Make iOS App more Robust and Security through Fuzzing

Wei Wang & Zhaowei Wang
2016-10-14
About us

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

- Status of iOS App Security Development Lifecycle
- Why Using AFL to Fuzz App during Development
- Port AFL to iOS
- Characteristics and 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
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
Status of iOS App Security Development Lifecycle
Why Using AFL to Fuzz App during Development

• Bugs should be found as earlier as possible
• We have the source code of our App, this is important for using AFL
• AFL is easy to configure and easy to use
• Can be integrated with CI (Continuous Integration)
• When run unit tests with CI, should also run AFL fuzzing
Why Using AFL to Fuzz App during Development

- SDL with AFL
Port AFL to iOS - Port Codes

- Change the API used to create shared memory: `shmget()` —> `shm_open()`
- All other changes are for this
- Get the code from my repo: [https://github.com/Proteas/afl/tree/ios-afl-clang-fast](https://github.com/Proteas/afl/tree/ios-afl-clang-fast)
- This method is also compatible with AFL 2.35b (currently latest version)
Port AFL to iOS - Build Clang

- Before building AFL, should first build clang

- Get code from: http://opensource.apple.com/

- Using Apple’s clang is for compatibility when building Xcode projects

- After building clang, add the result bin dir to PATH

  - export PATH="$\{CLANG_DIST_DIR\}/bin:$\{PATH\}"


Port AFL to iOS - Build AFL

- Set Env param: `export AFL_NO_X86=1`

- Cross-compile 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, then can fuzz macOS and iOS App using the same toolchain
Port AFL to iOS - 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, should limit the memory usage
- 
  ```
  ./afl-fuzz -i ${TEST_CASES} -o ${RESULT_DIR} -m 80M ${TARGET_APP}
  ```
Port AFL to iOS
Characteristics and 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
- 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
Characteristics and 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 come into our App
Characteristics and Attacking Surfaces of iOS App

- If there are no certificate validation issues

- We should pay more attention to this kind of Apps:

  - Apps like: iMessage, Twitter, Facebook, Dropbox, etc

- Different Apps have different attack surfaces depends on how it processing the user generated data
Characteristics and Attacking Surfaces of iOS App

• There are lots of iOS libraries on Github

• Writing iOS App is more and more like “stacking wood”

• Search “ios” on Github(1476435790):
Characteristics and 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 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

- 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](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

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

int DoFuzzing(int argc, char * argv[])
{
    if (argc != 3) {
        NSLog(@"Usage: ./ArchiveAll 0|1|2 ./test.zip");
        return -1;
    }
    NSString *inputFileName = [NSString stringWithUTF8String:argv[2]];
    if (![fileManager fileExistsAtPath:inputFileName]) {
        NSLog(@"%s: file not exist", __FUNCTION__);
        return -1;
    }
    // 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;
    }
}

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

Fuzz iOS App

• Edit `main.m`:

```c
#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

```bash
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

```bash
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
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.

• QQ Browser v6.7.2.2345

• 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
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 will show some fuzzers and analysis some of the fuzzing results
Fuzz 3rd Party Libraries

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

```c
// 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);
```
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
- Analysis the fuzzing results, you will find: stack overflows, integer overflows, …
Fuzz 3rd Party Libraries

- opus codec - encode - wav
- Some are exploitable
- Floating point exception: 8
- AddressSanitizer failed to allocate 0xfffffffff0004 bytes
- AddressSanitizer: stack-overflow on address 0x7fff5b3ceb88
- AddressSanitizer: heap-buffer-overflow on address 0x00014ad3c800
- …..
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 0xffffffffffe0004 bytes
  - AddressSanitizer: **SEGV** on unknown address 0x62de00001dac
  - AddressSanitizer: **unknown-crash** on address 0xffffffff504c0d42
  - ......
Fuzz 3rd Party Libraries

- **opus codec - encode - flac**
- AddressSanitizer: SEGV on unknown address
  0x0000000000000000
- Floating point exception: 8
- AddressSanitizer: SEGV ???:0 oi_strncasecmp
- ……
Fuzz 3rd Party Libraries

- **lame mp3 encoder - 3.99.5**
- **AddressSanitizer: SEGV on unknown address 0x60bfff05b38**
- **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
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 ?