# Firmware Entry Points in Memory

Mahmoud Dolah and Rafael De Los Santo

November 20th, 2016

## 1 Abstract

## 2 Introduction

## 3 Related Work

Firmware entry point analysis can be considered one of the many steps in the overall study of embedded systems. As more and more objects are entering the realm of IOT and incorporate embedded systems of their own, the discussion of firmware analysis is already growing to include a wide variety of new topics.

Meier et al[10], Coppola[2], and Zaddach and Costello[14] touch upon the general idea of manually analyzing disassembled firmware images, mainly to extract information from headers and also for use in emulation. Manual analysis is, however, time consuming, considering the amount of information one needs to gather before they may begin to inspect actual code (which will be done with a tool such as IDA Pro).

Because of this, many tools have been developed to automate or at least assist in much of the analysis. Zaddach et al[13] and Costin et al[4] take the dynamic analysis approach, and run a given firmware image within an emulator. Costin et al[4] is of particular interest because its emulation process includes a partial emulation of the underlying hardware, meaning of course that only firmware for known hardware devices can be tested.

This lack of environmental knowledge (and often physical hardware) prompts many static approaches, such as Shoshitaishvili et al[11], which builds off of techniques in Kruegel et al[9] to reverse engineer firmware when only a

binary-blob is present. This brand of analysis is especially useful, considering firmware entry point detection is usually a step in the process.

Similar efforts and challenges in static analysis are seen in Costin et al[3], but on a much larger scale. Tens of thousands of images were unpacked, requiring the analysis to be general enough for a large range of binaries. The results were compared to the popular firmware analysis tool, Binwalk[7], which is often used in manual static analysis. Heffner[7] uses known patterns to locate structures within executables, and can be particularly useful for locating firmware headers.

Heffner[8] and Viehbock[12] show examples of static analysis on firmware using Heffner[7] as well as other useful techniques for reverse engineering embedded systems.

Other techniques have been developed to target specific device vulnerabilities, rather than overarching analyses of firmware images. Meier[10] and Cui et al[5] focus on the exploitation of firmware updates to deliver malicious code (to fitness trackers and HP printers, respectively), and note a general lacking in the current state of firmware security. Similarly, FIE[6] is a tool developed to find bugs in the MSP430 family of micro controllers, building off of Cadar et al[1] for the purposes of conducting symbolic execution. While these analyses are limited in scope, they all shed light into interesting ways of inspecting firmware that can potentially be applied to a greater and more general range of devices.

# 4 Background

OPTIONAL

# 5 Design

# 6 Implementation

# 7 Evaluation

# 8 Discussion

OPTIONAL Place to explain if there is something weird about the result

# 9 Limitations and Future Work

# 10 Conclusion

# References


[1] C. Cadar, D. Dunbar, and D. Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. 2008.

[2] M. Coppola. Weighing in on issues with cloud scale. 2013.

[3] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti. A large-scale analysis of the security of embedded firmwares. 2014.

[4] A. Costin, A. Zarras, and A. Francillon. Automated dynamic firmware analysis at scale: A case study on embedded web interfaces. 2015.

[5] A. Cui, M. Costello, and S. J. Stolfo. When firmware modifications attack:a case study of embedded exploitation.

[6] D. Davidson, B. Moench, S. Jha, and T. Ristenpart. Fie on firmware: Finding vulnerabilities in embedded systems using symbolic execution. 2013.

[7] C. Heffner. Binwalk.

[8] C. Heffner. Reversing the wrt120ns firmware obfuscation, 2014.

[9] C. Kruegel, W. Robertson, F. Valeur, and G. Vigna. Static disassembly of obfuscated binaries. 2002.

[10] M. Meier, D. Reinhardt, and S. WendZel. Attacks on fitness trackers revisited: A case-study of unfit firmware security. 2016.

[11] Y. Shoshitaishvili, R. Wang, C. Hauser, C. Kruegel, and G. Vigna. Firmalice - automatic detection of authentication bypass vulnerabilities in binary firmware. 2015.

[12] S. Viehbock. Reverse engineering an obfuscated firmware image e01 unpacking, 2011.

[13] J. Zaddach, L. Bruno, A. Francillon, and D. Balzarotti. Avatar: A framework to support dynamic security analysis of embedded systems firmwares. 2011.

[14] J. Zaddach and A. Costello. Embedded devices security firmware reverse engineering. 2013.