULL) {
      return sendResponse(client_sock, error520);
    }
    //Check if the user exits
    if (!isUser(params)) {
      return sendResponse(client_sock, error400);
    } else {
      sendResponse(client_sock, success210);
      //Update the current active user name
      free(active_user_name);
      active_user_name = strdup(params);
      assert(active_user_name != NULL);
      //Update server state
      ts_state = USER_OK;
    }
  } else {
    return sendResponse(client_sock, error530);
  }

  return 0;
}
```

Fig.7 Assertion in tsUSER Function

```c
/**
 * Handle REGU-Register new user command
 * E.g. REGU test,testpass
 */
int tsREGU(char *params) {
  if (ts_state == LOGIN_SUCCESS) {
    if (params == NULL) {
      return sendResponse(client_sock, error520);
    }

    if (strcmp(active_user_name, "admin")) {
      return sendResponse(client_sock, error430);
    }

    //This command expects two arguments/parameters
    //(username and password) seperated by a comma
    //e.g. REGU newuser,newpassword
    char **tokens = NULL;
    int count = 0;
    tokens = strSplit(params, ",", &count);

    if (count == 2) {
      //Check if there exists an user with the same username
      khint_t k = getUser(tokens[0]);
      if (k != kh_end(users)) {
        sendResponse(client_sock, error460);
      } else {
        if (strlen(tokens[1]) > MAX_PASSWORD_LENGTH) {
          sendResponse(client_sock, error451);
        } else {
          user_info_t *user = newUser();
          strcpy(user->password, tokens[1]);

          char* new_username = strdup(tokens[0]);
          assert(new_username != NULL); // Ensure memory for string was allocated successfully

          ki = kh_put(hmu, users, new_username, &discard);
          kh_value(users, ki) = user;
          sendResponse(client_sock, success230);
        }
      }
    } else {
      sendResponse(client_sock, error520);
    }

    freeTokens(tokens, count);
  } else {
    sendResponse(client_sock, error530);
  }
  return 0;
}
```

Fig.8 Assertion in tsREGU Function

```c
/**
 * Handle LOAD-Load movies from a file
 * E.g. LOAD movies.txt
 */
int tsLOAD(char *params) {
  if (ts_state == LOGIN_SUCCESS) {
    if (params == NULL) {
      return sendResponse(client_sock, error520);
    }

    if (strcmp(active_user_name, "admin")) {
      return sendResponse(client_sock, error430);
    }

    //This command expects one argument/parameter
    //specifying the filename of the file containing the movie data
    //e.g. LOAD movies.txt

    FILE *fp;
    if ((fp = fopen(params, "rb")) == NULL) {
      return sendResponse(client_sock, error550);
    } else {
      //load movies from file to memory
      klist_t(lmv) *tmpMovies = kl_init(lmv);
      char line[MAX_MOVIE_INFO_LENGTH];

      while (fgets(line, sizeof(line), fp)) {
        assert(strlen(line) < MAX_MOVIE_INFO_LENGTH - 1);  // Ensure no buffer overflow

        //Check for the last emply line
        //Be a bit conservative here
        if (strlen(line) <= 2) continue;

        char** tokens = NULL;
        int count = 0;
        tokens = strSplit(line, ",", &count);

        if (count == 3) {
          movie_info_t *m = (movie_info_t *) malloc(sizeof(movie_info_t));
          assert(m != NULL);  // Identify null pointer dereference
          m->name = strdup(tokens[0]);
          assert(m->name != NULL); // Identify null pointer dereference
          if (!isNumber(tokens[1])) goto movie_format_error;
          if (atoi(tokens[1]) < 0) goto movie_format_error;
          m->length = atoi(tokens[1]);
```

Fig.9 Assertion in tsLOAD Function

```c
/**
 * Hanlde PLAY-Play a selected movie
 * E.g. PLAY 5
 */
int tsPLAY(char *params) {
  if (ts_state == LOGIN_SUCCESS) {
    if (params == NULL) {
      return sendResponse(client_sock, error520);
    }

    kliter_t(lmv) *it;
    it = kl_begin(movies);

    //Check if the movie list is empty
    if (it == kl_end(movies)) {
      return sendResponse(client_sock, error420);
    }

    if (isNumber(params)) {
      int index = atoi(params);
      char *mname = getMovieName(index);
      if (mname != NULL) {

        ki = getUser(active_user_name);
        user_info_t *user = kh_value(users, ki);

        //Check permission
        if (user->type >= getMovieType(index)) {
          sendResponse(client_sock, successcode);
          char tmpMovieStr[MAX_MOVIE_INFO_LENGTH];
          assert(strlen(mname) + strlen(" Playing ") + strlen(" ...\r\n") + 1 <= MAX_MOVIE_INFO_LENGTH);
          sprintf(tmpMovieStr, " Playing %s ...\r\n", mname);
          sendResponse(client_sock, tmpMovieStr);
        } else {
          sendResponse(client_sock, error470);
        }
      } else {
        sendResponse(client_sock, error480);
      }
    } else {
      sendResponse(client_sock, error520);
    }
  } else {
    sendResponse(client_sock, error530);
  }
  return 0;
}
```

Fig.10 Assertion in tsPLAY Function

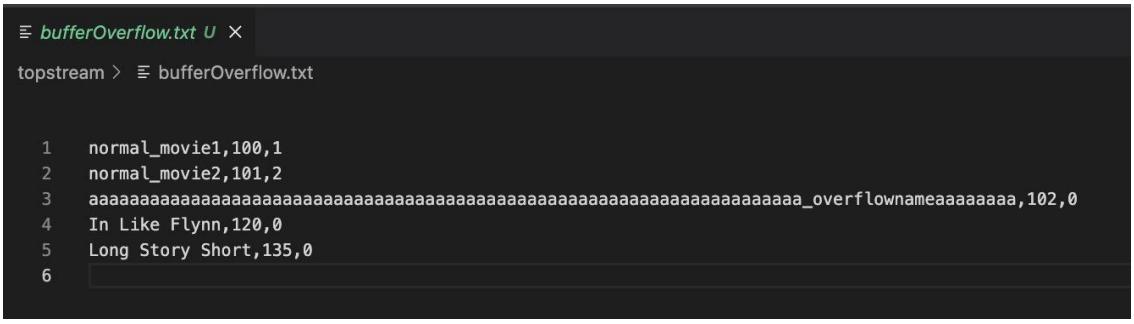
Since the above assertions are only successfully triggered during fuzzing, tsPLAY and tsLOAD are successfully triggered, so for the sake of seed coverage, the topstream.c we finally submitted only includes the assertions of PLAY and LOAD, and the full version is also stored in the folder: *results/others* called topstream_with_assert.c

2.4.2. Stack-Buffer-Overflow in tsLIST

2.4.2.1. Vulnerability Description

After loading movies from files, details such as the name, length, and type of the movies are stored in the movies list. Executing the ‘tsLIST’ function retrieves movie information from the list, reading it line by line into a char array of size ‘MAX_MOVIE_INFO_LENGTH’ (100 digits). However, the function does not account for potential instances where movie information might exceed the length of ‘MAX_MOVIE_INFO_LENGTH’. In such cases, the char array may experience a stack-buffer-overflow.

The vulunability was found with the seeds mutation by fuzzer and a bad movies file.



```
normal_movie1,100,1
normal_movie2,101,2
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa_overflownameaaaaaaaa,102,0
In Like Flynn,120,0
Long Story Short,135,0

```

Fig.11 Bad Movies File (bufferOverflow.txt) to Trigger Vulnerability

The third line includes: 93-digit movie name with 6-digit string, ",102,0", a total of 99 digits, and the last digit "\0" needs to be given for string type.

```
huangling — ubuntu@0c3ca235dc4e: ~/topstream — com.docker.cli ◂ docker exec -it 0c3ca235dc4e /bin/bash — 162×63
TopStream: receiving LOAD error_length_not_number.txt
TopStream: receiving LOAD error_length_negative.txt
TopStream: receiving LOAD error_type_not_number.txt
TopStream: receiving LOAD error_type_negative.txt
TopStream: receiving LOAD error_insufficient_data.txt
TopStream: receiving LOAD error_too_much_data.txt
TopStream: receiving LOGO
TopStream: receiving LIST
=================================================================
==50108==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7ffda9b64be4 at pc 0x000000446f8a bp 0x7ffda9b64a60 sp 0x7ffda9b64210
WRITE of size 109 at 0x7ffda9b64be4 thread T0
    #0 0x446f89  (/home/ubuntu/topstream/topstream-asan+0x446f89)
    #1 0x4472b6  (/home/ubuntu/topstream/topstream-asan+0x4472b6)
    #2 0x51905a  (/home/ubuntu/topstream/topstream-asan+0x51905a)
    #3 0x51c397  (/home/ubuntu/topstream/topstream-asan+0x51c397)
    #4 0x7fac084c1c86  (/lib/x86_64-linux-gnu/libc.so.6+0x21c86)
    #5 0x41a0f9  (/home/ubuntu/topstream/topstream-asan+0x41a0f9)

Address 0x7ffda9b64be4 is located in stack of thread T0 at offset 132 in frame
    #0 0x518aaf  (/home/ubuntu/topstream/topstream-asan+0x518aaf)

  This frame has 2 object(s):
    [32, 132) 'tmpMovieStr' (line 536)
    [176, 186) 'type' (line 537) <== Memory access at offset 132 partially underflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism or swapcontext
      (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow (/home/ubuntu/topstream/topstream-asan+0x446f89)
Shadow bytes around the buggy address:
  0x100035364920: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100035364930: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100035364940: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100035364950: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100035364960: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1
=>0x100035364970: 00 00 00 00 00 00 00 00 00 00 00 00[04]f2 f2 f2
  0x100035364980: f2 f2 00 02 f3 f3 f3 f3 00 00 00 00 00 00 00 00
  0x100035364990: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000353649a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000353649b0: 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1 f1 f1
  0x1000353649c0: 04 f2 00 00 f2 f2 00 00 f2 f2 00 00 00 00 00 00
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
==50108==ABORTING
[5]   Broken pipe             $WORKDIR/topstream/topstream-asan 127.0.0.1 8888 127.0.0.1 9999$WORKDIR/topstream/service 127.0.0.1 9999 > /dev/null 2>&1
[6]   Done                    aflnet-replay $WORKDIR/results/tsPLAY_LIST/replayable-crashes/id:000001,sig:06,src:000001+000090,op:splice,rep:2 TOPSTREAM 8888 > /d
ev/null 2>&1
ubuntu@0c3ca235dc4e:~/topstream$
```

Fig.12 Example of Trigger Vulnerability

2.4.2.2. Code Analysis

As shown in Fig.13, the vulnerability is caused by the code that does not verify whether the string " Playing " + mname + " ...\r\n" exceeds the array ‘tmpMovieSt’ size.

```c
int tsLIST(char *params) {
  kliter_t(lmv) *it;
  it = kl_begin(movies);

  //Check if the movie list is empty
  if (it == kl_end(movies)) {
    sendResponse(client_sock, error420);
    return 1;
  }

  //Otherwise, send back the movie list to the user
  //in a formated string
  int index = 0;
  while(it != kl_end(movies)) {
    movie_info_t *m = kl_val(it);
    sendResponse(client_sock, successcode);
    char tmpMovieStr[MAX_MOVIE_INFO_LENGTH];
    char type[10];
    if (m->type == 0) {
      strcpy(type, "FREE");
    } else if (m->type == 1) {
      strcpy(type, "BASIC");
    } else {
      strcpy(type, "VIP");
    }
    sprintf(tmpMovieStr, " %d. %s, %d, %s\r\n", ++index, m->name, m->length, type);
    sendResponse(client_sock, tmpMovieStr);
    it = kl_next(it);
  }
  return 0;
}
```

Fig.13 Code of tsLIST Function

2.4.2.3. Impact

Attackers can exploit the vulnerability to launch a Denial of Service attack by using excessively long strings, which leads to a stack overflow when executing the 'tsLIST' function after loading bad movies file. As the stack overflow isn't handled adequately by the server, it can cause it to crash, preventing other users from accessing the service.

In addition, attackers can include a malware script in the string that would write in the near memory to be executed to perform a series of actions that would cause financial damage to the TopStream service owner, such as leaking sensitive data or elevating privileges.

2.4.2.4. How to Replay

We have created a seed replay_tsLIST.dat that specifically triggers this vulnerability in the folder: **results/posc**
it can be reproduced using the following command:
cd topstream
make all

$WORKDIR/topstream/service 127.0.0.1 9999 > /dev/null 2>&1 & aflnet-replay $WORKDIR/results/posc/replay_tsLIST.dat TOPSTREAM 8888 > /dev/null 2>&1 & $WORKDIR/topstream/topstream-asan 127.0.0.1 8888 127.0.0.1 9999
Alternatively, to replay, you can use telnet to connect to the server and manually enter the following command（we recommend you in this way）.

- **Terminal 1:**
  - cd $WORKDIR/topstream
  - make all
  - ./service 127.0.0.1 9999
- **Terminal 2:**
  - cd $WORKDIR/topstream
  - ./topstream 127.0.0.1 8888 127.0.0.1 9999
- **Terminal 3:**
  - cd $WORKDIR/topstream
  - telnet 127.0.0.1 8888
  - USER admin
  - PASS admin
  - DPIN 1111
  - LOAD bufferOverflow.txt
  - LIST

2.4.3. Stack-Buffer-Overflow in tsPLAY

2.4.3.1. Vulnerability Description

Similar to ‘tsLIST’, the ‘tsPLAY’ function also retrieves movie information from the list after loading movies from files and reads it line by line into a char array of size ‘MAX_MOVIE_INFO_LENGTH’. Also, the function doesn't verify if the movie information might surpass this length, which could potentially lead to a stack overflow.

The vulunability was found with the seeds mutation by fuzzer and a bad movies file.

```
huangling — ubuntu@0c3ca235dc4e: ~/topstream — com.docker.cli ◂ docker exec -it 0c3ca235dc4e /bin/bash — 162×64
PLAY 100
TopStream: receiving PLAY 17
TopStream: receiving LOAD mix.txt
TopStream: receiving LOAD Movies_by_group15.txt
TopStream: receiving LOAD bufferOverflow.txt
TopStream: receiving USE
TopStream: receiving PASS admin
TopStream: receiving DPIN 1111
TopStream: receiving REGU testForPass,testpass
TopStream: receiving PLAY 5
TopStream: receiving LOGO
TopStream: receiving LOAD movies.txt
TopStream: receiving USER admin
TopStream: receiving PASS admin
TopStream: receiving DPIN 1111
TopStream: receiving LOAD
TopStream: receiving LOGO
TopStream: receiving USER testForPass
TopStream: receiving PASS testpass
TopStream: receiving LOAD movies.txt
TopStream: receiving LOGO
TopStream: receiving USER admin
TopStream: receiving PASS admin
TopStream: receiving DPIN 111;
TopStream: receiving LOAD movieeeeees.txt
TopStream: receiving LOAD movies1For
TopStream: receiving LOAD movies2ForError560.txt
TopStream: receiving LOAD movies.txt
TopStream: receiving LOAD mix.txt
TopStream: receiving LOAD error_length_not?
TopStream: receiving LOAD Frror_length_negative.txt
TopStream: receiving LOAD error_type_not_number.txt
TopStream: receiving LOAD error_type_negative.txt
TopStream: receiving LOAD error_insufficient_data.txt
TopStream: receiving LOAD error_too_much_data.txt
TopStream: receiving LOGO
TopStream: receiving LIST
TopStream: receiving PLAY 5
TopStream: receiving USER admin
TopStream: receiving PASS admin
TopStream: receiving DPIN 1111
TopStream: receiving PLAY
TopStream: receiving REGU freeaccount,freepass
TopStream: receiving LOAD bufferOverflow.txt
TopStream: receiving USER
TopStream: receiving UPDA freeaccount,FREE
TopStream: receiving LOGO
TopStream: receiving USER freeaccount
TopStream: receiving PASS freepass
TopStream: receiving PLAY 1
TopStream: receiving LOGO
TopStream: receiving USER admin
TopStream: receiving PASS admin
TopStream: receiving DPIN 1111
TopStream: receiving PLAY 100
TopStream: receiving PLAY ABC
TopStream: receiving PLAY 0
TopStream: receiving PLAY 1
TopStream: receiving PLAY 2
TopStream: receiving PLAY 3
topstream-asan: topstream.c:659: int tsPLAY(char *): Assertion `strlen(mname) + strlen(" Playing ") + strlen(" ...\r\n") + 1 <= MAX_MOVIE_INFO_LENGTH' failed.
[1]-  Exit 1                  $WORKDIR/topstream/service 127.0.0.1 9999 > /dev/null 2>&1
Aborted
ubuntu@0c3ca235dc4e:~/topstream$
```

Fig.14 Example of Trigger Vulnerability

The Fig.14 shown as above was captured by our oracle.

2.4.3.2. Code Analysis

Similar to ‘tsLIST’, as shown in Fig.15, the vulnerability is caused by the code that does not verify whether the string " Playing " + mname + " ...\r\n" exceeds the array tmpMovieStr size.

```c
int tsPLAY(char *params) {
  if (ts_state == LOGIN_SUCCESS) {
    if (params == NULL) {
      return sendResponse(client_sock, error520);
    }

    kliter_t(lmv) *it;
    it = kl_begin(movies);

    //Check if the movie list is empty
    if (it == kl_end(movies)) {
      return sendResponse(client_sock, error420);
    }

    if (isNumber(params)) {
      int index = atoi(params);
      char *mname = getMovieName(index);

      if (mname != NULL) {
        ki = getUser(active_user_name);
        user_info_t *user = kh_value(users, ki);

        //Check permission
        if (user->type >= getMovieType(index)) {
          sendResponse(client_sock, successcode);
          char tmpMovieStr[MAX_MOVIE_INFO_LENGTH];
          sprintf(tmpMovieStr, " Playing %s ...\r\n", mname);
          sendResponse(client_sock, tmpMovieStr);
        } else {
          sendResponse(client_sock, error470);
        }
```

Fig.15 Code of tsPLAY Function

2.4.3.3. Impact

Similar to ‘tsLIST’, this vulnerability also has potential threat in DoS attacks and other actions that would cause financial damage to the TopStream service owner.

2.4.3.4. How to Replay

We have created a seed replay_tsPLAY.dat that specifically triggers this vulnerability in the folder: **results/posc**

it can be reproduced using the following command:

cd topstream

make all

$WORKDIR/topstream/service 127.0.0.1 9999 > /dev/null 2>&1 & aflnet-replay $WORKDIR/results/posc/replay_tsPLAY.dat TOPSTREAM 8888 > /dev/null 2>&1 & $WORKDIR/topstream/topstream-asan 127.0.0.1 8888 127.0.0.1 9999

Alternatively, to replay, you can use telnet to connect to the server and manually enter the following command.

- **Terminal 1:**
  - cd $WORKDIR/topstream
  - make all
  - ./service 127.0.0.1 9999
- **Terminal 2:**
  - cd $WORKDIR/topstream
  - ./topstream 127.0.0.1 8888 127.0.0.1 9999
- **Terminal 3:**
  - cd $WORKDIR/topstream
  - telnet 127.0.0.1 8888
  - USER admin
  - PASS admin
  - DPIN 1111
  - LOAD bufferOverflow.txt
  - PLAY 3

2.4.4. Heap Overflow in tsUPDP

2.4.4.1. Vulnerability Description

Users can update their password by “UPDP newpass,newpass”, which is then stored in the users list. Although the password-length check is guaranteed when the administrator registers them, the system seems not to do any check about the length of the new password when users want to update their passwords. This oversight allows malicious users to exploit the UPDP vulnerability, using excessively long passwords to crash the system.

```
TopStream: receiving USER admin
TopStream: receiving PASS admin
TopStream: receiving DPIN 1111
TopStream: receiving UPDP
PDP newpass,newpassg UPDP newpassnewpass
TopStream: receiving UPDP admin,admin
TopStream: receiving REGU primary_user,primary_user
TopStream: receiving LOGO
TopStream: receiving USER primary_user
TopStream: receiving PASS primary_user
TopStream: receiving UPDP codecodecodecodecodecodecodecodeocdeocdecodecodecodecode,codecodecodecodecodecodecodecodeocdeocdecodeco
decodecode
=================================================================
==59040==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x6040000000f8 at pc 0x0000004ab12c bp 0x7fff0bc98570 sp 0x7fff
0bc97d20
WRITE of size 57 at 0x6040000000f8 thread T0
    #0 0x4ab12b  (/home/ubuntu/topstream/topstream-asan+0x4ab12b)
    #1 0x5160d0  (/home/ubuntu/topstream/topstream-asan+0x5160d0)
    #2 0x51c397  (/home/ubuntu/topstream/topstream-asan+0x51c397)
    #3 0x7f5aacc0fc86  (/lib/x86_64-linux-gnu/libc.so.6+0x21c86)
    #4 0x41a0f9  (/home/ubuntu/topstream/topstream-asan+0x41a0f9)

0x6040000000f8 is located 0 bytes to the right of 40-byte region [0x6040000000d0,0x6040000000f8)
allocated by thread T0 here:
    #0 0x4d9fb0  (/home/ubuntu/topstream/topstream-asan+0x4d9fb0)
    #1 0x512a03  (/home/ubuntu/topstream/topstream-asan+0x512a03)
    #2 0x516859  (/home/ubuntu/topstream/topstream-asan+0x516859)
    #3 0x51c397  (/home/ubuntu/topstream/topstream-asan+0x51c397)
    #4 0x7f5aacc0fc86  (/lib/x86_64-linux-gnu/libc.so.6+0x21c86)

SUMMARY: AddressSanitizer: heap-buffer-overflow (/home/ubuntu/topstream/topstream-asan+0x4ab12b)
Shadow bytes around the buggy address:
  0x0c087fff7fc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c087fff7fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c087fff7fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c087fff7ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c087fff8000: fa fa 00 00 00 00 00 fa fa fa 00 00 00 00 00 fa
=>0x0c087fff8010: fa fa fd fd fd fd fd fa fa fa 00 00 00 00 00[fa]
  0x0c087fff8020: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c087fff8030: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c087fff8040: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c087fff8050: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c087fff8060: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
```

Fig.16 Example of Trigger Vulnerability During Fuzzing Testing

Look at the user account name: primary_user in Fig.16, he is trying to make his new password become extremely long and finally makes the system crash. And the ERROR report is generated by address sanitizer when I replay the crash case.

2.4.4.2. Code Analysis

As shown in the following figure, the vulnerability is caused by the code that just verifies whether the two new inputs are the same but not to check if the length exceeds. That will provide the vulnerability that will let the maliciou users utilize the heap-buffer-overflow to paralyze the server.

```c
int tsUPDP(char *params) {
  if (ts_state == LOGIN_SUCCESS) {
    if (params == NULL) {
      return sendResponse(client_sock, error520);
    }

    //This command expect two arguments/parameters
    //e.g. UPDP strongpass,strongpass
    char **tokens = NULL;
    int count = 0;
    tokens = strSplit(params, ",", &count);

    if (count == 2) {
      if (strcmp(tokens[0], tokens[1])) {
        freeTokens(tokens, count);
        return sendResponse(client_sock, error450);
      }

      khint_t k = getUser(active_user_name);
      user_info_t *user = kh_value(users, k);
      strcpy(user->password, tokens[0]);
      sendResponse(client_sock, success300);
    } else {
      sendResponse(client_sock, error520);
    }

    freeTokens(tokens, count);
  } else {
    return sendResponse(client_sock, error530);
  }
  return 0;
}
```

Fig.17 Code of tsUPDP Function

2.4.4.3. Impact

Attackers can exploit the vulnerability to launch a Denial of Service attack by using excessively long strings, which leads to a heap overflow when updating passwords. Similar to 'tsLIST' and 'tsPLAY', this vulnerability also has potential threat in DoS attacks and other actions that would cause financial damage to the TopStream service owner.

2.4.4.4. How To Replay

- **Terminal 1:**
  - cd $WORKDIR/topstream
  - make all
  - ./service 127.0.0.1 9999
- **Terminal 2:**
  - cd $WORKDIR/topstream

- ./topstream 127.0.0.1 8888 127.0.0.1 9999
- **Terminal 3:**
  - cd $WORKDIR/topstream
  - telnet 127.0.0.1 8888
  - USER admin
  - PASS admin
  - DPIN 1111
  - REGU primary_user,primary_user
  - LOGO
  - USER primary_user
  - PASS primary_user
  - UPDP codecodecodecodecodecodecodecodecodeocdeocdecodecodecodecode,codecodecodecodecodecodecodecodecodecodecodecodecodecodecodecodecode

2.4.5. Unauthorized Access to Movies

2.4.5.1. Vulnerability Description

During the LOAD process, if an admin imports a movie file where the last line lacks a line break, the final character of that line will be replaced with '\0'. This alteration can modify the movie's type of the last move in the list. For example, considering a movie file as shown in Fig.18, if an admin loads it, the last movie, which should be of the 'VIP' type, but will be stored as the 'FREE' type in the movie list. This can result in unauthorized access issues, allowing 'FREE' users to view 'VIP' movies.

/home/ubuntu/topstream/Movies_by_group15.txt

```
A VIP movie,120,2
Another VIP(VIP),135,2
A BASIC movie,180,1
Would be VIP as the int is 12,120,12
MUSTBEVIP,120,2
```

Fig.18 Bad Format Movies File (Movies_by_group15.txt) to Trigger Vulnerability

```
USER admin
210 USER okay.
PASS admin
290 PASS okay. Please enter your PIN.
DPIN 1111
220 User logged in, proceed.
LOAD Movies_by_group15.txt
240 Movies loaded.
LIST
200 1. A VIP movie, 120, VIP
200 2. Another VIP(VIP), 135, VIP
200 3. A BASIC movie, 180, BASIC
200 4. Would be VIP as the int is 12, 120, VIP
200 5. MUSTBEVIP, 120, FREE
```

Fig.19 Results When Admin LOAD the Bad Format Movies File and Then LIST

The vulnerability was found when the fuzzer tried a movie file. After we checked against the content of the movies list. We found the movies which should be 'VIP' type but displayed as a 'FREE' type, and then we registered a new 'FREE' account to log in. This account can access and play this movie.

2.4.5.2. Code Analysis And Assertion

As shown in Fig.20, the vulnerability is caused by the code that does not verify whether the two movies' information string contains '\n', and replaces each line's last character with '\0', which may cause the type changing when the last line is not finished with '\n'.

```c
if (count == 3) {
  movie_info_t *m = (movie_info_t *) malloc(sizeof(movie_info_t));
  m->name = strdup(tokens[0]);
  if (!isNumber(tokens[1])) goto movie_format_error;
  if (atoi(tokens[1]) < 0) goto movie_format_error;
  m->length = atoi(tokens[1]);

  //set the newline character to null
  //to terminate the last string
  tokens[2][strlen(tokens[2]) - 1] = '\0';
  if (!isNumber(tokens[2])) goto movie_format_error;
  if (atoi(tokens[2]) < 0) goto movie_format_error;
  m->type = atoi(tokens[2]);
  goto movie_format_good;

format_error:
```

Fig.20 Code of tsLOAD Function

```c
//Otherwise, send back the movie list to the user
//in a formated string
int index = 0;
while(it != kl_end(movies)) {
  movie_info_t *m = kl_val(it);
  sendResponse(client_sock, successcode);
  char tmpMovieStr[MAX_MOVIE_INFO_LENGTH];
  char type[10];
  if (m->type == 0) {
    strcpy(type, "FREE");
  } else if (m->type == 1) {
    strcpy(type, "BASIC");
  } else {
    strcpy(type, "VIP");
  }
  sprintf(tmpMovieStr, " %d. %s, %d, %s\r\n", ++index, m->name, m->length, type);
  if(strcmp("MUSTBEVIP",m->name)==0){
    assert(strcmp(type, "VIP") == 0);
  }
  sendResponse(client_sock, tmpMovieStr);
  it = kl_next(it);
}
return 0;
```

Fig.21 Adding Assertion in tsLIST Function

```
TopStream: receiving LOAD Movies_by_group15.txt
TopStream: receiving LOAD bufferOverflou.txt
TopStream: receiving USER admin
TopStream: receiving PASS admin
TopStream: receiving DPIN 1111
TopStream: receiving REGU testForPass,testpass
TopStream: receiving PLAY 5
TopStream: receiving LOGO
TopStream: receiving LOAD movies.txt
TopStream: receiving USER admin
TopStream: receiving PASS admin
TopStream: receiving DPIN 1111
TopStream: receiving LOAD
TopStream: receiving LOGO
TopStream: receiving USER@
TopStream: receiving PASS testpass
TopStream: receiving LOAD movies.txt
TopStream: receiving LOGO
TopStream: receiving USER admin
TopStream: receiving PASS admin
TopStream: receiving DPIN 1111
TopStream: receiving LOAD movieeeeees.txt
TopStream: receiving LOAD movies1ForError560.txt
TopStream: receiving LOAD movies2ForError560.txt
TopStream: receiving LOAD moZies.txt
TopStream: receiving LOAD mix.txt
TopStream: receiving LOAD error_length_not_number.txt
TopStream: receiving LOAD error_length_negative.tx?
TopStream: receiving LOAD error_type_not_number.txt
nt_dataam: receiving LOAD error_type_negative.txt
LOAD error_insufficient_data.txt
TopStream: receiving LOAD error_too_much_data.txt
TopStream: receiving LOGO
TopStream: receiving LIST
topstream-asan: topstream.c:547: int tsLIST(char *): Assertion `strcmp(type, "VIP") == 0' failed.
Aborted
[1]-  Done                    $WORKDIR/topstream/service 127.0.0.1 9999 > /dev/null 2>&1
```

Fig.22 The Test Case Will Be Crashed During the Fuzzing When the Assertion Is Not Hold

2.4.5.3. Impact

Due to the lack of detailed specifications on the format of the file content storing movie names throughout the project, incorrect .txt files can easily be generated when new .txt files are added to the system.

The movie that should be exclusive to "VIP" users can now be accessed by "FREE" users. This means that the TopStream service owner loses the expected revenue, as there is no need to pay for premium content. For services that rely on membership fees or payments from more advanced users, this vulnerability could lead to significant financial losses. This is because users may share the existence of this vulnerability, resulting in more users choosing to sign up for free instead of paying to become a "VIP".

2.4.5.4. How to Replay

- **Terminal 1:**
  - cd $WORKDIR/topstream
  - make all
  - ./service 127.0.0.1 9999
- **Terminal 2:**
  - cd $WORKDIR/topstream
  - ./topstream 127.0.0.1 8888 127.0.0.1 9999

- **Terminal 3:**
  - cd $WORKDIR/topstream
  - telnet 127.0.0.1 8888
  - USER admin
  - PASS admin
  - DPIN 1111
  - LOAD Movies_by_group15.txt
  - LIST

If you performed those instructions by using the original topstream.c in the *topstream* folder, you will see the results as shown in Fig.23.

```
USER admin
210 USER okay.
PASS admin
290 PASS okay. Please enter your PIN.
DPIN 1111
220 User logged in, proceed.
LOAD Movies_by_group15.txt
240 Movies loaded.
LIST
200 1. A VIP movie, 120, VIP
200 2. Another VIP(VIP), 135, VIP
200 3. A BASIC movie, 180, BASIC
200 4. Would be VIP as the int is 12, 120, VIP
200 5. MUSTBEVIP, 120, FREE
```

Fig.23 Results If Run Original topstream.c

2.5. Improvements and Adjustments

Initially, to achieve higher line and branch coverage, one of our team members meticulously examined the source code. He ensured that the instructions written in the seeds could trigger as many branches as possible, thus executing the maximum number of code lines.

Even though we hit our coverage goal, the big seed files made our fuzz testing run too slow, making it hard to find problems. So, he and the team decided to break down the big seed files into smaller ones based on their functions, like tsLIST, tsLOAD, and tsUPDP (the splitted seeds for fuzzing testing are already submitted to the results/seed_corpus/raw folder). This made our testing a lot faster, about five times quicker than before. Plus, it lets us focus on testing specific parts and find issues that could crash the system in shrinked regions.

However, a predicament emerged: Owing to the inherent randomness of fuzz testing, not every command modified through the fuzzer was recognized as valid by the topstream system. Invalid commands rendered the effort futile both in terms of expanding coverage and vulnerabilities discoverage. To deal with that, a team member developed a dictionary (top.dict, it’s already submitted to the results/others folder) to furnish a more structured depiction of the foundational syntax. Upon integrating this dictionary, we witnessed a surge in the rate of discovering new paths, approximating nearly 300 paths within a mere hour. The more paths, the more probability of digging out vulnerabilities.

To further accelerating the progress of discoverge vulnerabilities, one of the teammate also make 2 or 3 commands("LOAD Movies_by_group15.txt", "LOAD bufferOverflow.txt" and "PLAY 3") that he thought will trigger the vulnerabilities that become the content of the dictionary. That would increase the chance for the fuzzer to input commands we're really interested in, leading to a better shot at finding issues that could crash the system.

3. Reflections and Analysis

3.1. Advantages and Limitations of AFLNet

AFLNet is an extension of the popular AFL fuzzer, specifically designed for testing and fuzzing network protocols and networked software. It's a valuable tool for finding vulnerabilities in network protocols and networked applications.

3.1.1. Advantages

- AFLNet combines techniques of mutation–based and instrumental strategies such as using dictionaries, providing excellent guidance to code coverage. This makes the process of achieving high coverage without requiring much priority knowledge and long setup times.

- In addition, AFLNet uses a state-based mutation mechanism, which greatly improves the intelligence and efficiency of fuzzy testing[1].

- Also, AFLNet allows executing with minimum size seeds, so we can start with several small test cases but still have characteristics of triggering vulnerabilities. These test cases are more concise and intuitive to help to gain a deeper understanding and fix potential problems.

3.1.2. Limitations

- AFLNet always requires a somewhat tedious setup before starting the fuzzing, especially for beginners. This is because newcomers not only need to figure out how to make the program under test acceptable to AFLNet, but also cannot modify the code being tested during runtime. This necessitates a thorough check of all artifacts (codes, dictionary, output folders) that will be used during testing, each time before setting up.
- About 'Unique Crashed'. It seems that the crash will be recognised as unique based on the mechanism that we are not pretty sure. AFL tries to induce crashes by randomly altering the input, which might lead to the same underlying issue being triggered in multiple different ways.

References

[1] V. -T. Pham, M. Böhme and A. Roychoudhury, "AFLNET: A Greybox Fuzzer for Network Protocols," *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, Porto, Portugal, 2020, pp. 460-465, doi: 10.1109/ICST46399.2020.00062.