Poster: EPOXY—Enabling Robust Protection for Bare-metal Systems

Abraham A. Clements*, Naif Saleh Almakhdhub†, Khaled S. Saab‡, Prashast Srivastava†,
Jinkyu Koo†, Saurabh Bagchi†, Mathias Payer†
*Purdue University and Sandia National Laboratories, clemen19@purdue.edu
†Purdue University, {nalmakhd, srivas41, kooj, sbagchi}@purdue.edu, mathias.payer@nebelwelt.net
‡Georgia Institute of Technology, ksaab3@gatech.edu

***Abstract*—Embedded systems are ubiquitous in every aspect of modern life. As the Internet of Thing expands, our dependence on these systems increases. Many of these interconnected systems are and will be low cost bare-metal systems, executing without an operating system. Bare-metal systems rarely employ any security protection mechanisms and their development assumptions (unrestricted access to all memory and instructions), and constraints (runtime, energy, and memory) makes applying protections challenging.**

To address these challenges we present EPOXY, an LLVM-based compiler. It uses a novel technique, called privilege overlaying, wherein operations requiring privileged execution are identified and only these operations execute in privileged mode. This provides the foundation on which code-integrity, adapted control-flow hijacking defenses, and protections for sensitive IO are applied. We also design fine-grained randomization schemes, that work within the constraints of bare-metal systems to provide further protection against control-flow and data corruption attacks. Our evaluation shows these defenses are effective and operate within the constraints of bare-metal systems.

I. Introduction

Many of the devices in the Internet of Things are bare-metal systems. These systems execute a single application directly on its hardware, without the use of an Operating System (OS). Some of these systems are: Amazon's dash button, smart locks, wireless systems on a chip, and memory cards. As the prevalence of the Internet of Things increases, such devices become increasingly connected, exposing them to network based exploitation. We rely on these systems to provide secure and reliable computation, communication, and data storage. Yet they lack even basic defenses against code injection or control-flow hijacking.

This lack of defenses arises from design assumptions, and system constraints that make applying defenses particularly challenging. In the design, of a bare-metal program, the programmer needs access to all instructions and all memory locations. This is necessary because a single program manages the hardware and implements program logic. Managing parts of the hardware, for example the Memory Protection Unit (MPU), requires execution in privileged mode (*.i.e.*, root level access). Since there is only one program on the system it executes exclusively in privileged mode. This prevents effective implementation of basic foundational defenses like DEP. As the MPU which is used to enforce DEP can be trivially disabled by a privileged memory corruption vulnerability. Further complicating the application of defenses on bare-metal systems are tight constraints on run-time, memory, and energy consumption. Combined, these challenges have prohibited adoption of any protection for bare-metal systems. Thus, in practice bare-metal systems are vulnerable to manipulation of sensitive IO, stack smashing [4], code injection, ROP attacks [7], and data corruption attacks.

To address these challenges, we developed EPOXY (Embedded Privilege Overlay on X hardware with Y software) [2], an—LLVM based compiler that brings both generic and system-specific protections to bare-metal programs. EPOXY adds protection against code injection, control-flow hijack, data corruption attacks, and direct manipulation of IO. Central to our design is a lightweight *privilege overlay*, which solves the dichotomy of allowing the developer to assume access to all instructions and memory while restricting access at runtime. To further protect the program, EPOXY applies, fine-grained diversification techniques which use all available memory, and strong stack protections, by adapting SafeStack [3] to bare-metal systems.

II. EPOXY

EPOXY modifies device startup to reduce privileges of the entire program and configure the MPU. The MPU's configuration enforces DEP and protects sensitive IO, thus providing code-integrity. However, reducing privileges of the entire program breaks functionality. Thus, EPOXY uses static analysis to identify instructions requiring privileges and only elevates those instructions. Conceptually, this is like overlaying the original program with a mask which elevates just the instructions requiring privileges. In some ways, the privilege overlay is similar to a program making an operating system call and transitioning from unprivileged mode to privileged mode. However, here, instead of being a fixed set of calls which operate in the operating system's context, it creates a minimal set of instructions that execute in their original context (the only context used in a bare-metal application execution). By elevating just those instructions, we simplify the development process and limit the power of memory corruption vulnerabilities.

EPOXY uses two static analyses to place instructions in the privilege overlay. The first performs instruction matching to identify instructions defined by the architecture that require privileged execution. The second static analysis examines the back slice of each load and store to determine if it may load or store from a constant address. When found, loads and stores to constant addresses, which are restricted by the MPU configuration, are added to the privilege overlay. For each instruction in the privilege overlay EPOXY emits additional instructions that obtain privileges prior to the original instruction's execution, and then reduces privileges immediately after. To further protect the program, EPOXY adapts SafeStack [3] to bare-metal systems, protecting against control-flow hijack attacks. SafeStack is a split stack defense that uses static analysis to move local variables which may be used in an unsafe manner to a separate *unsafestack*, thereby protecting return pointers and *safe* local variables. EPOXY employs global data randomization with padding, function randomization with padding, and register selection randomization to protect against data corruption and code reuse attacks. These randomization techniques are adapted to use all of the memory on the systems to maximize the entropy they produce.

III. Evaluation

We evaluate EPOXY's performance on 75 applications from the BEEBs benchmark suite [5] and three representative IoT applications. Overheads for the benchmarks average 1.6% for runtime and 1.1% for energy. For the IoT applications, the average overhead is 1.8% for runtime, and 0.5% for energy. We also examine the memory usage on the IoT applications, as shown in Table I. We find the IoT applications requires less than 3,400 bytes of additional code space and under 200 bytes of additional RAM. Our results for execution time, power usage, and memory usage show that our techniques work within the constraints of bare-metal applications.

To understand the security protections of our system, we compare with FreeRTOS-MPU [1], an embedded OS that uses the MPU to isolate different threads of execution. We compare the number of instructions executed and the number of privileged instructions for the IoT applications. EPOXY uses significantly fewer privileged instructions, see Table II. To evaluate EPOXY's diversification techniques, we use a ROP compiler [6] to identify gadgets in 1,000 variants for each of the three IoT applications. No gadget survives across more than 107 binaries, as shown in Table III. This implies that an adversary cannot reverse engineer a single binary to create a ROP chain with a single gadget that scales beyond a small fraction of devices, and shows the effectiveness of our diversification techniques.

TABLE I
Increase in memory usage for the IoT applications

					Stack			
App	Text		Global Data		SafeStack		Priv. Over.	
PinLock	3,390	29%	14.6	1%	104	25%	0	0%
FatFs-uSD	2,839	12%	18.2	1%	128	3%	36	1%
TCP-Echo	3,249	8%	7.2	0%	128	29%	0	0%

TABLE II
Comparison FreeRTOS vs. EPOXY showing memory usage, number of instructions executed (Exe), and number of privileged instructions (PI).

App	Tool	Code	RAM	Exe	PI
PinLock	EPOXY	16KB	2KB	823K	1.4K
	FreeRTOS	44KB	30KB	823K	813K
FatFs-uSD	EPOXY	27KB	12KB	33.3M	3.9K
	FreeRTOS	58KB	14KB	34.1M	33.0M
TCP-Echo	EPOXY	43KB	35KB	310.0M	1.5K
	FreeRTOS	74KB	51KB	321.8M	307.0M

TABLE III
Survival of ROP gadgets over 1,000 variants.

		Num. Surviving				
App	Total	2	5	25	50	Last
PinLock	294K	14K	8K	313	0	48
FatFs-uSD	1,009K	39K	9K	39	0	32
TCP-Echo	676K	22K	9K	985	700	107

IV. Conclusion

EPOXY enables the simple application of state-of-the-art defenses to bare-metal systems. We maintain the developers assumption of access to all instruction and all memory locations and apply the defenses within the constrains (runtime, memory, and energy) of bare-metal systems. Our mechanisms provides code-integrity, protection of sensitive IO, and defenses against control-flow hijacking and data corruption attacks. In summary, EPOXY fast forwards bare-metal applications security several decades, providing necessary defenses for today's connected systems. The source code of EPOXY is available at **https://github.com/HexHive/EPOXY.**

Acknowledgments

This material is based in part on work supported by the National Science Foundation under Grant Numbers CNS-1464155 and CNS-1548114. Any opinions, findings, and conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. This work is also supported by Sandia National Laboratories, a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energys National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] FreeRTOS-MPU. http://www.freertos.org/FreeRTOS-MPU-memory-protection-unit.html
[2] A. A. Clements, N. S. Almakhdhub, K. S. Saab, P. Srivastava, J. Koo, S. Bagchi, and M. Payer, Protecting bare-metal embedded systems with privilege overlays, In *IEEE Symp. on Security and Privacy*. IEEE, 2017.
[3] V. Kuznetsov, L. Szekeres, M. Payer, G. Candea, R. Sekar, and D. Song, Code Pointer Integrity, *USENIX Symp. on Operating Systems Design and Implementation*, 2014.
[4] A. One, Smashing the stack for fun and profit, *Phrack Magazine*, vol. 7, no. 49, pp. 14–16, 1996.
[5] J. Pallister, S. J. Hollis, and J. Bennett, BEEBS: open benchmarks for energy measurements on embedded platforms, *CoRR*, vol. abs/1308.5174, 2013.
[6] J. Salwan, ROPgadget - Gadgets Finder and Auto-Roper, 2011. http://shell-storm.org/project/ROPgadget/
[7] H. Shacham, The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86), In *ACM Conf. on Computer and Communications Security*, 2007, pp. 552–561.