Mobile Hacking VI

m0bdev

iOS Jailbreak

@crakun

Sat, 10-19-13 Day 5

Made Possible &
Many, Many Thankz 2:

p0sixninja

#openjailbreak-ops

Mobile Hacking VI
Class Agenda

- Part I:
  - Communications
  - Intro to .mov files
  - Submittal Process
  - Status Update

- Part II
  - Tool Development
  - Homework

Communications

crakun contact Info

Email: crakun@m0bdev.com

Twitter: @crakun

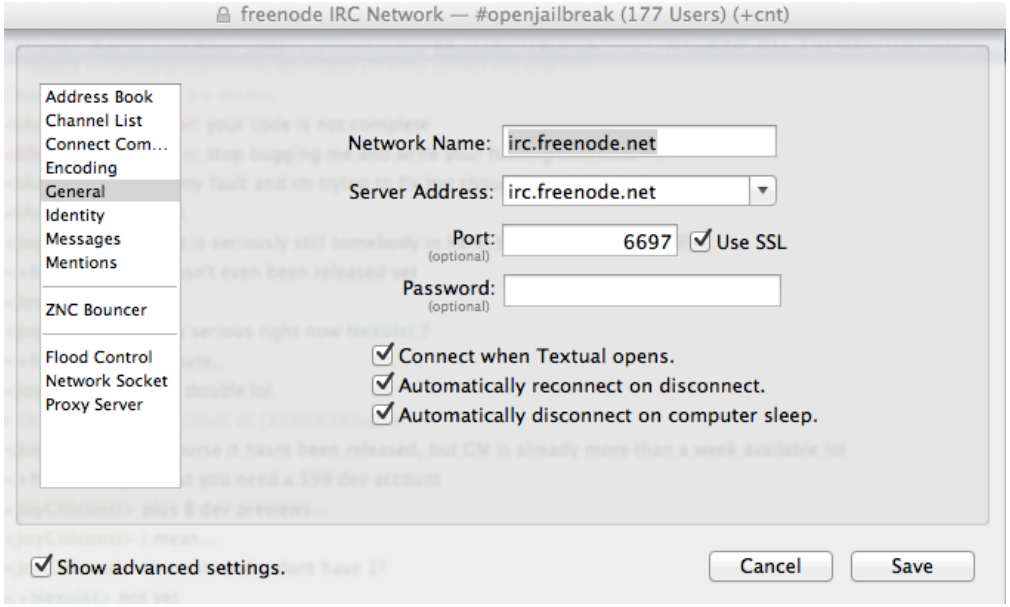
Skype: m0bdev (m0bdev is spelled with a zero)

IRC: #openjailbreak on freenode

**-> If I am slow responding / or no response, ping me again because I may have missed it

Class Communications

Text -> IRC: #openjailbreak on freenode



Skype: m0bdev (m0bdev is spelled with a zero)

Please mute Skype unless you have a question during clazz

Skype Communications is ‘limited’

Meaning, no room for :

sp00kz who don’t contribute

Flies on the wall

Apple Spiez

Intro to .mov files

Why .mov Files?

- .mov files are old, feature rich, remote, and a quick burn
- it will be patched before we complete our jb simply because we are openly working on the exploit

Long history of Quicktime bugs

For example: 2012

http://secunia.com/community/advisories/47447

- Insufficient validation when parsing encoded movie files
- Boundary errors
- Stack-based buffer overflow
- Heap-based buffer overflow
- Buffer underflow
- Off-by-one error causing a single byte buffer overflow.
- Integer overflow MPEG files.
- Use-after-free error

Complexity with Media Players -Codec

"Codec" is a technical name for "compression/decompression". It also stands for "compressor/decompressor" and "code/decode". All of these variations mean the same thing: a codec is a computer program that both shrinks large movie files, and makes them playable on your computer . Codec programs are required for your media player to play your downloaded music and movies.

"Why do we need codecs?"

ANS: Because video and music files are large, they become difficult to transfer across the Internet quickly. To help speed up downloads, mathematical "codecs" were built to encode ("shrink") a signal for transmission and then decode it for viewing or editing. Without codecs, downloads would take three to five times longer than they do now.

"Is there only one codec I need?"

ANS: Sadly, there are hundreds of codecs being used on the Internet, and you will need combinations that specifically play your files. There are codecs for audio and video compression, for streaming media over the Internet, videoconferencing, playing mp3's, speech, or screen capture. To make matters more confusing, some people who share their files on the Net choose to use very obscure codecs to shrink their files. This makes it very frustrating for users who download these files, but do not know which codecs to get to play these files. If you are a regular downloader, you will probably need ten to twelve codecs to play your music and movies.

What codec is used for .MOV files

- *As far as I know they can only be en/decoded by Quicktime. (**mediaserverd for iPhone**)*

- MOV files have used dozens of different video and audio formats over the years.

- Currently, the most common is H.264 video and AAC audio, and they can in fact be played by nearly everything, mainly because the MOV format is basically the same as MP4. You might have to remux the file though, as there are some subtle differences between MOV and MP4. Yes, this is yet another reason to hate Apple.

Class review of .mov Crashes

Ideally, we are looking for a .mov file that is exploitable for

- Jailbroken iPhone4/5 (6.1.2)
- UnJailbroken iPhone4 (7.0.2)
- UnJailbroken 5/5S (7.0.2)

Starting on Oct 26, Based on Submittals we will go over a few .mov files to determine if they are exploitable as a class

1st Steps in Bug Hunting

- Find / Create Fuzzer (http://caca.zoy.org/wiki/zzuf, peach)
- Find / Make sample files
  - FFmpeg, Camtasia, powerpoint, Final Cut Pro, Quicktime Pro
- Generate semi-invalidate samples
- Fuzz Application
  - Log such that you know which file caused fault
- Review Faults
- AptDiff (Good file, Mutated file)
- Change each difference back, to identify the byte that caused the fault
- Verify/ Debug to determine exploitability

Crash Analysis

- If we can determine that we have some level of control over the instruction pointer (Program Counter, PC in ARM) or structured exception handler, we can quickly determine that the crash is in fact exploitable.

- However, when this is not the case, analysis of crash data becomes significantly more difficult.

- Determining if a crash allows us to corrupt portions of memory or manipulate structures require a great amount of time and understanding of the application.

Reverse Engineering Quicktime Bugs

(thank u Bruno for the Info Research)

- A Bug Hunter's Dairy (Chapter 8)
- http://reversemode.com/index.php?option=com_content&task=view&id=69&Itemid=1
- http://shakacon.org/2009/talks/Exploit_or_Exception__DeMott.pdf
- https://www.corelan.be/index.php/2013/07/02/root-cause-analysis-integer-overflows/
- https://www.corelan.be/index.php/2013/02/26/root-cause-analysis-memory-corruption-vulnerabilities/
- http://opensecuritytraining.info/IntroductionToReverseEngineering.html

Crash Report References

- http://www.plausible.coop/blog/?p=176

- https://gist.github.com/dennda/56851a63d935b54d3147

Tools to view Quicktime Format

Mp4Player (Windows)
Dumpster (Mac)
Atom Explorer (Mac)

Export sections of the file?

- Your best bet is to extend MP4 Explorer to do this, or write your own parser. Parsing the atoms is actually pretty simple, things start to get complicated when you need to interpret the content of the atoms and cross-reference them to, for example, locate where the frame data is.

- The QuickTime file format specification is the best resource for Apple generated QuickTime files, but you may need to do some reverse engineering, as the spec is not very complete in some areas, like the handling of MPEG-2 and MPEG-4 video.

- If you have access to ISO specs, the have access to ISO specs, the ISO/IEC 14496-12 is a standardized version of the QuickTime format (or better said, of a subset of it). The ISO/IEC 14496-15 specification builds on top of 14496-12 and defines a specific implementation of this format for the H. 264 format. This is the so called MP4 format.

Submittal Process

Submittal for .mov Crashes

Email Me (crakun@m0bdev.com) with a zip file of:

1. Which phones and firmware were the .mov file tested on

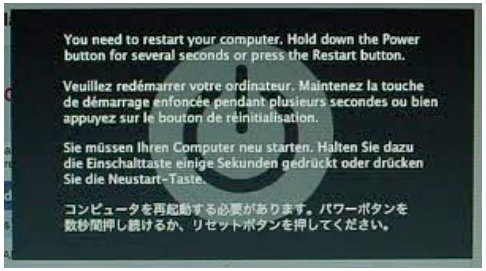
2. Include Crash Report

3. Include original, unfuzzed .mov file and fuzzed .mov file

4. Tell me what you think it is?

Bugs, Vulns, Exploits we are looking for in class

- Remote Exploits
- Userland Vulnerabilities(to obtain mobile)
- Privilege escalation (mobile to root)
- Escaping sandbox techniques
- Memory leaks
- Bypassing code signing techniques
- Kernel Vulnerabilities
- Strategies for dealing with KASLR on 64-bit ARM



Submittal for other Bugs/ Vulns / Exploits

Email Me (crakun@m0bdev.com) with a zip file of:

1. Detail Step-by-Step Method how / what you did so I can reproduce it
2. What devices and firmware does this occur on?
3. Tell me what you think it is?
4. Include Crash Report
5. Include Proof-of-Concept source code

Today’s Status Updates

Status Updates

Submittals so Far:

SIGSEGV
Kernel Panic
Priviledge Escalation
Push Button Hacking

mumble.m0bdev.com being built: thank q <timhstar-yt>
Wiki for m0bdev being built -> thank q <saelo>

Tool Development

m0bdev Team

Jailbreak Codename:

chemrail

An Analogy,…

http://www.youtube.com/watch?v=oIBtePb-dGY

Tool Talk Showcase

- Tool development for Jailbreak class:

Cykey http://nexuist.tumblr.com/

Nexuist https://github.com/Cykey/mobilesafari-fuzzer

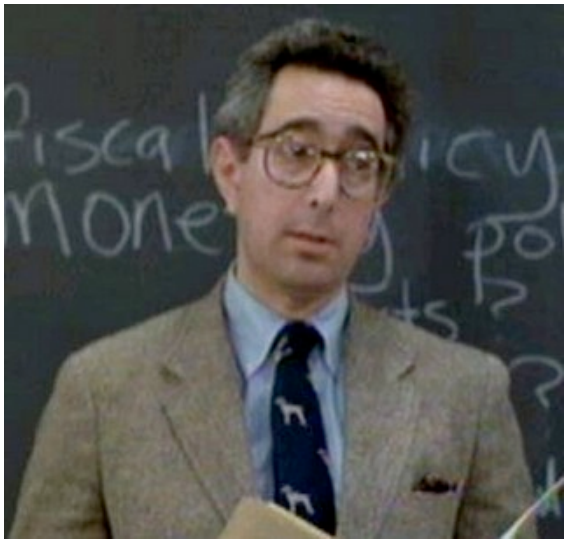
DarkMalloc http://github.com/DarkMalloc/pinyserv

isa56k **http://**www.hotwan.com/class/fuzzers/fuzzyDuck.sh

Ask these people and others (**iAdam1n, compilingEntropy**) on IRC #openjailbreak to get your fuzzing up and running

Homework

Review of 'YOUR' Oct 12 Homework Assignment



- Prepare a list of the best IRC channels and websites for iOS Jailbreak research (we will distribute this to class for everyone)

Homework Assignment

- Review slide deck.
  - Think about what you can do. Let me know.
  - Contact me if you want to share bugs, confirmed vulns, exploits via Email: crakun@m0bdev.com

- r0ck'n:
  - Fuzz .mov files / Submit

- Spread the Word. We need Volunteers for this Jailbreak

Stay Tuned

More

Surprises

are yet to come

Finding iOS Kernel Bugs

- http://reversemode.com/index.php?option=com_content&task=view&id=69&Itemid=1

Next Meeting:
(Day 6): Sat, Oct 26, 2013 6am PST

- Distribution of select bugs to class for analysis
- Intro to Crash Report
- Intro to Reverse Engineering

- Weekly Class: Saturdays, 6am PST

Open Floor Discussion

APPENDIX

ARM Reference Material

ARM Quick reference card:

- http://web.eecs.umich.edu/~prabal/teaching/eecs373-f10/readings/ARM_QRC0001_UAL.pdf

Reverse Engineering ARM:

- http://www.peter-cockerell.net/aalp/html/frames.html
- http://media.hacking-lab.com/scs3/scs3_pdf/SCS3_2011_Bachmann.pdf
- http://www.raywenderlich.com/37181/ios-assembly-tutorial
- http://www.phrack.com/issues.html?issue=66&id=12

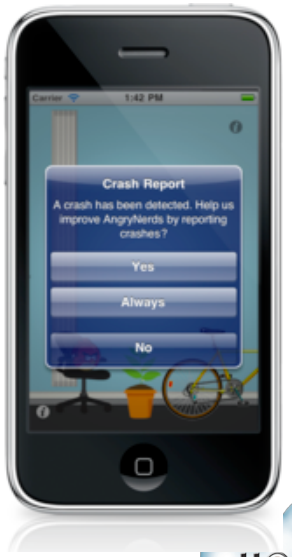
To Begin:

- http://lightbulbone.com/post/27887705317/reversing-ios-applications-part-1
- http://yurichev.com/writings/RE_for_beginners-en.pdf
- http://blog.claudxiao.net/wp-content/uploads/2011/07/Elementary-ARM-for-Reversing.pdf

Fuzzing References

Fuzzing .mov Files

- .mov files are old, feature rich, remote, and a quick burn
- Any crashes so far? (don't send them to Apple)
- Do we need to build a custom .mov file fuzzer?
  - http://shakacon.org/2009/talks/Exploit_or_Exception__DeMott.pdf
  - http://www.blackhat.com/presentations/bh-usa-05/bh-us-05-sutton.pdf
  - http://www.cert.org/vuls/discovery/bff.html
  - http://peachfuzzer.com/v3/TutorialFileFuzzing.html
  - Chapter 6: iOS Hacker's Handbook
  - https://developer.apple.com/standards/classicquicktime.html



More on Fuzzing

- Exploring Code Paths
- Sending invalid data to an interface in hopes of triggering a error condition or fault condition.
- These errors can lead to exploitable vulnerabilities

Signals

Signals

https://developer.apple.com/library/ios/documentation/system/conceptual/manpages_iphoneos/man3/signal.3.html

Signals allow the manipulation of a process from outside its domain, as well as allowing the process to manipulate itself or copies of itself (children). There are two general types of signals: those that cause termination of a process and those that do not. Signals which cause termination of a program might result from an irrecoverable error or might be the result of a user at a terminal typing the `interrupt' character.

Most signals result in the termination of the process receiving them, if no action is taken; some signals instead cause the process receiving them to be stopped, or are simply discarded if the process has not requested otherwise. Except for the SIGKILL and SIGSTOP signals, the **signal**() function allows for a signal to be caught, to be ignored, or to generate an interrupt. These signals are defined in the file <signal.h>:

```
Vortexs-MacBook-Air:~ vortex$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGEMT       8) SIGFPE
 9) SIGKILL     10) SIGBUS      11) SIGSEGV     12) SIGSYS
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGURG
17) SIGSTOP     18) SIGTSTP     19) SIGCONT     20) SIGCHLD
21) SIGTTIN     22) SIGTTOU     23) SIGIO       24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGINFO     30) SIGUSR1     31) SIGUSR2
```

SIGNALS

- **<ep0k>** Not all crashes are interesting : aborts, timeouts or out of memory kind of crashes are useless.

- Verify the crash dump in Settings / General / About / Diagnostics & Usage / Diagnostic & Usage Data that the crash report you created is of Exception Type SIGILL, SIGBUS or SIGSEGV. This are useful types.

SIGILL

The **SIGILL** signal is raised when an attempt is made to execute an invalid, privileged, or ill-formed instruction.

SIGILL is usually caused by a program error that overlays code with data or by a call to a function that is not linked into the program load module.

<C0deH4cker> SIGILL is very unlikely to encounter but in general the best one to find, as it means its executing code where it should'nt be

SIGSEGV

<C0deH4cker> SIGSEGV is the next best one, and it likely to be exploitable

In SIGSEGV, it's trying to read/write to an invalid address.

Signal 11– SIGSEGV

– Always the hottest because it means we're already dealing with a memory read/write issue

- We want a SEGV based on a write operation

Ideas toward popular heap exploitation techniques

- Clobber function pointer on Heap with overwrite
- Get that function called

SIGABRT

```
15. Exception Type:  EXC_CRASH (SIGABRT)
16. Exception Codes: 0x0000000000000000, 0x0000000000000000
```

abort() sends the calling process the SIGABRT signal, this is how abort() basically works.

abort() is usually called by library functions which detect an internal error or some seriously broken constraint. For example malloc() will call abort() if its internal structures are damaged by a heap overflow.

Often indicate heap issues when fuzzing

- But that's why were here, so back to the QuickTime crash from my Mac
  - **SIGFPE** is the signal sent to computer programs that perform erroneous arithmetic operations on POSIX compliant platforms. The symbolic constant for SIGFPE is defined in the header file signal.h. Symbolic signal names are used because signal numbers can vary across platforms.
- So….. is it exploitable or not? Probably not.
  - Why? Because the chance for memory corruption, and thus execution redirection looks minimal. See example:

```
int main() {
  int x = 42/0;
  return 0; /* Never reached */
}
```

- I hear that. You really always need to reverse each exception to gain full understanding of the crash
  - ***GET TO THE ROOT CAUSE VIA REing***
    - If the value is used to influence allocation or similar functions, it could still have a chance of being exploitable
- On the other hand, no invalid memory access is directly happening here, so the chances that it would be exploitable seems very thin

Null pointer derefence

Null-pointer dereference issue can occur through a number of flaws, including race conditions, and simple programming omissions.

They can be exploitable sometimes. Ask Mark Dowd, Also A Bug Hunter's Diary (p 35, Ch4 and p153)

Good coding practice: Before using a pointer, ensure that it is not equal to NULL

Sk00l Supplies
Software / Hardware

- Xcode (run latest version)
- IDA Pro
- Gdb
- OxED http://www.suavetech.com/0xed/
- Mac book running Mountain Lion
- Serial Debugging Cable
- WiFi
- Jailbroken iPhone4/5 (6.1.2)
- UnJailbroken iPhone4 (7.0.2)
- UnJailbroken 5/5S (7.0.2)

Great Starter Books

- Mac OSX and iOS Internals
- A Bug Hunter's Diary
- Hacking and Securing iOS Applications
- iOS Hacker's Handbook
- Mac Hacker's Handbook
- OSX and iOS Kernel Programming
- Cocoa Application Security
- C in a nutshell
- Learn C on the Mac: For OSX and iOS
- ARM Assembly Language –An Introduction

The Goal

The Goal of this class is to build an iOS untethered Jailbreak from scratch by creating a new Community-based Jailbreak Team