Literature Review

Harry Moulton

November 2022

1 Introduction

The problem in which I am attempting to solve in my project is the lack of tools available for handling, parsing and reverse engineering firmware files on Apple platforms such as iOS, iPadOS and macOS.

Currently there is a distinct lack of available applications or tools that can handle all of these firmware files, instead it's a mix of different tools that are published either on Github or kept closed-source, and aren't routinely updated.

The solutions that do exist are, again, not regularly updated - such as JTool[1], or extremely expensive - such as IDA64[2].

2 Themes

2.1 Mach-O File Format

Handling Mach-O files is a primary requirement of this project. All iOS and macOS applications are compiled as Mach-O executables and the XNU kernel is also compiled as a Mach-O. Essentially any executable code from the Kernel up will use Mach-O. This means that handling this file format is the first functionality that would need to be implemented into any solution.

In early 2020 I began developing a C library for working with strings, lists, compression, files, Apple's Image4 format and the Mach-O executable file format. The library provides APIs for working with Mach-O files and allow for a binary to be translated into a C structure that can then be used throughout a client program. While the API does provide a Mach-O parser, much is needed to be built on-top of it for the program to meet the specified requirement - and an understanding of the file format is required too.

I briefly cover the basics of the Mach-O format in an article I wrote on my blog[1]. I cover the Mach-O header, load commands and segment commands, and how these can be handled in C. However, the article doesn't act as a user guide for libhelper, nor does it cover some other aspects of Mach-O files such as the different types, code signing and entitlements as it was only meant to be a quick-to-read introduction.

An article published by Scott Lester, however, does mention these areas in far greater detail. His blog post "A look at Apple Executable Files" covers the file format in much greater detail[2]. More importantly, Lester discusses how code signing and entitlement information is embedded into segment commands - again something Libhelper currently does not have support for. In the past only iOS binaries required code signing, but in more recent versions of macOS it's become something of a soft-requirement and therefore will be an important feature of any application that claims to be able to analyse Mach-O files. These missing features could either be implemented within libhelper directly, or built on-top as part of HTool.

Lester's article is more useful int eh context of the problem as it covers both the Mach-O format in far greater detail than anyone with exception of Apple's developer documentation, and discusses aspects such as code signing and entitlements. While my own article does introduce libhelper, which will be extensively used in HTool, I do not go into sufficient detail in the article regarding the file format.

2.2 Kernel Cache Format

The Kernel Cache file format is the second most important aspect of the project, behind Mach-O files. Where the Mach-O format has plenty of public documentation and knowledge, the same cannot be said for the Kernel cache.

There are two reasons for this. The first being Apple's commitment to secrecy regarding the internals of its operating systems - particular iOS. And the second being that the format is tweaked every few versions of iOS. The topic is also something that has never been the centre of attention, instead usually being something that is a small mention in articles about related topics.

Brandon Azad, formerly of Google's Project Zero, published in 2018 an article entitled "Analyzing the iOS 12 kernelcaches tagged pointers"[3]. The primary focus of this article was the introduction of a concept known as Pointer Authentication[], or PAC, in the ARMv8.3 architecture on the iPhone. A side note of his research into this was the different variants of the Kernel cache format.

Azad has observed the following; "There appear to be at least 3 distinct kernelcache formats".

1. iOS 11-normal: The format used on iOS 10 and iOS 11. It has split kexts, untagged pointers and around 4000 symbols.
2. iOS 12-normal: The format used on iOS 12 beta for iPhone9,1. It is similar to the iOS 11 kernelcache, but with some structural changes that confuse IDA 6.95.
3. iOS 12-merged: The format sed on iOS 12 beta for iPhone7,1. It is missing prelink segments, has merged kexts, uses tagged pointers, and, to the dismay of security researchers, is completely stripped.

This snippet from his article is extremely useful in understanding the different variants of the kernel cache. Once I begin the development of HTool I now already have a basis to work from when further researching the kernel cache format. This means my solution will be able to support as many different versions of iOS firmware files as possible.

Unfortunately, however, there is zero documentation on the new kernel cache format introduced in iOS 15. The "merged-style" that Azad mentions was used from iOS 12 to iOS 14, when it was replaced with the new "fileset-style". This is something I would have to investigate myself by reverse engineering the files. The initial development of HTool as a Mach-O parser would be the first step here, as it would allow me to visualise the kernel cache mach-o and compare with older versions to see what has changed.

2.3 Reverse Engineering

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

this is a test Atmel Corporation 2016

3 References

References

Atmel Corporation (2016). *AVR Instruction Set Manual*. URL: `https://herts.instructure.com/courses/81373/files/2097603/download` (**urlseen** 04/2021).