

Make iOS App more Robust and Security through Fuzzing

Wei Wang & Zhaowei Wang

2016-10

About us

- ID: Proteas, Shrek_wzw
- Working at: Qihoo 360 Nirvan Team
- Focused on: iOS and macOS Security Research
- Twitter: @ProteasWang, @Shrek_wzw

Agenda

- Status of iOS App Security Development Lifecycle
- Why Using AFL to Fuzz App during Development
- Build AFL Toolchain
- Characteristics and Remote Attacking Surfaces of iOS App
- Fuzz iOS App
- Fuzz 3rd Party Libraries

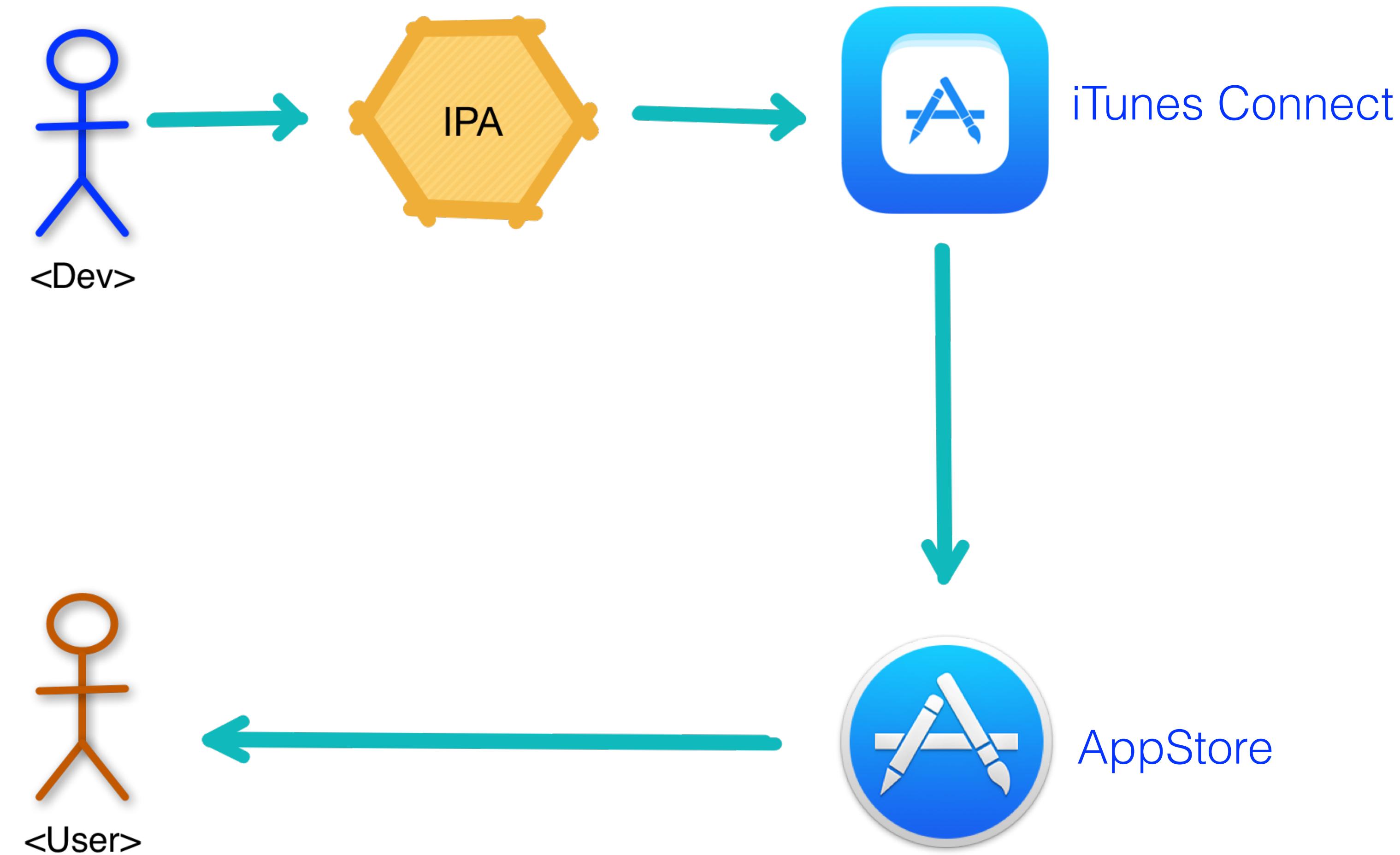
Status of iOS App Security Development Lifecycle

- There are about 2 million Apps on Apple AppStore as of June 2016
- Most developed by individual developers or small companies
- For most of those developers or companies, there is no security engineer to protect the Apps
- So the SDL may be like this:

Status of iOS App Security Development Lifecycle

- After development and testing, the IPA is submitted to iTunes Connect
- Then Apple will review the submitted IPA
- After reviewing, the IPA is distributed to AppStore
- Users now can search and download the App from AppStore
- As the following image shows, there is no security related node in the whole process

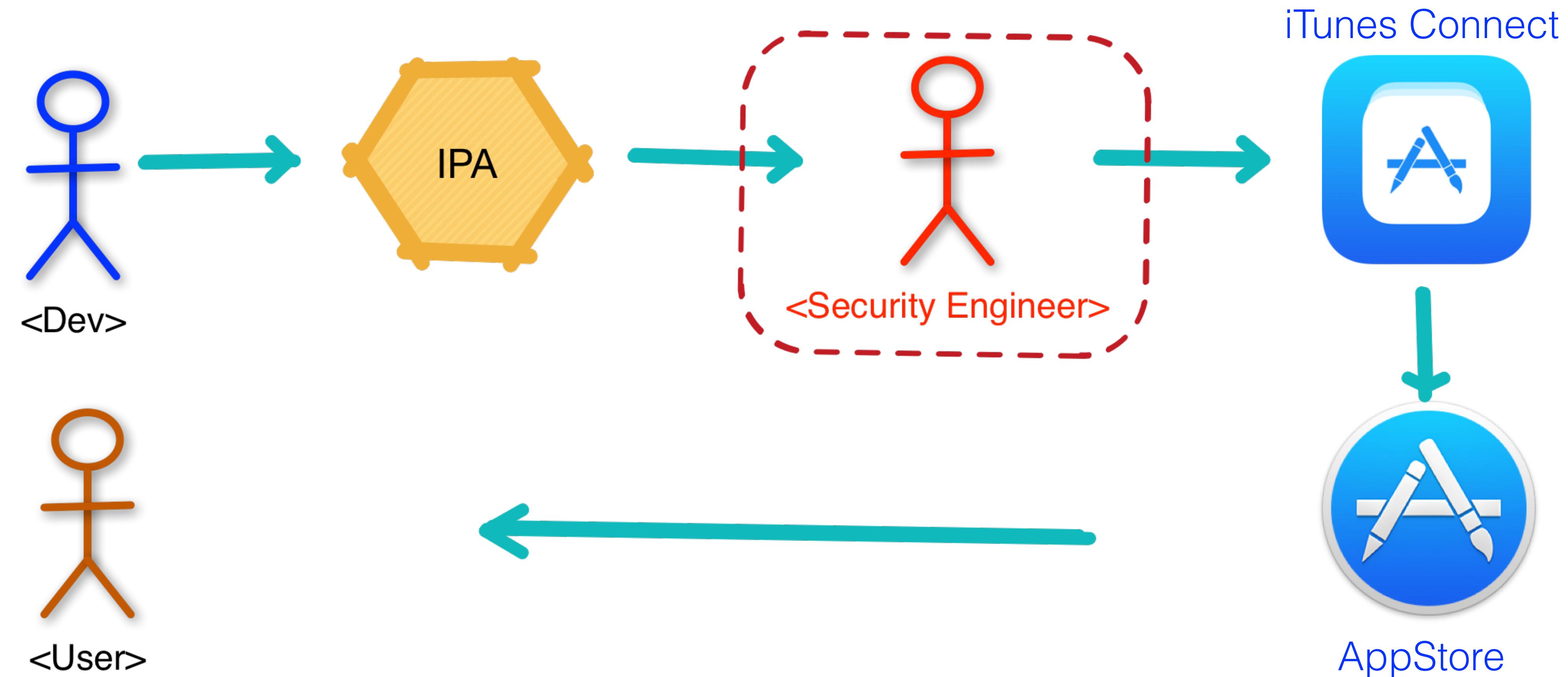
Status of iOS App Security Development Lifecycle



Status of iOS App Security Development Lifecycle

- For companies with iOS security engineers
- Developers submit the App to security engineers first
- Security engineers assess the App using the blackbox way
- After security assessment, the App is submitted to iTunes Connect
- There is a security node in the whole process, which is after the development node

Status of iOS App Security Development Lifecycle



Why Using AFL to Fuzz App during Development

- Security bugs are also a kind of bug
- Bugs should be found as earlier as possible
- Because it will take lower cost to fix
- During development, when code changed, we will run unit test to guarantee the change is correct
- When code suitable for fuzzing test changed, we should also run fuzzing test

Why Using AFL to Fuzz App during Development

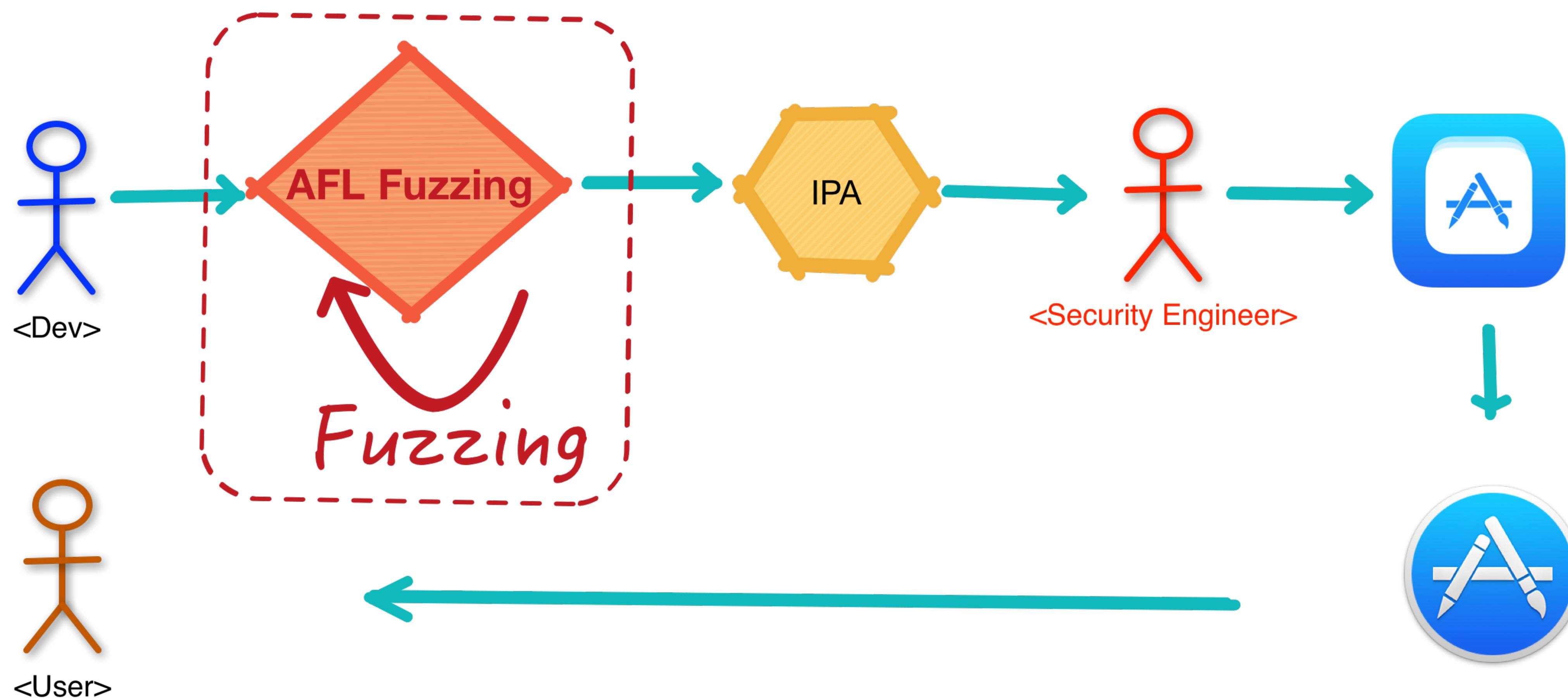
- We have the source code of our App, this is import for using AFL
- AFL is easy to config and easy to use:
 - no need to know how to do mutation, how to cover more code path
 - just compile the App with AFL
 - give AFL the normal input
 - all the else will be done by AFL

Why Using AFL to Fuzz App during Development

- AFL is pretty suitable for developers who have less knowledge of security to improve the security of their App
- AFL can be integrated with CI(Continuous Integration)
- So we can config to run daily AFL fuzzing
- Improve the security of target App daily by daily

Why Using AFL to Fuzz App during Development

- SDL with AFL



Build AFL Toolchain - Port Codes

- First pull the code of AFL from Github, and create a new branch for iOS
- AFL calls `shmget()` to create shared memory
- On iOS, App will be killed if it calls this API
- So we need to change the way to create shared memory
- Change the API from `shmget()` to `shm_open()`
- All other changes are for this
- Get the code from my repo: <https://github.com/Proteas/afl/tree/ios-afl-clang-fast>

Build AFL Toolchain - Build Clang

- Before building AFL, we should first build clang
- Get code from: <http://opensource.apple.com/>
- Using Apple's clang is for compatibility when building Xcode projects
- run `./configure -help` to get all the params for building clang
- After building clang, add the result bin dir to PATH
- `export PATH="${CLANG_DIST_DIR}/bin:${PATH}"`

Build AFL Toolchain - For macOS

- Checkout target tag, for example: `git checkout "tags/2.13b"`
- Go to the dir of AFL, run `make`
- Go to `llvm_mode`, run `make`
- Go to the dir of AFL again, run `make install`
- Tips: If meet link error, copy clang binary from `$(BUILD_ROOT)/Release+Asserts/bin` to `$(CLANG_DIST_DIR)/bin`

Build AFL Toolchain - For iOS

- Checkout the iOS branch, for example: git checkout "iOS-2.13b"
- Config the cross-compile Env with the following shell script:

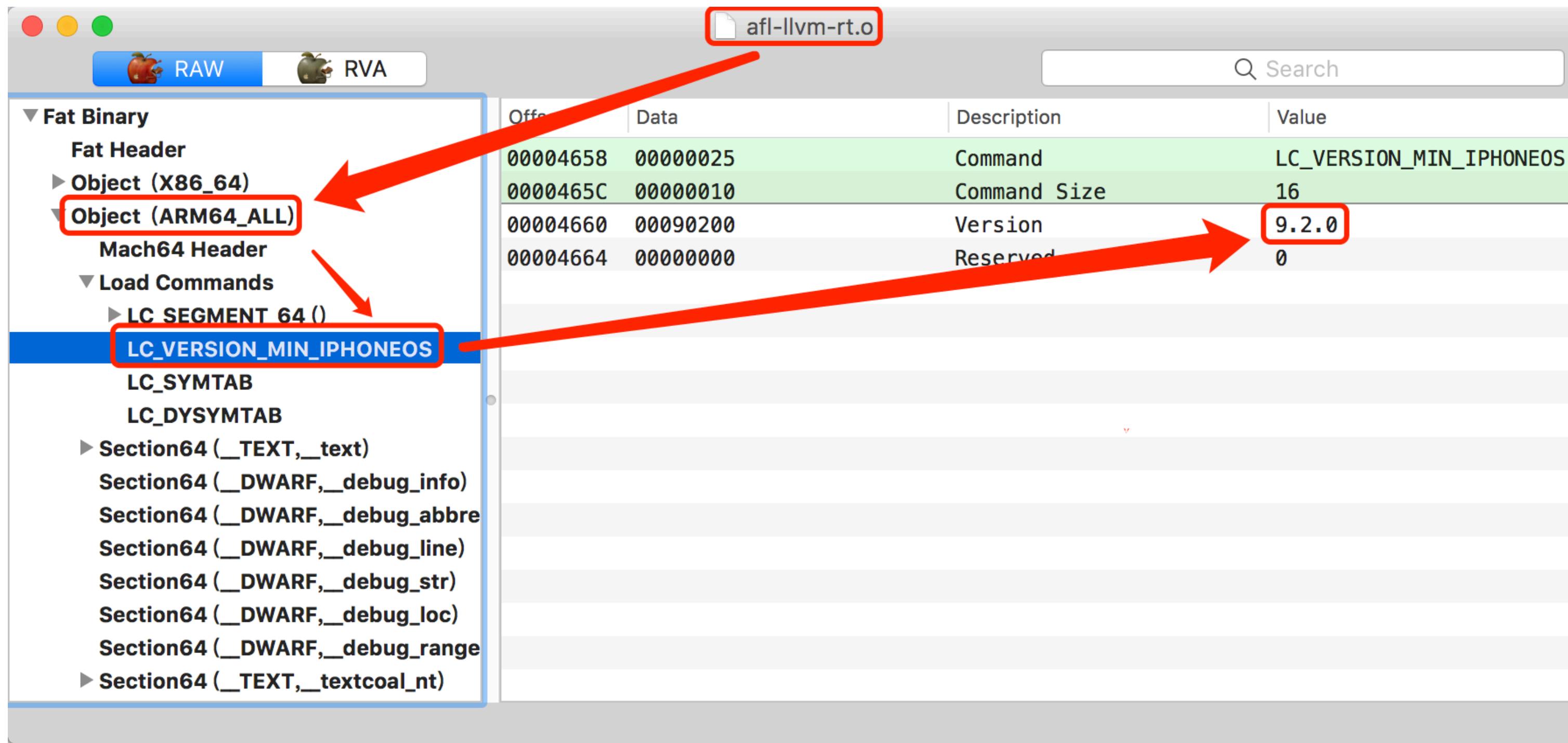
```
1 CC_IOS=`xcodebuild -sdk iphoneos -find clang`  
2 SDK_ROOT_IOS=`xcodebuild -version -sdk iphoneos Path`  
3  
4 export CC="${CC_IOS}"  
5 export CXX="${CC_IOS}"  
6 export CFLAGS="-isysroot ${SDK_ROOT_IOS} -arch arm64"  
7 export CXXFLAGS="${CFLAGS}"  
8 export AFL_NO_X86=1  
9
```

Build AFL Toolchain - For iOS

- Cross-compile the following targets:
 - `afl-fuzz`, `afl-showmap`, `afl-tmin`, `afl-gotcpu`, `afl-analyze`
 - `./llvm_mode/afl-llvm-rt.o`
- Native compile: `afl-clang-fast`
- Use `lipo` to merge the build results, so we can fuzz macOS and iOS App using the same toolchain

Build AFL Toolchain - For iOS

- Tips: change LC_VERSION_MIN_IPHONEOS of afl-llvm-rt.o to 9.0



- Otherwise, you may meet link error
- Now you can do AFL fuzzing on iDevice

Build AFL Toolchain - For iOS



Build AFL Toolchain - Tips and Tricks

- Currently AFL-iOS can only fuzz arm64 binary
- Because AFL using C++11's thread local storage, the App deployment target should be ≥ 9.0
- Because of Jetsam, we should limit the memory usage of target App
 - `./afl-fuzz -i ${TEST_CASES} -o ${RESULT_DIR} -m 80M ${TARGET_APP} @@`
- Otherwise, if the fuzzing target consumes too much memory, the fuzzer itself will be killed

Characteristics and Remote Attacking Surfaces of iOS App

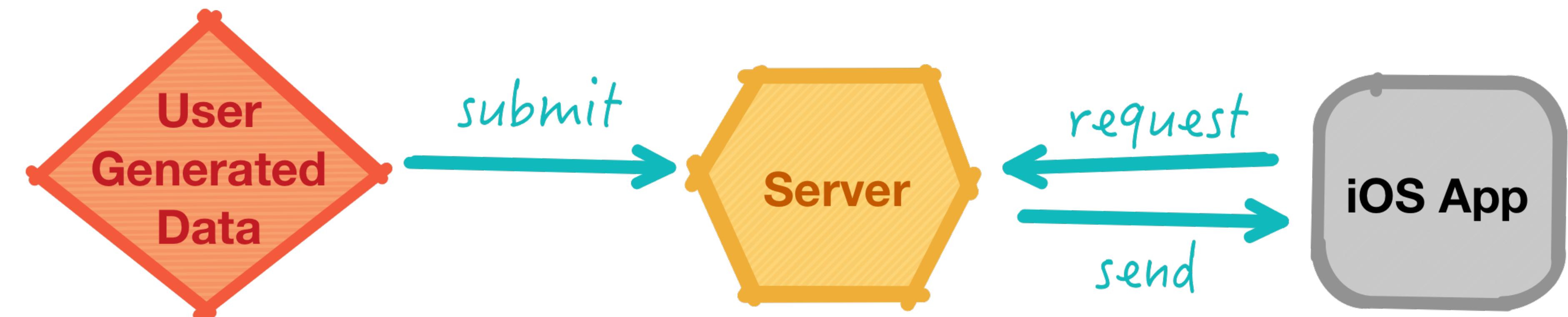
- Most of the Apps only communicate with their own server
- Requires HTTPS connections for iOS Apps by the end of this year(2016)
- The remote attacking surface is narrow relatively after using HTTPS
- If there are certificate validation vulnerabilities or config mistakes in iOS App
- Traditional remote attacking surfaces will be back: most are MitM related

Characteristics and Remote Attacking Surfaces of iOS App

- Most of the communication protocol of iOS App based on:
 - JSON
 - XML
 - Protocol Buffers
- If can be hijacked, the type-confusion is a kind of issue
- We should validate the input data immediately after receiving it:
 - JSON Schema
 - XML Schema
- Not allow any malformed data to come into our App

Characteristics and Remote Attacking Surfaces of iOS App

- If there are no certificate validation issues
- We should pay more attention to this kind of Apps:



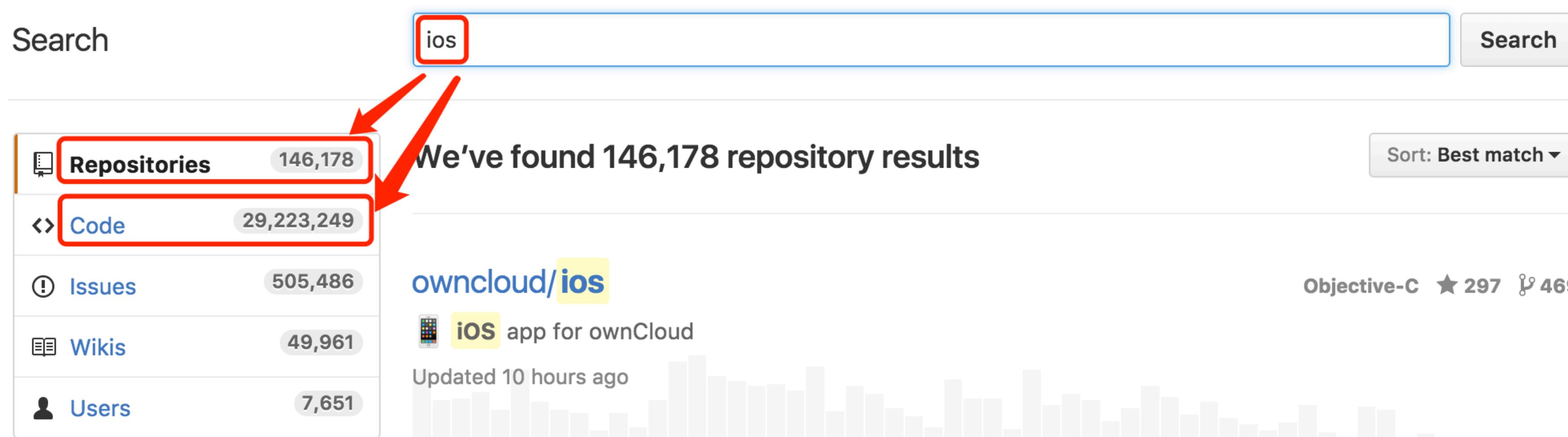
- User generated data comes into the server first,
- Then the data goes into iOS App
- And iOS App will process the data

Characteristics and Remote Attacking Surfaces of iOS App

- Apps like:
 - Instant Messaging Clients: *iMessage, WeChat*
 - Social Networking Clients: *Twitter, Facebook, Weibo*
 - Network Storage Clients: *MEGA, Dropbox*
- Different Apps have different attack surfaces depends on how it processing the user generated data

Characteristics and Remote Attacking Surfaces of iOS App

- There are lots of iOS libraries on Github
- Writing iOS App is more and more like “stacking wood”
- So make 3rd iOS libraries more security becomes important
- Search “ios” on Github(1476435790):



Characteristics and Remote Attacking Surfaces of iOS App

- Sharing is great
- There are so many codes on Github
- Some are shared by companies with fully testing or security assessment
- Some are written by individual developers
- Some are just demos
- We should do something to make the code more security
- Using AFL is a practical choice

Characteristics and Remote Attacking Surfaces of iOS App

- What libraries are more suitable for fuzzing with AFL ?
 - Parsers: JSON Parser, XML Parser, DSLs Parser
 - Video & Audio Encoder and Decoder
 - Image Encoder and Decoder
 - Archive related libraries
 - ...

Fuzz iOS App & 3rd Party Libraries

Shrek_wzw

Fuzz iOS App

- Introduce practical steps about how to fuzz our own codes
- We will use an open source app to demonstrate all the process
- The key point here is: the target function to be fuzzed is coupled seriously
- So the target function can't be fuzzed on macOS
- We need to do fuzzing on iDevice

Fuzz iOS App

- The demo App: <https://github.com/songfei/ArchiveALL>
- Function of ArchiveALL is unarchiving rar, lzma, zip on iOS
- Function code is seriously coupled with the demo app
- It is not easy to extract the specific function(for example: unrar)

Fuzz iOS App

- clone the repository, and create a new branch: AFL-Fuzz
- check out the newly created branch
- copy main.m to main-normal.m
- create file: main-afl.m
- add following contents to main-afl.m:

Fuzz iOS App

main-afl.m

```
#import "SFArchiveFileItem.h"
#import "SF7zArchive.h"
#import "SFRarArchive.h"
#import "SFZipArchive.h"

int DoFuzzing(int argc, char * argv[]);
int FuzzArchive(SFBaseArchive *archive);
int FuzzUnzip(NSString *fileName);
int FuzzUnrar(NSString *fileName);
int FuzzUn7z(NSString *fileName);

int main(int argc, char * argv[])
{
    @autoreleasepool {
        return DoFuzzing(argc, argv);
    }
}
```

```
int DoFuzzing(int argc, char * argv[])
{
    ...
    // Fuzz Type
    int type = 0;

    NSString *inputType = [NSString
stringWithUTF8String:argv[1]];
    type = (int)[inputType integerValue];

    if (type == 0) {
        return FuzzUnzip(inputFileName);
    }
    else if (type == 1) {
        return FuzzUnrar(inputFileName);
    }
    else if (type == 2) {
        return FuzzUn7z(inputFileName);
    }
    else {
        NSLog(@"error fuzz type");
        return -1;
    }
}
```

Fuzz iOS App

- Edit main.m:

```
#ifdef AFL_FUZZ
    #include "./main-afl.m"
#else
    #include "./main-normal.m"
#endif
```

- Key point of above code is using macro to control the entry of the App

Fuzz iOS App

- Create `afl-ios.xcconfig` to config build params for AFL building

```
ONLY_ACTIVE_ARCH = NO
ARCHS = arm64
VALID_ARCHS = arm64
ENABLE_BITCODE = NO
OTHER_CFLAGS = "-DAFL_FUZZ=1"
OTHER_CPLUSPLUSFLAGS = "-DAFL_FUZZ=1"
OTHER_LDFLAGS = $(PATH_TO_AFL_DIST)/afl/afl-llvm-rt.o
```

Fuzz iOS App

- Build with new source code and new xcconfig

```
AFL_ROOT_DIR="TODO"

export AFL_PATH="${AFL_ROOT_DIR}"
export PATH="${AFL_ROOT_DIR}:$PATH"

rm -rf "./Build"

xcodebuild \
    CC="${AFL_ROOT_DIR}/afl-clang-fast" \
    CXX="${AFL_ROOT_DIR}/afl-clang-fast++" \
    -project "ArchiveALL.xcodeproj" \
    -target "ArchiveALL" \
    -xcconfig "./afl-ios.xcconfig" \
    -configuration "Debug"
```

Fuzz iOS App

- Run it on iDevice
- Fuzzing Unrar

```
american fuzzy lop 2.13b (ArchiveALL)
process timing
run time : 0 days, 0 hrs, 0 min, 39 sec
last new path : 0 days, 0 hrs, 0 min, 0 sec
last uniq crash : none seen yet
last uniq hang : 0 days, 0 hrs, 0 min, 5 sec
cycle progress
now processing : 0 (0.00%)
paths timed out : 0 (0.00%)
stage progress
now trying : calibration
stage execs : 7/10 (70.00%)
total execs : 714
exec speed: 27.08/sec (slow!)
fuzzing strategy yields
bit flips : 0/0, 0/0, 0/0
byte flips : 0/0, 0/0, 0/0
arithmetics : 0/0, 0/0, 0/0
known ints : 0/0, 0/0, 0/0
dictionary : 0/0, 0/0, 0/0
havoc : 0/0, 0/0
trim : 0.00%/99, n/a
overall results
cycles done : 0
total paths : 41
uniq crashes : 0
uniq hangs : 1
map coverage
map density : 1276 (1.95%)
count coverage : 1.82 bits/tuple
findings in depth
favored paths : 1 (2.44%)
new edges on : 34 (82.93%)
total crashes : 0 (0 unique)
total hangs : 1 (1 unique)
path geometry
levels : 2
pending : 41
pend fav : 1
own finds : 39
imported : n/a
variable : 0
3 项, 6.23 GB 可用
[cpu: 76%]
```

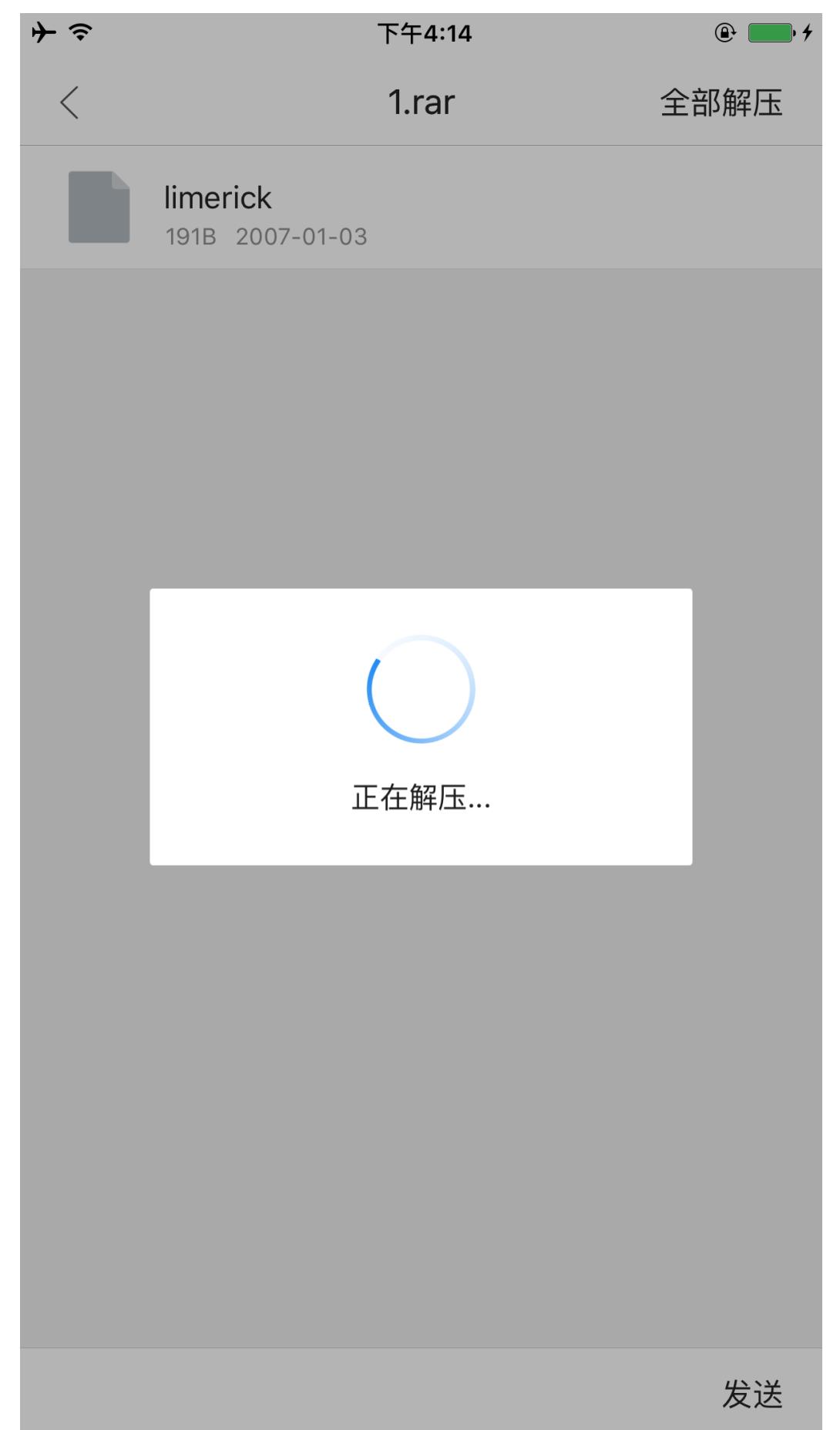
Fuzz iOS App

- As the image shows: In less than 1 minute, we got a DoS
- It can also DoS the App used this library.
- All the following fuzzers and fuzzing results can be downloaded from:
- <https://github.com/Proteas/fuzzers> based on afl

Fuzz iOS App

- QQ Browser v6.7.2.2345
- unrar DoS
- CPU Usage: 99.4%
- The GUI is freezing
- Need to kill the app

34,148	com.apple.dt.ins	root	0	2	1.14 MiB	664.89 MiB	arm64	00.56069
34,156	mttlite	mobile	99.4	11	65.02 MiB	899.41 MiB	arm64	1:35.719...
34,158	com.apple.dt.ins	root	4.1	5	2.32 MiB	666.88 MiB	arm64	04.752408



Fuzz 3rd Party Libraries

- With the doc of AFL and the previous information
- You can build your own fuzzers based on AFL
- Although we can fuzz on iOS, we prefer to do fuzzing on OS X
- The following are fuzzers and analysis of some fuzzing results

Fuzz 3rd Party Libraries

- **ZXingObjC - v3.1.0**
- An Objective-C Port of ZXing
- Out-of-Bounds Read
- 140+ hangs(infinite loop)

```
SUMMARY: AddressSanitizer: heap-buffer-overflow ZXBitMatrix.m:140 -[ZXBitMatrix getX:y:]  
Shadow bytes around the buggy address:  
0x1c06000033b0: fa  
0x1c06000033c0: fa  
0x1c06000033d0: fa fa fa fa 00 00 00 00 fa fa 00 00 00 fa fa fa  
0x1c06000033e0: 00 00 00 fa fa fa fd fd fa fa fa 00 00 00 fa  
0x1c06000033f0: fa fa fd fd fa fa fa fd fd fa fa fa 00 00  
=>0x1c0600003400: 00 fa[fa]fa 00 00 00 00 fa fa fd fd fa fa fa  
0x1c0600003410: fd fd fd fa fa fa fd fd fa fa fd fd fa  
0x1c0600003420: fa fa fd fd fa fa fa fd fd fa fa fd fd fa  
0x1c0600003430: fd fa fa fa fd fd fa fa fa fd fd fa fa fa  
0x1c0600003440: fd fd fd fa fa fd fd fa fa fa 00 00 00 00  
0x1c0600003450: fa fa fd fd fd fa fa fd fd fa fa fa fd fd
```

Fuzz 3rd Party Libraries

- **Unrar4iOS - 1.0.0 - 6c90561**
- heap overflow: -[Unrar4iOS extractStream:]
- heap overflow in C, but ObjC object may be overwritten
- Unrar4iOS.mm

```
// alloc buffer
NSLog(@"buffer size: %lu", length);
UInt8 *buffer = (UInt8 *)malloc(length * sizeof(UInt8));
.....
// copy data to buffer
NSLog(@"memcpy size: %ld", P2);
memcpy(*buffer, (UInt8 *)P1, P2);
```

```
unrar[12069:2318258] buffer size: 191
unrar[12069:2318258] memcpy size: 214
```

Fuzz 3rd Party Libraries

- **opus codec**
- Audio Codecs
- Versions
 - flac-1.3.0
 - libogg-1.3.2
 - opus-1.1
 - opus-tools-0.1.9
- Analyze the fuzzing results: stack overflows, integer overflows, ...

Fuzz 3rd Party Libraries

- opus codec - encode - wav
- Some are exploitable
- Floating point exception: 8
- AddressSanitizer failed to allocate 0xfffffffffe0004 bytes
- AddressSanitizer: **stack-overflow** on address 0x7fff5b3ceb88
- AddressSanitizer: **heap-buffer-overflow** on address 0x00014ad3c800
-

```
american fuzzy lop 2.13b (opusenc)

process timing
run time : 9 days, 4 hrs, 12 min, 54 sec
last new path : 0 days, 3 hrs, 41 min, 53 sec
last uniq crash : 0 days, 6 hrs, 48 min, 41 sec
last uniq hang : 0 days, 3 hrs, 15 min, 37 sec

cycle progress
now processing : 959* (62.31%)
paths timed out : 0 (0.00%)

stage progress
now trying : havoc
stage execs : 151k/160k (94.42%)
total execs : 7.32M
exec speed : 12.26/sec (zzzz...)

fuzzing strategy yields
bit flips : 265/177k, 72/177k, 36/177k
byte flips : 8/22.2k, 3/21.7k, 1/21.6k
arithmetics : 210/1.21M, 4/843k, 0/104k
known ints : 29/100k, 70/500k, 25/933k
dictionary : 0/0, 0/0, 13/264k
havoc : 822/2.57M, 0/0
trim : 20.33%/10.8k, 2.25%

overall results
cycles done : 0
total paths : 1539
uniq crashes : 34
uniq hangs : 45

map coverage
map density : 2893 (4.41%)
count coverage : 7.03 bits/tuple

findings in depth
favored paths : 47 (3.05%)
new edges on : 89 (5.78%)
total crashes : 43.1k (34 unique)
total hangs : 4061 (45 unique)

path geometry
levels : 4
pending : 1496
pend fav : 25
own finds : 1538
imported : n/a
variable : 228

[cpu: 73%]
```

Fuzz 3rd Party Libraries

- **opus codec - encode - aif**
- **Some are exploitable**
- AddressSanitizer: **stack-overflow** on address *0x7ffed2b175d8*
- AddressSanitizer: **heap-buffer-overflow** on address *0x62e000000000*
- AddressSanitizer failed to allocate *0xfffffffffe0004* bytes
- AddressSanitizer: SEGV on unknown address *0x62de00001dac*
- AddressSanitizer: unknown-crash on address *0xffffffff504c0d420*
-

```
american fuzzy lop 2.13b (opusenc)

process timing
run time : 9 days, 0 hrs, 33 min, 49 sec
last new path : 1 days, 16 hrs, 45 min, 25 sec
last uniq crash : 6 days, 23 hrs, 6 min, 50 sec
last uniq hang : 6 days, 8 hrs, 43 min, 3 sec

cycle progress
now processing : 593* (47.21%)
paths timed out : 0 (0.00%)

stage progress
now trying : interest 32/8
stage execs : 63.6k/123k (51.62%)
total execs : 7.89M
exec speed : 0.83/sec (zzzz...)

fuzzing strategy yields
bit flips : 279/273k, 71/273k, 40/273k
byte flips : 8/34.1k, 3/33.1k, 5/33.1k
arithmetics : 168/1.85M, 9/1.29M, 1/141k
known ints : 28/155k, 77/762k, 16/1.32M
dictionary : 0/0, 0/0, 12/265k
havoc : 567/1.09M, 0/0
trim : 47.00%/17.0k, 3.06%

overall results
cycles done : 0
total paths : 1256
uniq crashes : 29
uniq hangs : 123

map coverage
map density : 2904 (4.43%)
count coverage : 6.83 bits/tuple

findings in depth
favored paths : 58 (4.62%)
new edges on : 87 (6.93%)
total crashes : 40.9k (29 unique)
total hangs : 681 (123 unique)

path geometry
levels : 3
pending : 1233
pend fav : 44
own finds : 1255
imported : n/a
variable : 190

[cpu: 52%]
```

Fuzz 3rd Party Libraries

- **opus codec - encode - flac**
- AddressSanitizer: *SEGV* on unknown address
0x00000000000000
- Floating point exception: 8
- AddressSanitizer: *SEGV* ??:0 *oi_strncasecmp*
-

```
american fuzzy lop 2.13b (opusenc)

process timing
| run time : 9 days, 0 hrs, 31 min, 48 sec
| last new path : 1 days, 7 hrs, 56 min, 4 sec
| last uniq crash : 8 days, 16 hrs, 13 min, 5 sec
| last uniq hang : none seen yet

cycle progress
| now processing : 297 (30.56%)
| paths timed out : 0 (0.00%)

stage progress
| now trying : auto extras (over)
| stage execs : 366k/435k (84.23%)
| total execs : 20.3M
| exec speed : 3.15/sec (zzzz...)

fuzzing strategy yields
| bit flips : 148/3.67M, 76/3.67M, 58/3.67M
| byte flips : 6/459k, 3/26.9k, 2/27.3k
| arithmetics : 161/1.49M, 18/1.32M, 1/1.20M
| known ints : 20/87.6k, 68/354k, 73/610k
| dictionary : 0/0, 0/0, 60/496k
| havoc : 271/2.80M, 0/0
| trim : 1.07%/57.0k, 94.13%

overall results
| cycles done : 0
| total paths : 972
| uniq crashes : 19
| uniq hangs : 0

map coverage
| map density : 3158 (4.82%)
| count coverage : 5.23 bits/tuple

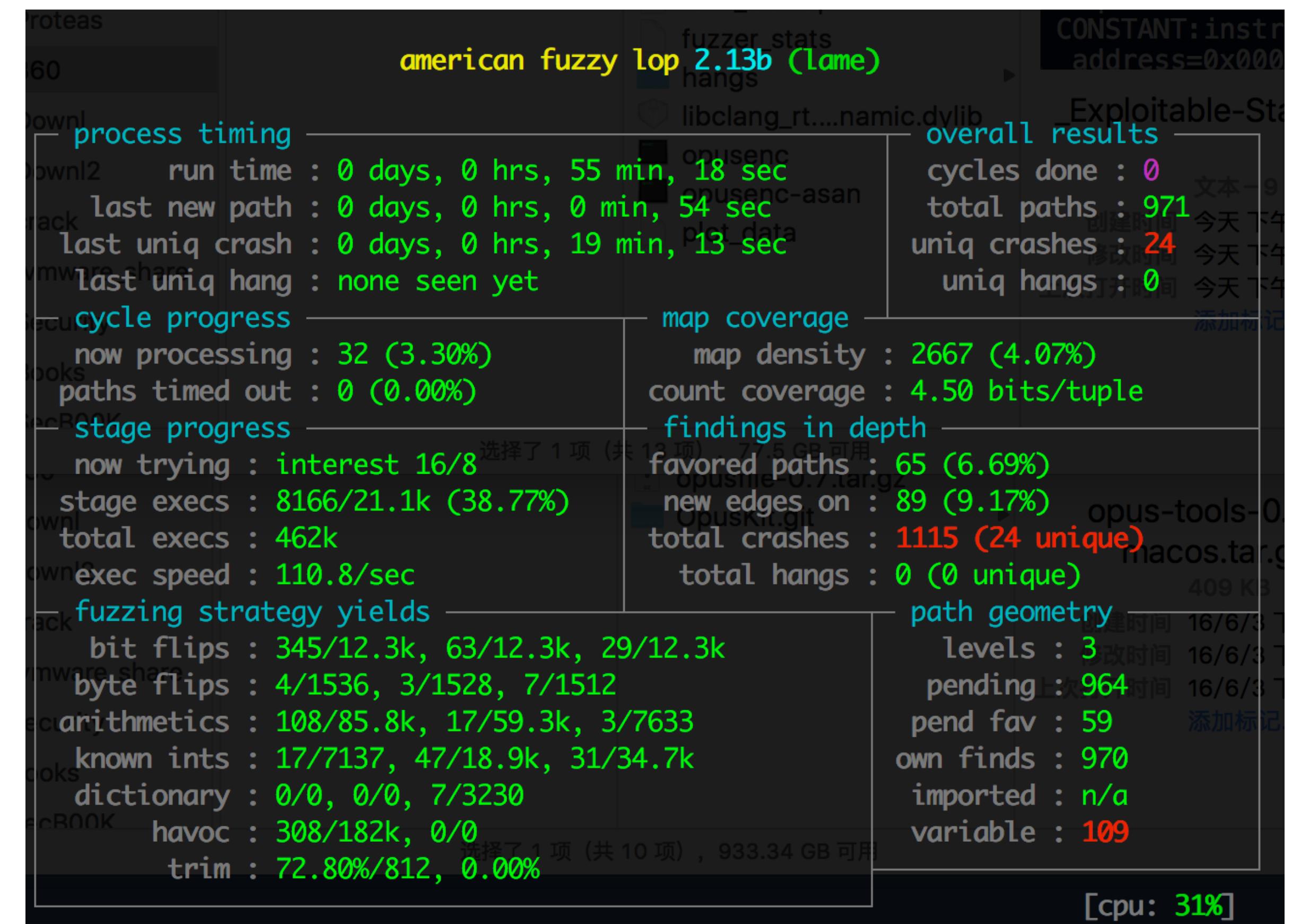
findings in depth
| favored paths : 152 (15.64%)
| new edges on : 208 (21.40%)
| total crashes : 8423 (19 unique)
| total hangs : 0 (0 unique)

path geometry
| levels : 3
| pending : 919
| pend fav : 123
| own finds : 970
| imported : n/a
| variable : 122

[cpu: 86%]
```

Fuzz 3rd Party Libraries

- **lame mp3 encoder - 3.99.5**
- AddressSanitizer: SEGV on unknown address
0x60bffff05b38
- AddressSanitizer: SEGV ???:0 fill_buffer
- AddressSanitizer: SEGV on unknown address
0x000000000000
- AddressSanitizer: **heap-buffer-overflow** on address
0x60c00000bd3c
- AddressSanitizer: **heap-buffer-overflow** ???:0 fill_buffer
-



Fuzz 3rd Party Libraries

- **KxMovie(ffmpeg decoder) - 2c5324b0**
- iOS movie player based on ffmpeg
- Fuzz results: decode flv
- You could clone the fuzzer and continue to fuzz other formats

```
american fuzzy lop 2.13b (ffmpeg-ios-decoder)

process timing
    run time : 0 days, 15 hrs, 47 min, 43 sec
    last new path : 0 days, 0 hrs, 4 min, 33 sec
    last uniq crash : 0 days, 0 hrs, 13 min, 38 sec
    last uniq hang : 0 days, 15 hrs, 40 min, 53 sec

cycle progress
    now processing : 0 (0.00%)
    paths timed out : 0 (0.00%)

stage progress
    now trying : interest 16/8
    stage execs : 12.0k/160k (7.46%)
    total execs : 546k
    exec speed : 9.68/sec (zzzz...)
    fuzzing strategy yields
        bit flips : 577/35.3k, 110/35.3k, 68/35.3k
        byte flips : 4/4412, 8/4411, 21/4409
        arithmetics : 129/246k, 32/126k, 14/11.8k
        known ints : 45/19.9k, 0/0, 0/0
        dictionary : 0/0, 0/0, 0/0
        havoc : 0/0, 0/0
        trim : 0.00%/1090, 0.00%

overall results
    cycles done : 0
    total paths : 1003
    uniq crashes : 35
    uniq hangs : 1

map coverage
    map density : 5259 (8.02%)
    count coverage : 3.30 bits/tuple

findings in depth
    favored paths : 1 (0.10%)
    new edges on : 160 (15.95%)
    total crashes : 558 (35 unique)
    total hangs : 1 (1 unique)

path geometry
    levels : 2
    pending : 1003
    pend fav : 1
    own finds : 1002
    imported : n/a
    variable : 3

[cpu: 98%]
```

Thanks

- Thanks To Michal Zalewski <lcamtuf@google.com>
- For developing and sharing AFL

Reference

- Number of apps available in leading app stores as of June 2016
- American Fuzzy Lop: <http://lcamtuf.coredump.cx/afl/>
- ArchiveALL: <https://github.com/songfei/ArchiveALL>
- ZXingObjC: <https://github.com/TheLevelUp/ZXingObjC>
- Unrar4iOS: <https://github.com/ararog/Unrar4iOS>
- opus codec: <https://www.opus-codec.org/>
- KxMovie: <https://github.com/kolyvan/kxmovie>

Question ?