SWEN90006
Security & Software Testing

Assignment 2

Group 45
Chengyi Huang, Mao Rui, Ruoyu Lu

1. Initial Fuzzing Setup

First, build and run the docker container using the commands provided on github:

```
docker build . -t swen90006-assignment2
docker run -it swen90006-assignment2
```

Modify the topstream.c file, add test_mode in *generatePIN()*, the result directly returns 1234 when turned on, to speed up the testing process, and then use the command to compile the source file:

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

Create a new restart.sh script file as shown below in the /home/ubuntu/topstream folder so that the telco server can be restarted from an excited state during testing:

```
#!/bin/bash
pkill -9 service
./service 127.0.0.1 9999 >> /home/ubuntu/service.log 2>&1 &
```

Add permissions to it using `chmod +x restart.sh`. Afterwards, add the dictionary file to home/ubuntu/topstream and create in-dir and out-dir. Start fuzzing using the command line:

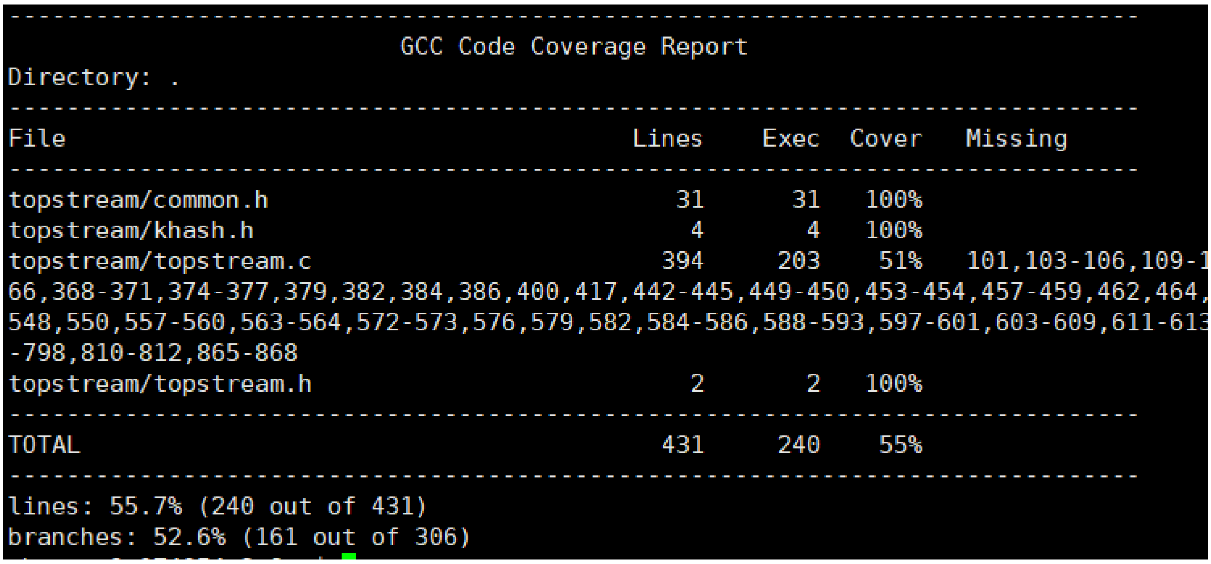
```
afl-fuzz -i /home/ubuntu/in-dir -o /home/ubuntu/out-dir -N
tcp://127.0.0.1/8888 -c /home/ubuntu/topstream/restart.sh -x
/home/ubuntu/topstream/rtsp.dict -P TOPSTREAM -D 10000 -t 800 -m 30 -q 3
-s 3 -E -K -R /home/ubuntu/topstream/topstream-fuzz 127.0.0.1 8888
127.0.0.1 9999
```

At the end of fuzzing, we created a result.sh script file as follows. After running the script, use the `gcovr -r . -s` command to read the results of the coverage test.

```
cd $WORKDIR/topstream
for f in $(echo ../results/topstream_out/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. Achieving High Code Coverage

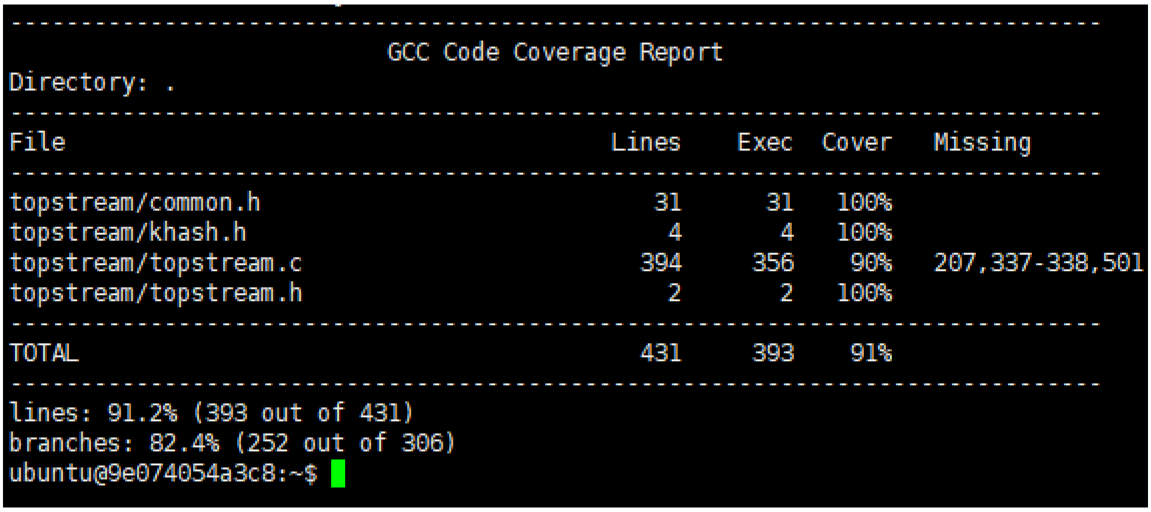
initially, we used a relatively simple seed corpus, seed1 (see appendix).
But the test results were not favorable:



```
------------------------------------------------------------------------------
                           GCC Code Coverage Report
Directory: .
------------------------------------------------------------------------------
File                                       Lines    Exec  Cover   Missing
------------------------------------------------------------------------------
topstream/common.h                            31      31   100%
topstream/khash.h                              4       4   100%
topstream/topstream.c                        394     203    51%   101,103-106,109-1
66,368-371,374-377,379,382,384,386,400,417,442-445,449-450,453-454,457-459,462,464,
548,550,557-560,563-564,572-573,576,579,582,584-586,588-593,597-601,603-609,611-613
-798,810-812,865-868
topstream/topstream.h                          2       2   100%
------------------------------------------------------------------------------
TOTAL                                        431     240    55%
------------------------------------------------------------------------------
lines: 55.7% (240 out of 431)
branches: 52.6% (161 out of 306)
```

In order to ensure branch coverage and line coverage, our seed corpus uses a full set of commands, as seed2 (see appendix). In the dictionary file, we use some keywords that may cause problems for input (see appendix).

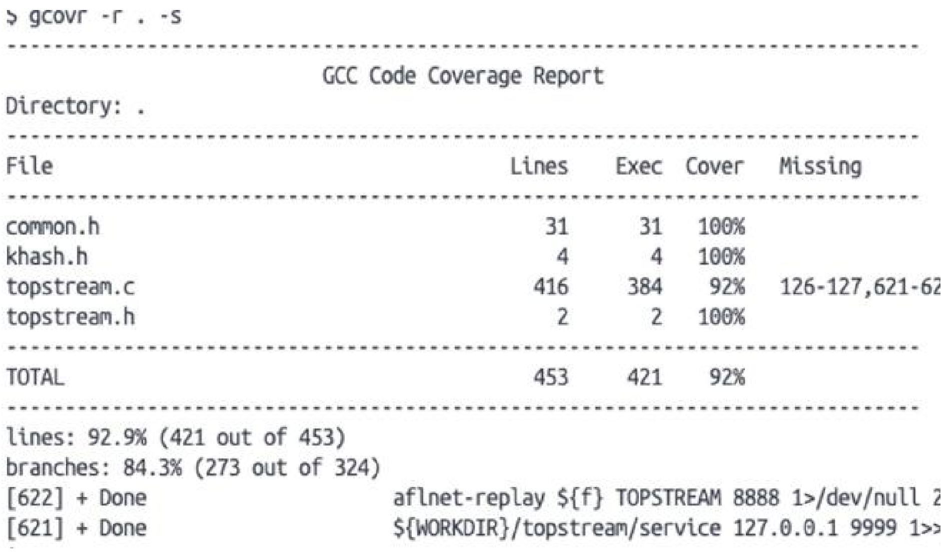
This time, we achieved a high coverage rate. However, this is from running 26h fuzzing with a windows computer.



```
------------------------------------------------------------------------------
                           GCC Code Coverage Report
Directory: .
------------------------------------------------------------------------------
File                                       Lines    Exec  Cover   Missing
------------------------------------------------------------------------------
topstream/common.h                            31      31   100%
topstream/khash.h                              4       4   100%
topstream/topstream.c                        394     356    90%   207,337-338,501
topstream/topstream.h                          2       2   100%
------------------------------------------------------------------------------
TOTAL                                        431     393    91%
------------------------------------------------------------------------------
lines: 91.2% (393 out of 431)
branches: 82.4% (252 out of 306)
ubuntu@9e074054a3c8:~$
```

In the process, we discovered an interesting phenomenon, which is that using a macbook (even if it's an intel chip) can make fuzzing exponentially faster. So, for the final test, we switched to a macbook and made the following adjustments:

Added movies1.txt and movies2.txt by adjusting dictionary and seed contents based on uncovered line number information. Beside, we adjust the seed to seed3(see appendix).



```
$ gcovr -r . -s
------------------------------------------------------------------------------
                           GCC Code Coverage Report
Directory: .
------------------------------------------------------------------------------
File                                       Lines    Exec  Cover   Missing
------------------------------------------------------------------------------
common.h                                      31      31   100%
khash.h                                        4       4   100%
topstream.c                                  416     384    92%   126-127,621-62
topstream.h                                    2       2   100%
------------------------------------------------------------------------------
TOTAL                                        453     421    92%
------------------------------------------------------------------------------
lines: 92.9% (421 out of 453)
branches: 84.3% (273 out of 324)
[622] + Done                      aflnet-replay ${f} TOPSTREAM 8888 1>/dev/null 2
[621] + Done                      ${WORKDIR}/topstream/service 127.0.0.1 9999 1>>
```

3. Vulnerability Discovery

Traverse all the test cases that caused the crash to see the cause of the crash using AddressSanitizer

```
for f in $(echo ../results/topstream_out/replayable-crashes/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-asan 127.0.0.1 8888 127.0.0.1 9999;
done
```

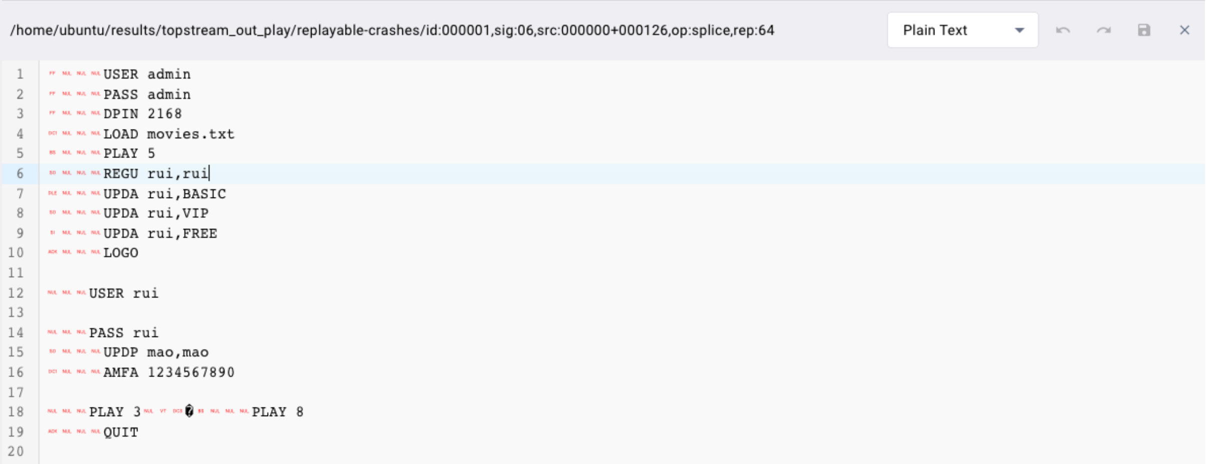
Traverse all the test cases in the folder replayable-queue using AddressSanitizer.

```
for f in $(echo ../results/topstream_out/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-asan 127.0.0.1 8888 127.0.0.1 9999;
done
```

3.1 Crash 1. User playing permission issues

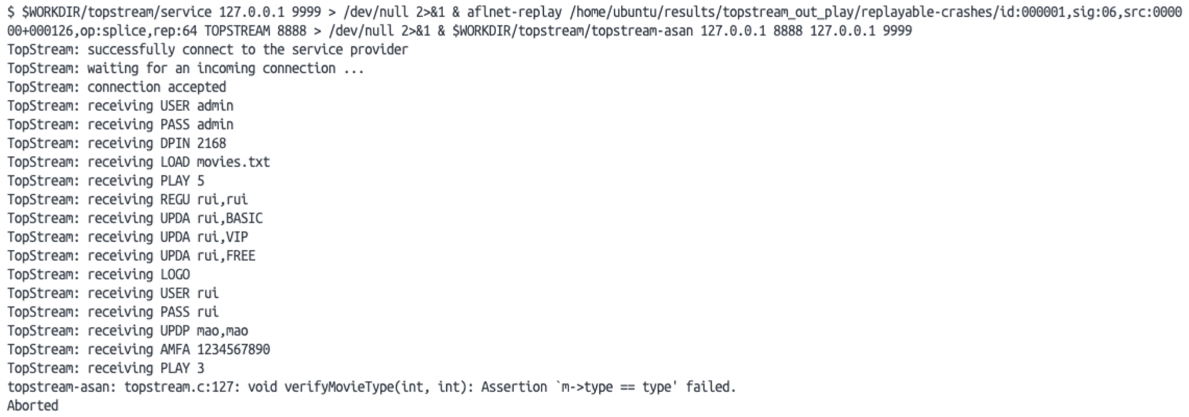
Type of vulnerability

There is an error in determining the playing permission of the movie: e.g. Users with Basic permission can play the third movie with VIP permission.



/home/ubuntu/results/topstream_out_play/replayable-crashes/id:000001,sig:06,src:000000+000126,op:splice,rep:64

```
FF NUL NUL NUL USER admin
FF NUL NUL NUL PASS admin
FF NUL NUL NUL DPIN 2168
DC1 NUL NUL NUL LOAD movies.txt
BS NUL NUL NUL PLAY 5
SO NUL NUL NUL REGU rui,rui
DLE NUL NUL NUL UPDA rui,BASIC
SO NUL NUL NUL UPDA rui,VIP
SI NUL NUL NUL UPDA rui,FREE
ACK NUL NUL NUL LOGO

NUL NUL NUL USER rui

NUL NUL NUL PASS rui
SO NUL NUL NUL UPDP mao,mao
DC1 NUL NUL NUL AMFA 1234567890

NUL NUL NUL PLAY 3 NUL VT DC3 � BS NUL NUL NUL PLAY 8
ACK NUL NUL NUL QUIT

```



```
$ $WORKDIR/topstream/service 127.0.0.1 9999 > /dev/null 2>&1 & aflnet-replay /home/ubuntu/results/topstream_out_play/replayable-crashes/id:000001,sig:06,src:0000
00+000126,op:splice,rep:64 TOPSTREAM 8888 > /dev/null 2>&1 & $WORKDIR/topstream/topstream-asan 127.0.0.1 8888 127.0.0.1 9999
TopStream: successfully connect to the service provider
TopStream: waiting for an incoming connection ...
TopStream: connection accepted
TopStream: receiving USER admin
TopStream: receiving PASS admin
TopStream: receiving DPIN 2168
TopStream: receiving LOAD movies.txt
TopStream: receiving PLAY 5
TopStream: receiving REGU rui,rui
TopStream: receiving UPDA rui,BASIC
TopStream: receiving UPDA rui,VIP
TopStream: receiving UPDA rui,FREE
TopStream: receiving LOGO
TopStream: receiving USER rui
TopStream: receiving PASS rui
TopStream: receiving UPDP mao,mao
TopStream: receiving AMFA 1234567890
TopStream: receiving PLAY 3
topstream-asan: topstream.c:127: void verifyMovieType(int, int): Assertion `m->type == type' failed.
Aborted
```

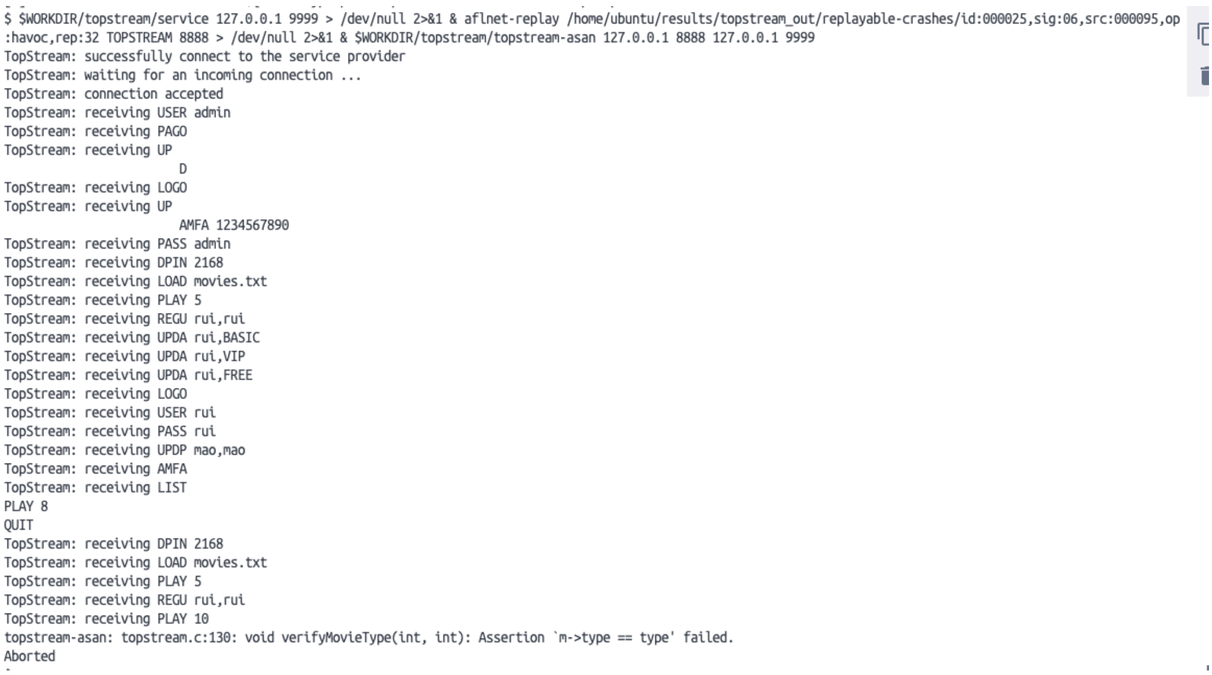
How they were discovered

Since it can't play the second part, we've initially determined that it's due to an error in the function that gets permission to play the movie.

By means of Metamorphic testing, a function was added to topstream.c to re-verify the movie playback permissions: *VerifyMovieType(int index, int type)*, using an assertion to determine if the original code returned the same value as the new function obtained. The code was then recompiled with the make clean all command.

```
/**
* Verify movie type gotten
*/
void verifyMovieType(int index, int type) {
kliter_t(lmv) *it;
it = kl_begin(movies);
for (int i = 1; i < index; i++) {
if (it != kl_end(movies)) {
it = kl_next(it);
}
}
if (it == kl_end(movies)) {
assert(-1 == type);
return;
}
movie_info_t *m = kl_val(it);
assert(m->type == type);
}
```

Subsequently, PLAY 3, PLAY 6, and PLAY 10 were added to the dictionary, which are more likely to result in a FAILURE based on the principle of boundary values.
The following figure shows the presence of the PLAY 10 command:

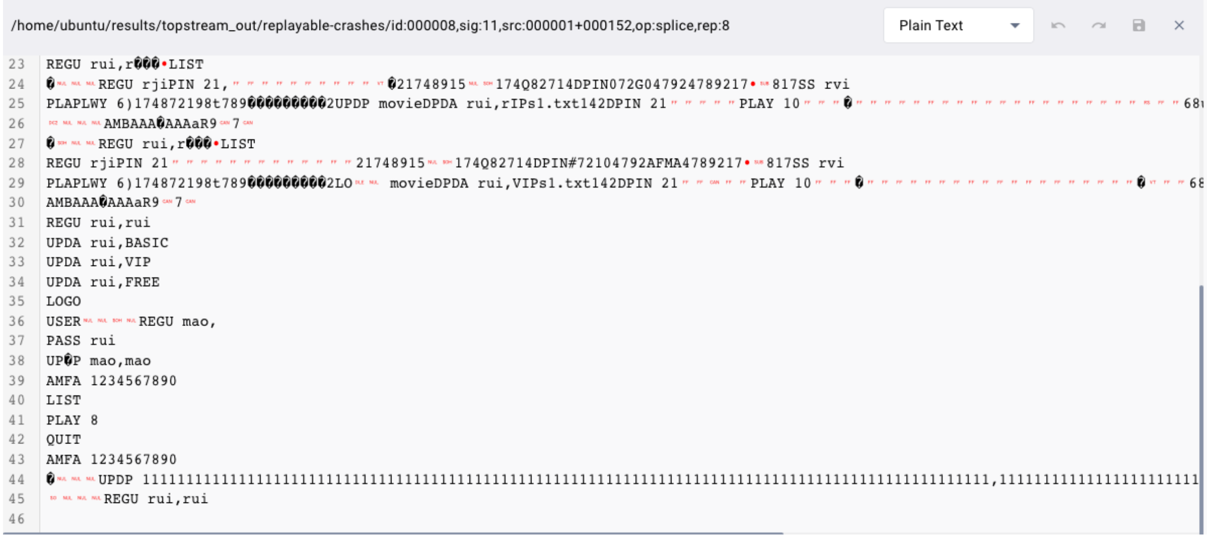


```
$ $WORKDIR/topstream/service 127.0.0.1 9999 > /dev/null 2>&1 & aflnet-replay /home/ubuntu/results/topstream_out/replayable-crashes/id:000025,sig:06,src:000095,op
:havoc,rep:32 TOPSTREAM 8888 > /dev/null 2>&1 & $WORKDIR/topstream/topstream-asan 127.0.0.1 8888 127.0.0.1 9999
TopStream: successfully connect to the service provider
TopStream: waiting for an incoming connection ...
TopStream: connection accepted
TopStream: receiving USER admin
TopStream: receiving PAGO
TopStream: receiving UP
                        D
TopStream: receiving LOGO
TopStream: receiving UP
                        AMFA 1234567890
TopStream: receiving PASS admin
TopStream: receiving DPIN 2168
TopStream: receiving LOAD movies.txt
TopStream: receiving PLAY 5
TopStream: receiving REGU rui,rui
TopStream: receiving UPDA rui,BASIC
TopStream: receiving UPDA rui,VIP
TopStream: receiving UPDA rui,FREE
TopStream: receiving LOGO
TopStream: receiving USER rui
TopStream: receiving PASS rui
TopStream: receiving UPDP mao,mao
TopStream: receiving AMFA
TopStream: receiving LIST
PLAY 8
QUIT
TopStream: receiving DPIN 2168
TopStream: receiving LOAD movies.txt
TopStream: receiving PLAY 5
TopStream: receiving REGU rui,rui
TopStream: receiving PLAY 10
topstream-asan: topstream.c:130: void verifyMovieType(int, int): Assertion `m->type == type' failed.
Aborted
```

3.2 Crash 2. Buffer overflow

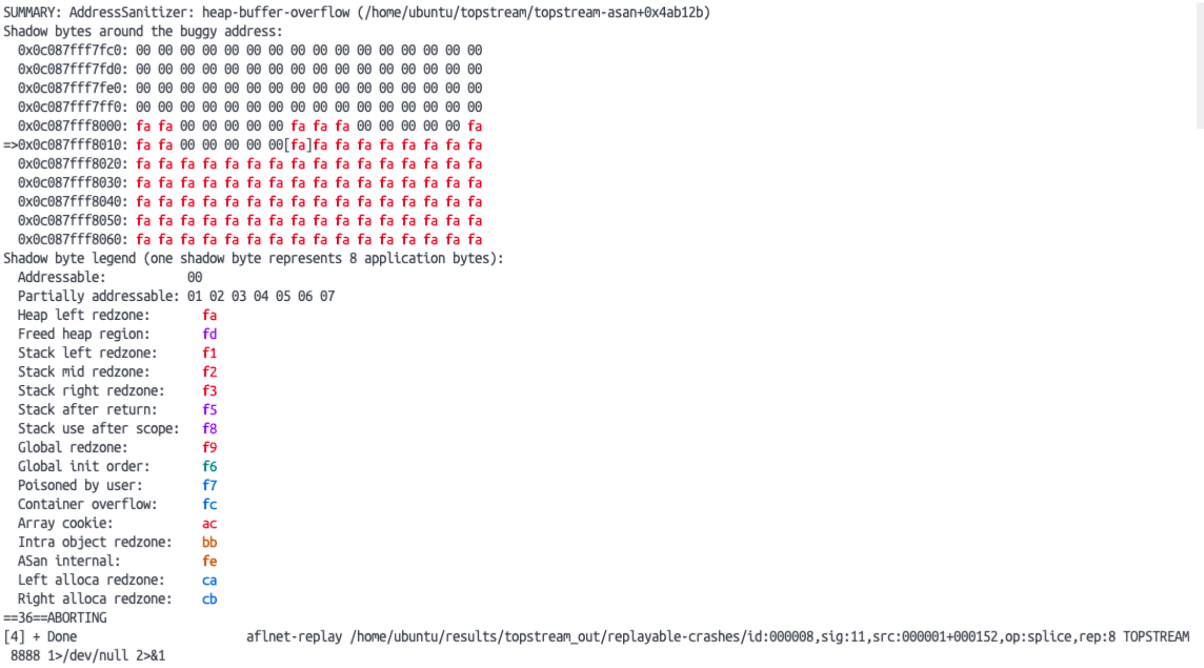
Type of vulnerability

When updating a password, if the new password is too long, it will cause a buffer overflow.



/home/ubuntu/results/topstream_out/replayable-crashes/id:000008,sig:11,src:000001+000152,op:splice,rep:8

```
REGU rui,r���•LIST
�NUL NUL NULREGU rjiPIN 21,FF FF FF FF FF FF FF FF FF FF VT�21748915NUL SOH174Q82714DPIN072G047924789217•SUB817SS rvi
PLAPLWY 6)174872198t789����������2UPDP movieDPDA rui,rIPs1.txt142DPIN 21FF FF FF FF FF PLAY 10FF FF FF�FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF RS FF FF 68
DC2 NUL NUL NUL AMBAAA�AAAaR9CAN7CAN
�SOH NUL NULREGU rui,r���•LIST
REGU rjiPIN 21FF FF FF FF FF FF FF FF FF FF FF FF FF21748915NUL SOH174Q82714DPIN#72104792AFMA4789217•SUB817SS rvi
PLAPLWY 6)174872198t789����������2LODLE NUL movieDPDA rui,VIPs1.txt142DPIN 21FF FF CAN FF FF PLAY 10FF FF FF�FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF�VT FF FF 68
AMBAAA�AAAaR9CAN7CAN
REGU rui,rui
UPDA rui,BASIC
UPDA rui,VIP
UPDA rui,FREE
LOGO
USERNUL NUL SOH NULREGU mao,
PASS rui
UP�P mao,mao
AMFA 1234567890
LIST
PLAY 8
QUIT
AMFA 1234567890
�NUL NUL NULUPDP 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111,11111111111111111111111
SO NUL NUL NULREGU rui,rui

```



```
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 00 00 00 00 00[fa]fa fa fa fa fa fa fa 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):
  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
==36==ABORTING
[4] + Done                    aflnet-replay /home/ubuntu/results/topstream_out/replayable-crashes/id:000008,sig:11,src:000001+000152,op:splice,rep:8 TOPSTREAM
 8888 1>/dev/null 2>&1
```

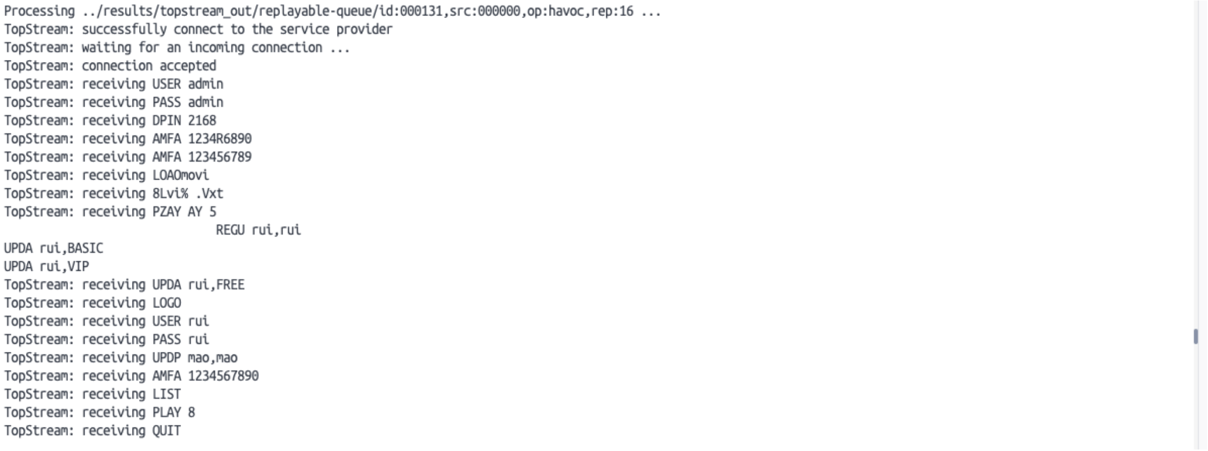
How they were discovered

Used AddressSanitizer to get the cause of the crash - buffer overflow. Observing the communication messages received by the TopStream server, it was found that it was due to a long password when updating the password.

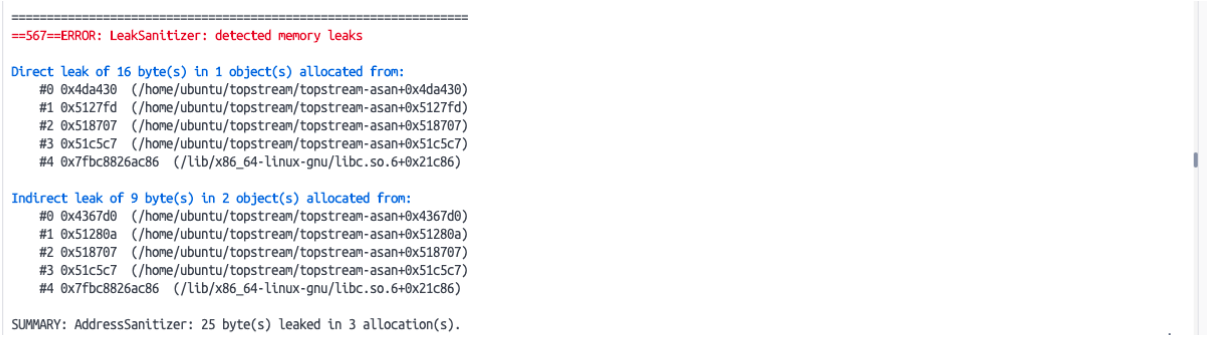
3.3 Crash 3. Memory leakage

Type of vulnerability

As shown in the figure:



```
Processing ../results/topstream_out/replayable-queue/id:000131,src:000000,op:havoc,rep:16 ...
TopStream: successfully connect to the service provider
TopStream: waiting for an incoming connection ...
TopStream: connection accepted
TopStream: receiving USER admin
TopStream: receiving PASS admin
TopStream: receiving DPIN 2168
TopStream: receiving AMFA 1234R6890
TopStream: receiving AMFA 123456789
TopStream: receiving LOAOmovi
TopStream: receiving 8Lvi% .Vxt
TopStream: receiving PZAY AY 5
                              REGU rui,rui
UPDA rui,BASIC
UPDA rui,VIP
TopStream: receiving UPDA rui,FREE
TopStream: receiving LOGO
TopStream: receiving USER rui
TopStream: receiving PASS rui
TopStream: receiving UPDP mao,mao
TopStream: receiving AMFA 1234567890
TopStream: receiving LIST
TopStream: receiving PLAY 8
TopStream: receiving QUIT
```



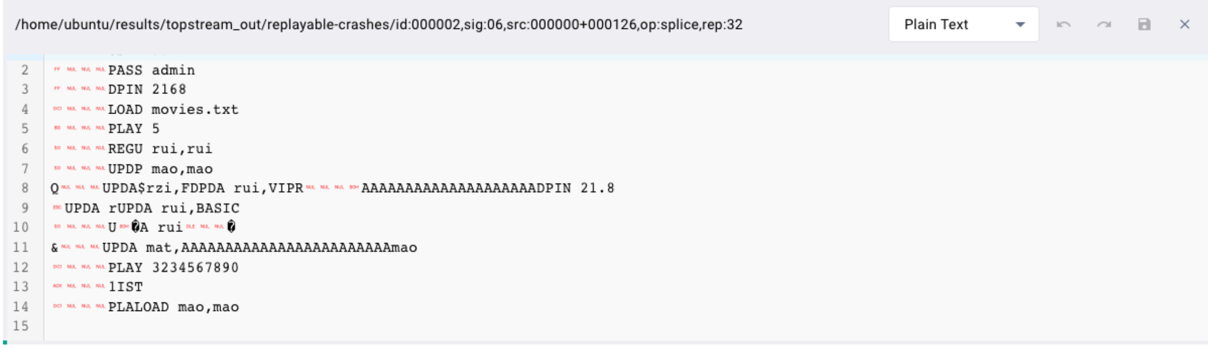
```
=================================================================
==567==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 16 byte(s) in 1 object(s) allocated from:
    #0 0x4da430  (/home/ubuntu/topstream/topstream-asan+0x4da430)
    #1 0x5127fd  (/home/ubuntu/topstream/topstream-asan+0x5127fd)
    #2 0x518707  (/home/ubuntu/topstream/topstream-asan+0x518707)
    #3 0x51c5c7  (/home/ubuntu/topstream/topstream-asan+0x51c5c7)
    #4 0x7fbc8826ac86  (/lib/x86_64-linux-gnu/libc.so.6+0x21c86)

Indirect leak of 9 byte(s) in 2 object(s) allocated from:
    #0 0x4367d0  (/home/ubuntu/topstream/topstream-asan+0x4367d0)
    #1 0x51280a  (/home/ubuntu/topstream/topstream-asan+0x51280a)
    #2 0x518707  (/home/ubuntu/topstream/topstream-asan+0x518707)
    #3 0x51c5c7  (/home/ubuntu/topstream/topstream-asan+0x51c5c7)
    #4 0x7fbc8826ac86  (/lib/x86_64-linux-gnu/libc.so.6+0x21c86)

SUMMARY: AddressSanitizer: 25 byte(s) leaked in 3 allocation(s).
```

How they were discovered

Traversing the replayable-queue folder with AddressSanitizer detects a large memory overflow.

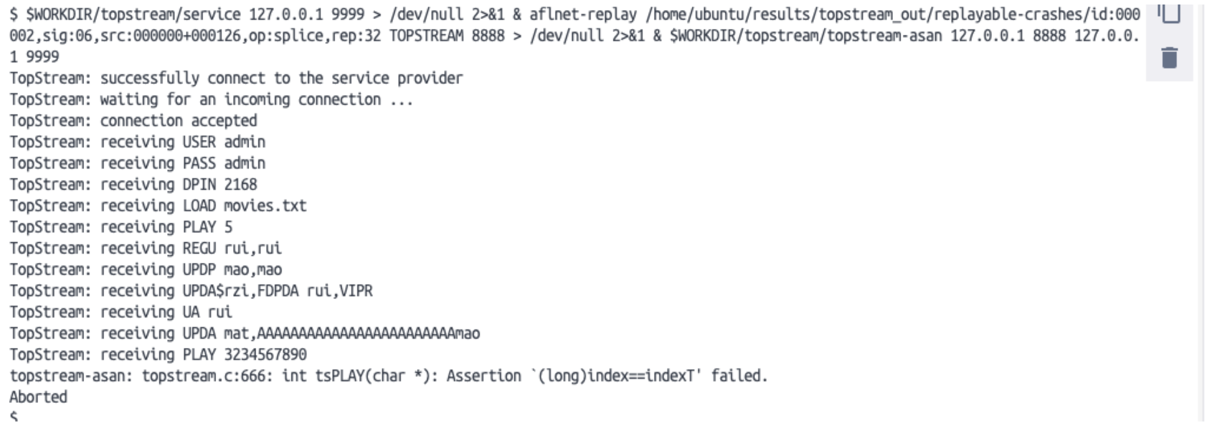
3.4 Crash 4. movie playback

When using `PLAY movie_id` to play a movie, if the movie ID exceeds the limit of the int type, it won't result in an error; it will play the first movie.



/home/ubuntu/results/topstream_out/replayable-crashes/id:000002,sig:06,src:000000+000126,op:splice,rep:32

```
␌␀␀␀PASS admin
␌␀␀␀DPIN 2168
␑␀␀␀LOAD movies.txt
␈␀␀␀PLAY 5
␎␀␀␀REGU rui,rui
␎␀␀␀UPDP mao,mao
Q␀␀␀UPDA$rzi,FDPDA rui,VIPR␀␀␀␁AAAAAAAAAAAAAAAAAAAAAADPIN 21.8
␛UPDA rUPDA rui,BASIC
␎␀␀␀U␁�A rui␐␀␀�
&␀␀␀UPDA mat,AAAAAAAAAAAAAAAAAAAAAAAAAmao
␑␀␀␀PLAY 3234567890
␆␀␀␀1IST
␑␀␀␀PLALOAD mao,mao

```



```
$ $WORKDIR/topstream/service 127.0.0.1 9999 > /dev/null 2>&1 & aflnet-replay /home/ubuntu/results/topstream_out/replayable-crashes/id:000
002,sig:06,src:000000+000126,op:splice,rep:32 TOPSTREAM 8888 > /dev/null 2>&1 & $WORKDIR/topstream/topstream-asan 127.0.0.1 8888 127.0.0.
1 9999
TopStream: successfully connect to the service provider
TopStream: waiting for an incoming connection ...
TopStream: connection accepted
TopStream: receiving USER admin
TopStream: receiving PASS admin
TopStream: receiving DPIN 2168
TopStream: receiving LOAD movies.txt
TopStream: receiving PLAY 5
TopStream: receiving REGU rui,rui
TopStream: receiving UPDP mao,mao
TopStream: receiving UPDA$rzi,FDPDA rui,VIPR
TopStream: receiving UA rui
TopStream: receiving UPDA mat,AAAAAAAAAAAAAAAAAAAAAAAAAmao
TopStream: receiving PLAY 3234567890
topstream-asan: topstream.c:666: int tsPLAY(char *): Assertion `(long)index==indexT' failed.
Aborted
$
```

How they were discovered

Through the Metamorphic testing method, code for verifying the obtained movie IDs has been added in topstream.c.

```
long indexT = strtol(params, NULL, 10);
assert((long)index==indexT);
```

Use assertions to determine if it is consistent. Subsequently recompile the code with the `make clean all` command.

Adjust the position of the play command in the seed to a forward position to make it easier to obfuscate valid play commands.

4. Reflections

During the testing process, we found that the test speed was too slow, and the exec speed in the AFL interface could only reach a speed of about 7-8 sometimes. Therefore, we took the following measures to speed up the test:

- Turning on -d nearly doubled the speed, but it also resulted in a lower coverage than the standard mode, so we removed -d. In this way, initialization code in each fuzzing run could be closer to the actual usage scenario of the target program
- Adjusted -t, at first we adjusted the -t time to 200ms, but this resulted in too many time out test cases. Therefore, we gradually increased the value to 800ms to make the testing process not waste time and also to ensure that the number of time out is reduced;

- Adjusted -m to shorten the subroutine memory requirement to avoid wasting resources and reducing efficiency.
- Tests were performed in multiple parallel using a single system using the -M and -S commands. However, by doing so, the speed of the tests was not significantly improved and the speed of each test coincidentally decreased when using one -M and three -S windows. We think this result may be because multiple instances cause CPU or disk caching to become less efficient, which in turn affects speed.
- we use -C, which means crash exploration mode, is more prone to crash.
- We use -s, which is helpful for making fuzzing runs deterministic. We have found after several tests that it is difficult to reproduce all the crashes in the same test when using -s 2. So -s is considered to be effective for reproducing crashes.
- Documents imported into Docker require a change in access permissions.

```
sudo chmod 644 /home/ubuntu/results/seed_corpus/seed1.raw
```

5. Analysis of AFLNet

5.1 advantages

- AFLNet is easy to use. It doesn't require a complicated setup. we can get it up and running with a few command-line instructions.
- AFLNet provides useful error messages and debugging options, making it easier to identify and solve any issues you might encounter during fuzzing.
- AFLNet can be easily integrated into automated testing pipelines, requiring minimal manual intervention once it is properly configured.

5.2 limitations

- The quality of the initial seed files can greatly impact the fuzzer's effectiveness. Poor initial seeds may lead to less effective fuzzing.
- If the initial seed corpus does not adequately cover the input space, it may take a long time to find a reproducible bug.
- In some cases, even if the crash is found, it still doesn't reproduce the crash very well.

6. Appendix

seed1

```
user admin
pass admin
dpin 1234
regu newuser,newpass
```

```
logo

user newuser
pass newpass
play 99999999999999999999999999999999999999
logo
quit
```

seed2

```
USER admin
PASS admin
DPIN 1234
REGU newuser,newpass
UPDA newuser,VIP
LOAD moives.txt
LOGO

USER newuser
PASS newpass
AMFA 0123456789
UPDP longpassword,longpassword
PLAY 9
LOGO
QUIT
```

seed3

```
USER admin
PASS admin
DPIN 1370
REGU rui,rui
UPDA rui,BASIC
LOAD movies.txt
LOGO
USER rui
PASS rui
UPDP mao,mao
AMFA 1234567890
LIST
PLAY 6
LOGO
```

```
QUIT
```

dict1

```
"USER"
"PASS"
"DPIN"
"REGU"
"LIST"
"UPDA"
"UPDP"
"AMFA"
"LOAD"
"PLAY"
"-1"
"aaaaaaaaaaaaaaaaaaaa"
"99999999999999999999"
" "
"$"
","
"\x00"
```