Protecting Embedded Systems from Zero-Day Attacks

Stephen Taylor
Thayer School of Engineering
Dartmouth College
Hanover, USA
stephen.taylor@dartmouth.edu

***Abstract*—Recently, a quiet revolution in embedded systems technology has occurred with the appearance of System-on-Chip (SoC) devices. These couple Field Programmable Gate Array (FPGA) logic, with a broad array of peripherals, encryption hardware, and high-performance multi-core processors -- all within the boundary of a single chip. This innovation has been accompanied by a coming-of-age in High-Level Synthesis, enabling hardware-software co-design in a systems programming language, such as C. Concurrently, the emergence of light-weight OCI-compliant containers have revolutionized the distribution and maintenance of vertically integrated software stacks. This short position paper explores how these advances can be leveraged to improve the security of embedded systems and, in particular, protect against sophisticated attacks employing kernel-level, zero-day exploits.**

Keywords—cyber-attack, security, trust, zero-day exploit, embedded-system, advanced persistent threat.

I. Introduction

In order to understand how new technologies can improve security, it is valuable to first qualify the perceived threat model. This paper is concerned with sophisticated threats that operate through the general process outlined in Fig. 1 [1]. The process involves several steps that begin with *surveillance* to determine which vulnerabilities exist at the target. These vulnerabilities may revolve around specific people, processes, organizations, and/or network infrastructures. Based on the available vulnerabilities, campaigns may incorporate network attack components; These involve multiple initial *touches* on target networks to develop points-of-presence through some form of *implant*.

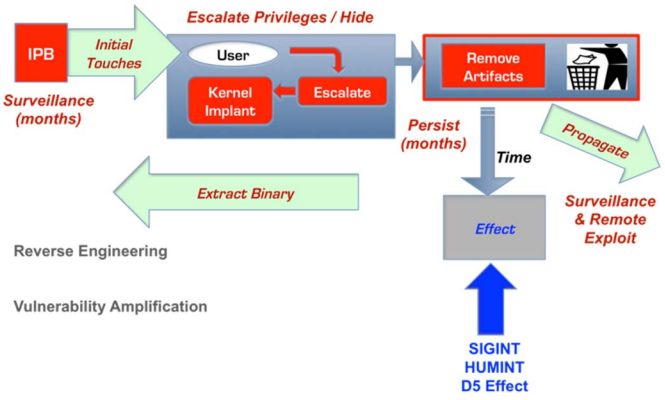

Fig 1. Threat Model

The initial touches may employ remote user- or kernel-level zero-day exploits, insiders, theft of legitimate credentials, supply chain interdiction, physical attacks at an end-point, electro-magnetic and infrastructure weaknesses. Development from the initial points-of-presence may involve privilege escalation, removing exploit artifacts, and hiding behavior. On-going surveillance may involve obtaining binary codes, collecting devices from the field, or gathering configuration information. These enable *reverse-engineering* which lead to discovery of additional attack vectors. Implants *persist* for a time sufficient enough to carry out some malicious effect, obtain useful information, or propagate to other systems. Propagation is considerably simplified by the fact that many systems employ the same bare-metal software image or operating system core, leading to *vulnerability amplification*.

Unlike the time to execute an exploit, the time spent in surveillance and persistence may range from minutes to *months or even years* depending upon the intended effect. Moreover, the presence of an intrusion may *never* be detected by network defenses but instead be recognized indirectly by a noticeable deviation from expected behavior or intelligence data gathered separately. Several factors make detection unreliable: signature-based intrusion detection systems do not detect zero-day exploits unless they resemble known exploits; anomaly detectors are equally unreliable since not every malicious action is anomalous (i.e. attacks may be crafted to resemble normal operation), and not every anomaly is malicious (i.e. a large-numbers of false-alarms may obscure the attack). Consequently, to assure system operation a comprehensive defense-in-depth is required that significantly increases attacker workload, limits the attackers window of opportunity, and provides the ability to operate through attacks.

II. System-on-Chip Devices

The emergence of SoC devices in 2012 with the availability of the Xilinx Zynq architecture, shown in Fig. 2., heralded a new wave of embedded systems technology. Throughout 2013, devices from both Xilinx and Altera gained considerable market traction from system designers pursuing consolidation and hardware acceleration: system components previously implemented in off-chip FPGA could now be integrated within the processor boundary reaping rewards afforded by high-performance AXI-standard interfaces coupling processing cores to on-chip FPGA resources. 2014 saw the introduction of 64-bit ARM cores and interest in the market segment by Intel

Corporation, culminating in the purchase of Altera. 2015 saw a radical performance improvement with the introduction of the Xilinx Ultrascale MPSoC featuring quad-core 64-bit ARM A53 processors, dual-core R4, and several other enhancements. Intel subsequently made several announcements concerning SoC offerings.

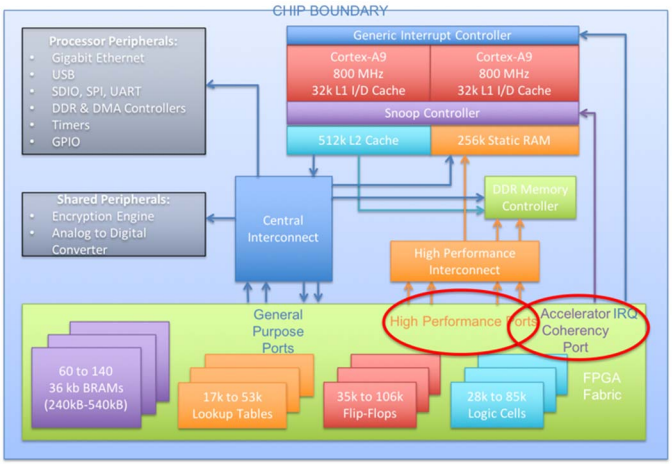

Fig 2. The Zynq Architecture

The collocation of processing cores, FPGA fabric, and peripherals within the processor boundary enables a radical change in security footing: shrinking the system boundary-of-trust to be co-located with the physical chip boundary [2]. This change ensures there are no exposed busses external to the trust-boundary thereby mitigating the opportunity for reverse-engineering. In consequence, to compromise a lost or stolen device requires a significant increase in attack-workload: high-value information -- sensitive algorithms, encryption keys, hashes, etc. – must be obtained by exotic techniques, such as acid etching, which destroys the device, or side-channels, which are time-consuming, inexact and costly. Moreover, on-chip peripherals, shown in grey in Fig. 2., allow tamper sensors to be directly connected into the trust boundary; This offers the opportunity to mitigate reverse engineering by destroying critical information stored in the FPGA, thereby adding a *time-dependent* limit to the attacker's window of opportunity.

A second and more subtle aspect of this architectural design is the presence of high performance AXI-master ports, shown circled in red in Fig. 2. These link the FPGA fabric directly into the processor memory hierarchy and provide a new opportunity: the ability to look up directly into the context of the processor from a hardware base-of-trust in the FPGA. This capability allows *unobservable* hardware monitors to be hidden in the FPGA with full visibility on all software running on the processor, even if that software is executing with kernel-level privileges [2]. As a result, if a malicious implant is able to gain a point-of-presence, it can potentially be detected; moreover, the fact that it has been detected cannot be observed by the malicious implant since it has no downward facing visibility into the hardware. A wide-range of system attributes can be monitored using this concept: code integrity, invariants associated with data and control, network traffic characteristics, and program behavioral characteristics – all in a manner that is unobservable to the implant [3].

Hidden hardware monitors are particularly valuable in embedded systems since they operate continuously. In order to compromise system software, its code must be patched at least once to periodically enter malicious code, and again to re-enter the control systems normal flow of operation. Consequently, rather than detect zero-day intrusions, a preferable approach is to continuously monitor *running* code for *any* single-bit change from the FPGA. A Xilinx IP-block that provides this zero-day monitoring capability is available from MicroArx: The block can be integrated into typical control loops providing a signal if any code patch occurs.

There are multiple corrective actions that can be taken when a detection occurs: the signal could be used to simply halt the processor in extreme circumstances, invoke additional monitoring or journaling capabilities, or alert an administrator. The preferred methodology is to provide an additional out-of-band network channel in the FPGA over which to raise the alert [2]. This channel can notionally be connected to a separate monitoring network making it possible for administrators to notice the alert and take appropriate actions without fear of informing a malicious implant.

There is one corrective action that is particularly valuable: to *refresh* the control system code with via a read-only gold-standard copy, obtained via the out-of-band channel [2]. This effectively wipes away the implant thereby mitigating persistence of the implant. The refresh can be carried out semi- or fully-automatically, at sub-microsecond speeds, allowing the system to operate through attacks. Moreover, refresh can also be initiated non-deterministically (i.e. un-predictably) at random intervals, even if no detection has occurred; thereby imposing another *time-dependent* limit on the adversary's window of opportunity.

Hidden hardware monitoring, when coupled with automated code refresh, removes an implants ability to exploit, hide, and persist. Unfortunately, it does not prevent use of the same exploit repeatedly against a single system or use of the same exploit against copies of the system running on other equipment. To mitigate this vulnerability amplification, it is possible to generate a diverse collection of functionally equivalent binary codes, each of which performs the same embedded function but with a completely different memory layout [4]. These codes have the property that *no two variants share the same exploitable jump instruction*. This is achieved through two non-deterministic operations: padding code blocks with a random number of no-ops to move jumps within code blocks (i.e. loops and branches) and randomly shuffling functions to change jump targets. As a consequence, an exploit crafted against any variant can compromise only that variant; refreshing system code with a new variant removes the ability to re-exploit the code. Since the engineering cost of zero-day exploits is high and the time to develop them is often measured in months, these techniques significantly raise the barrier to entry for the adversary and lower the utility of the resulting exploit. A *diversity loader* employing these ideas is available from MicroArx integrated with the normal SoC bootstrapping chain.

III. High-Level Synthesis

While on-chip FPGA's provide a valuable tool for hardening embedded systems, the difficulty of circuit design has long limited their application. Fortunately, in tandem with SoC development, High-Level Synthesis (HLS) has reached a new level of maturity: it can now be used robustly for non-trivial projects and is integrated within SoC co-design tool chains. Fig. 3 shows how these tools operate [3].

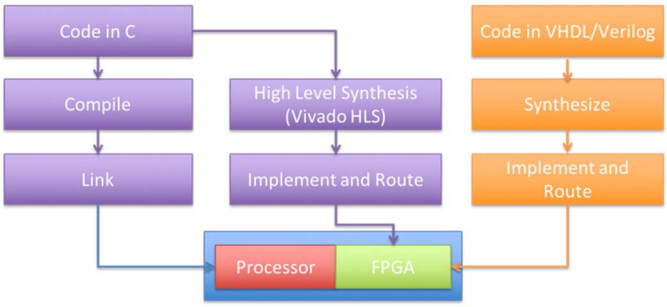

Fig 3. Hardware-Software Co-design

FPGA logic configuration is usually described using a hardware description language (HDL) such as VHDL or Verilog. HDLs are transformed via synthesis and implementation tools into a *bitfile* for use in configuring the FPGA logic, as shown on the right (orange) column of Fig. 3. This process is analogous to the normal software design process, shown on the left column of Fig. 3, where a C-program, is compiled, then linked into an executable image, and loaded for execution onto the processor.

High-level synthesis allows portions of a system to be developed in C-code then *automatically* transformed into a loadable FPGA bit-stream file for the FPGA. This is achieved using the Vivado HLS tool chain shown in the center column of Fig. 3. The resulting bit-stream file can be uploaded into the FPGA and the executable file uploaded to the processor via an out-of-band channel [2]. Upon completion of the upload, aspects of the FPGA design cannot be viewed or modified from the processor.

This process allows software-like development and deployment flexibility while maintaining the security benefits of a hardware root-of-trust. It also allows regression testing to be performed using test-benches written in C and opens the door to hardware development for the community of systems programmers. In addition, the ability to define a separable component of a system and render it directly into hardware opens the door to *separable accreditation* of complex systems; thereby reducing the overall cost and time to add new features to a large accredited system running on the processor.

MicroArx has available a wide selection of independent HLS building blocks that can be combined and configured to build security related products and embedded systems. These blocks include network adapters, packet inspection engines, protocol monitors and translators, encryption, and hashing algorithms. The blocks are currently employed in the MicroArx DT-Series (see: microarx.com) network diodes; these allow passive monitoring of Modbus RTU and Modbus TCP, with data exfiltration via Cellular, Ethernet, and 802.11 connections to standalone servers or cloud analytics. A Ruby design platform – Apiotics (see: apiotics.com) -- is also available, allowing embedded systems to be configured, deployed, and controlled through web-applications; the platform was recently featured in a RailsConf 2018 programming workshop.

IV. OCI-Compliant Containers

Concurrent with the evolution of HLS for SoC co-design, there has been another radical shift in technology: this time in the organizational principles associated with *cloud computing*. The move has been motivated primarily by the desire to speed application deployment, share computing resources more efficiently, and improve portability between testing and production environments. The move has resulted from the realization that web-based applications, to work reliably, must be associated with a very specific version of the operating system with associated patches, language packs, binaries, libraries, security certificates, etc. By grouping an application with its dependencies, and deploying them as a unit, the problem of version dependency is eliminated allowing applications to become more portable: what works in a test and evaluation environment will thus more likely work in a production cloud environment.

This alternative paradigm – *containers* – typified by BSD Jails and Docker -- is fast becoming the mainstay of cloud computing. Each container has its own software and network stack, associated with a particular operating system image, application binaries and libraries etc.; multiple containers share the underlying hardware via a *single kernel* with an associated middle-ware layer, such as the Docker Engine, providing access to network daemons caching images and dependencies. Since there are no replicated operating system images, containers have low resource requirements, deploy more quickly, and are standardized based on a description of their needed resources.

Unfortunately, the software stack associated with cloud-computing -- hypervisors, their management interfaces, virtual machines, and on up through the kernel into container systems -- have steadily grown in size and complexity over the years. It is well known that the number of vulnerabilities in software is directly related to the size of the source code [5]. Attackers have already shifted to methods for detecting hypervisors and compromising them directly; a class of attacks that can be expected to grow in frequency and sophistication [6]. The bottom-line here is that hypervisors are increasingly as much of a security liability as the kernels they are expected to isolate and protect. Moreover, despite the hype associated with wrapping and isolation of containers, kernel level security and vulnerability amplification are *not* significantly improved.

The importance of these ideas to security of embedded systems lies in the ability to *reliably* discern a particular set of operating system functions, patches, environment values, designated library versions, and device drivers and to deploy these with an application. This begs the question: what represents a minimal stack to support containers on SoC hardware?

While the software stacks supporting cloud-environments are enormous, the capabilities required to use containers natively on SoC devices, purely as an encapsulation mechanism, are simple: They use an Application Programmers Interface (API) comprising just five core functions; *create* a container, *start* the container running, *query* its state, *kill* its execution, and *delete* it from the system. These functions are at the core of the Open Container Initiative (OCI) – a light-weight set of open standards for specifying and running containers.

On SoC devices, these capabilities may be implemented either in software or directly in hardware using an SoC's FPGA resources. The former provides considerable flexibility, while the latter provides *hardware-enforced container isolation* within the base-of-trust provided by the FPGA. MicroArx has available a light-weight, OCI-compliant *nano-marshal* capable of deploying multiple containers on a variety of SoC development boards. These containers support the development of bare-metal containers for embedded systems. The nano-marshal will be released open-source to the embedded systems community at the conclusion of beta-testing. An FPGA version incorporating a variety of security enhancements is also under development.

V. Summing Up

This position paper has explored three evolving technologies from the viewpoint of embedded system security: SoC devices with on-chip FPGA's, high-level synthesis, and OCI-compliant containers.

SoC devices provide an unprecedented opportunity to mitigate reverse engineering by shrinking the protection boundary to the chip-boundary. Hidden hardware monitors provide the ability to detect change and thereby mitigate zero-day exploits without detection. Code refresh provides a mechanism to mitigate implant persistence. Diversity mitigates re-infection and vulnerability amplification. Anti-tamper sensors, out-of-band management, and two-factor authentication implemented within the chip-boundary boundary mitigate insider attacks and reverse-engineering on lost or stolen devices. Finally, rapid technology evolution and accreditation are supported through HLS, OCI-compliant containers, and Apiotics.

Taken together these ideas provide a unique opportunity to encapsulate, reliably deploy, and protect embedded systems from advanced persistent threats involving kernel-level zero-day exploits. The resulting defense-in-depth allows embedded system designers to *own the base-of-trust in hardware* – an essential characteristic that distinguishes the approach from conventional intrusion detection systems.

Acknowledgment

This paper draws heavily on joint research work with a number of talented graduate students at Dartmouth College over the last decade. Among these, the Ph.D. research of Jason Dahlstrom, Morgon Kanter, Michael Henson, and Stephen Kuhn, in particular, have significantly enhanced the strategic vision presented here for combatting kernel-level zero-day exploits and advanced persistent threats in embedded systems.

References

[1] M. Henson and S. Taylor, "Beyond full disk encryption: protection on security enhanced commodity processors," Proceedings of the 11th International Conference on Applied Cryptography and Network Security (ACNS '13), pp 307-321, June 25-29, 2013.

[2] J. Dahlstrom and S. Taylor, "System-on-chip data security appliance and methods of operating the same", U.S. Utility Patent 15/077,519, 2016 (Allowed).

[3] J. Dahlstrom and S. Taylor, "Hardware-based code monitors on hybrid, processor-FPGA system-on-chip architectures", pp 968-973, MILCOM 2015.

[4] M. Kanter and S. Taylor, "Diversity in cloud systems through runtime and compile-time relocation," 2013 IEEE International Conference of Technologies for Homeland Security (HST) Nov, 2013.

[5] P.K. Pandey and V. Tiwari, "Reliability issues in open source software," International Journal of Computer Applications, p 1, 34th edn., 2011.

[6] R. Denz and S. Taylor, "A survey on securing the virtual cloud," Journal of Cloud Computing: Advances, Systems, and Applications, Vol 2, Issue 1, Nov 2013.