Analyzing Peripheral Interupts for Use in Embedded System Emulation

Fernando Maymi and Casey McGinley

1. Related work

Embedded devices are woefully insecure and vulnerable, and there are several studies in the current literature which corroborate this statement. For example, in 2014 a French research group used basic symbolic execution techniques to discover 38 new vulnerabilities in 693 firmware samples [8]. Additionally, in 2013 a research group at Columbia demonstrated a general purpose methodology for leveraging vulnerable firmware update mechanisms on many embedded devices [9]. As such, there is need for new and effective methods to analyze firmware images dynamically. We believe our tool represents one of the first steps towards an effective and generalized solution for emulating embedded devices, ultimately facilitating the dynamic analysis of embedded firmware images. Considering this larger goal, our work is just one small part, namely mapping interrupt numbers to the specific interrupt handlers that they trigger. In this very specific context, as far as we know, no previous work has attempted to solve the same problem. That said, there is a large body of existing research into the dynamic analysis of embedded devices and their overall security.

The classical way of dynamically analyzing an embedded device requires the firmware to be executed on the hardware itself, with a hardware debugger attached. JTAG [1] is the standard developed by IEEE for these debuggers. Analyzing firmware images in this way is limited in several capacities. First of all, hardware debuggers are difficult to work with, and especially difficult to integrate into existing dynamic analysis platforms. Furthermore, when working with hardware debuggers, there is the possibility that your test and analyses might bring the device into an improper state, causing damage to the hardware itself. Finally, analyzing directly on hardware is an unscalable solution, as for every firmware image you want to analyze, you need a corresponding piece of hardware, which can get expensive and logistically problematic.

One recently proposed alternative to hardware debuggers is Avatar [18], a dynamic analysis platform designed for firmware images. Avatar makes use of an existing general-purpose dynamic analysis platform known as S2E [6] to emulate and analyze firmware images. In order to deal with the problem of poorly specified peripherals in embedded devices, Avatar forwards all attempted interactions with the peripherals in the S2E environment to the actual hardware. This hybrid solution facilitates the use of existing dynamic analyses on firmware images, but since the hardware is still used partially, Avatar is not a scalable solution and may cause damage to the hardware.

Given that, we believe whole system emulation of embedded devices to be the only viable solution. Unlike hardware debuggers or hybrid solutions, whole system emulation doesnt suffer from problems of scale or potential hardware damage as the physical hardware is removed from the equation entirely. Additionally, in an emulated environment, everything is implemented as software, meaning that anything can be modified or edited, allowing for easy instrumentation and dynamic analysis of firmware images. This will allow us to leverage powerful existing dynamic analysis platforms like PANDA [10], which allows users to record and replay execution traces so that dynamic analysis can be performed out of sync with the execution itself, an idea first proposed by Chow et. al. at VMware [7]. PANDAs dynamic analyses are built on top of QEMU [3], an open-source emulation platform supporting many different CPU architectures.

Now that weve established where our research fits into the big picture (embedded device emulation and dynamic analysis), we consider the narrower scope of the specific work presented in this paper, namely, mapping interrupt numbers to interrupt handlers via symbolic execution. Symbolic execution [11] is a method of analyzing a program by executing on symbolic values, values which represent valid ranges or conditions. In contrast, concrete execution is business as usual, executing a program on real or concrete inputs. Symbolic execution allows a user to determine parameters for inputs that result in code execution along a given path. This neatly aligns with our needs, as we need to deter-

mine what inputs (interrupt numbers) lead to execution of which interrupt handlers. In addition, although often used interchangeably with symbolic execution, concolic execution [15] is actually a hybrid technique wherein concrete inputs identify a single execution trace which is then explored symbolically.

We will be using the angr framework [17] for our analysis because of the general and under constrained symbolic execution options. Angr provides a wide array of analytical capabilities, from static to dynamic, and from purely symbolic to concolic. SimuVEX is angrs symbolic execution engine, which tracks the program state using a python based wrapper [16] of the intermediate representation VEX [12]. The dynamic symbolic execution is based on the techniques described in Mayhem [5] which is more memory efficient than other options, allowing us to execute the firmware long enough for it to populate things like interrupt handlers which may not be present in memory until after boot. Angr supplements Mayhem with the veritesting static symbolic execution system [2] which reduces the execution space when multiple paths would have the same result. Another feature of angr that may be useful is called Under Constrained Symbolic Execution (UCSE) [13], which allows individual functions to be analyzed, though at the cost of increased false positives. This feature is implemented in angr as UC-angr, and it is based on a previous tool called UC-KLEE [14], which itself was a UCSE enhancement on top of the dynamic symbolic execution tool KLEE [4].

References

[1] Ieee standard for test access port and boundary-scan architecture. pages 1–444, 2013.

[2] Thanassis Avgerinos, Alexandre Rebert, Sang Kil Cha, and David Brumley. Enhancing symbolic execution with veritesting. In *Proceedings of the 36th International Conference on Software Engineering*, pages 1083–1094. ACM, 2014.

[3] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, 2005.

[4] Cristian Cadar, Daniel Dunbar, and Dawson Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, pages 209–224, Berkeley, CA, USA, 2008. USENIX Association.

[5] Sang Kil Cha, Thanassis Avgerinos, Alexandre Rebert, and David Brumley. Unleashing mayhem on binary code. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 380–394, Washington, DC, USA, 2012. IEEE Computer Society.

[6] Vitaly Chipounov, Volodymyr Kuznetsov, and George Candea. S2e: a platform for in-vivo multi-path analysis of software systems. *ACM SIGPLAN Notices*, 46(3):265–278, 2011.

[7] Jim Chow, Tal Garfinkel, and Peter M Chen. Decoupling dynamic program analysis from execution in virtual environments. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 1–14, 2008.

[8] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. A large-scale analysis of the security of embedded firmwares. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 95–110, 2014.

[9] Ang Cui, Michael Costello, and Salvatore J Stolfo. When firmware modifications attack: A case study of embedded exploitation. In *NDSS*, 2013.

[10] Brendan F Dolan-Gavitt, Josh Hodosh, Patrick Hulin, Tim Leek, and Ryan Whelan. Repeatable reverse engineering for the greater good with panda. 2014.

[11] James C King. Symbolic execution and program testing. *Communications of the ACM*, 19(7):385–394, 1976.

[12] Nicholas Nethercote and Julian Seward. Valgrind: a framework for heavyweight dynamic binary instrumentation. In *ACM Sigplan notices*, volume 42, pages 89–100. ACM, 2007.

[13] David A. Ramos and Dawson Engler. Under-constrained symbolic execution: Correctness checking for real code. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, pages 49–64, Berkeley, CA, USA, 2015. USENIX Association.

[14] David A Ramos and Dawson R. Engler. Practical, low-effort equivalence verification of real code. In *Proceedings of the 23rd International Conference on Computer Aided Verification*, CAV'11, pages 669–685, Berlin, Heidelberg, 2011. Springer-Verlag.

[15] Koushik Sen. Concolic testing. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 571–572. ACM, 2007.

[16] Yan Shoshitaishvili, Ruoyu Wang, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. Firmalice-automatic detection of authentication bypass vulnerabilities in binary firmware. 2015.

[17] Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Andrew Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis. In *IEEE Symposium on Security and Privacy*, 2016.

[18] Jonas Zaddach. Development of novel dynamic binary analysis techniques for the security analysis of embedded devices. TELECOM, ParisTech, 2015.