Security Challenges in Mobile and IoT Systems

Sandip Ray
NXP Semiconductors Inc.
Austin, TX, USA.
sandip.ray@nxp.com

Jayanta Bhadra
NXP Semiconductors Inc.
Austin, TX, USA.
jayanta.bhadra@nxp.com

***Abstract*— Security is a critical challenge for modern embedded, mobile, and IoT devices. One the one hand, these devices contain sensitive assets that must be protected from unauthorized access. On the other hand, high design complexity, aggressive time-to-market, and a complex, global supply-chain provide numerous opportunities for introduction of errors, vulnerabilities, and security backdoors that can be exploited on-field to compromise the device. In this paper we look at some of the security challenges in this era: questions on the root of trust and certification challenges in mobile and embedded devices, conflicts and trade-offs between security and functional debug, and vulnerability results from widespread application of electronic design automation (EDA) tools for system synthesis.**

I. Introduction

Computing devices pervade in our everyday life now, and include smartphones, tablets, wearables, implants, smart sensors, etc. The number and diversity of these devices is increasing at an explosive rate, with the advent of the Internet-of-Things era. We are moving towards a world with projected 50 billion smart, connected computing devices by 2020 from a "mere" 500 million in 2003 [1], representing the fastest growth point by a large measure in the history of computing.

A critical gating factor for this new regime is security. With computing devices being employed for a large number of highly personalized activities (*e.g.*, shopping, banking, fitness tracking, providing driving directions, etc.), these devices have access to a large amount of sensitive, personal information which must be protected from unauthorized or malicious access. On the other hand, communication of this information to other peer devices, gateways, and datacenters is in fact crucial to providing the kind of adaptive, "smart" behavior that the user expects from the device. For example, a smart fitness tracker must detect from its sensory data (*e.g.*, pulse rate, location, speed, etc.) the kind of activity being performed, the terrain on which the activity is performed, and even the motivation for the activity in order to provide anticipated feedback and response to the user; this requires a high degree of data processing and analysis much of which is performed by datacenters or even gateways with higher computing power than the tracker device itself. The communication and processing of one's intimate personal information by the network and the cloud exposes the risk that it may be compromised by some malicious agent along the way. In addition to personalized information, computing devices contain highly confidential collateral from architecture, design, and manufacturing, such as cryptographic and digital rights management (DRM) keys, programmable fuses, on-chip debug instrumentation, defeature bits, etc. Malicious or unauthorized access to secure assets in a computing device can result in identity thefts, leakage of company trade secrets, even loss of human life. Consequently, a crucial component of a modern computing system architecture includes authentication mechanisms to protect these assets.

Modern computing systems are typically developed as system-on-chip (SoC) designs, *i.e.*, a single integrated circuit encompassing the system functionality. An SoC design involves composition of a large number of design modules (often referred to as *intellectual properties* or IPs) that coordinate with through a number of on-chip communication fabrics to implement the system functionality. Secure assets in such a design are sprinkled across the different IPs, and their access control requirements are defined by a collection of highly complex *security policies*. The policies specify the conditions under which a security asset can be accessed at any point in the system execution. An SoC design consequently requires a *security architecture*, *i.e.*, a mechanism of authentication to ensure that the system enforces and manages these policies.

In spite of its obvious importance, there has been little work on systematizing the development of security assurance for modern SoC designs. Security architecture and validation in current industrial practice involve ad hoc, point approaches based on creative insights from designers, architects, and validators. As systems scale to higher and higher complexity, it is difficult for such approaches to scale. Exacerbating the issues is the fact that security requirements, security policies, and design invariants exploited to implement them are rarely formalized or even documented. This makes it impossible to validate if the final design indeed enforces proper authentication for all design assets: such information is buried within a plethora of architectural and design documents, specified in ambiguous natural language descriptions, and often left implicit. Unsurprisingly, security vulnerabilities are abound in modern SoC designs, as evidenced by the frequency and ease in which activities like identity theft, DRM override, device jailbreaking, etc. are performed.

In this paper, we outline some key research needs in SoC system-level security. This goal of the paper is to provide a broad overview of security validation activities across the system life-cycle, and point out some of the gaps and challenges as we move into the IoT era. There has been some progress on addressing these gaps, and such related work is discussed in the context.

The remainder of the paper is organized as follows. Section II provides an overview of security activities, requirements, and specifications from the SoC design perspective. We discuss challenges with functional debug in Section III, hardware/software validation challenges in Section IV, and specification challenges in Section V. Section VI discusses some upcoming challenges as we move into the IoT regime. We conclude in Section VII.

II. Background

A. Security Policies

Security policies [2], [3] identify the authentication, access, and protection requirements for the different assets in the design. At a high level, the policies are typically instances of confidentiality, integrity, and availability requirements [4]. The role of a policy is to define an instantiation of these requirements for specific assets, and provide an "actionable" specification for the SoC system architect and designer on the protection mitigation strategies that need to be implemented. For example, the following sample policies define some of the policies for cryptographic keys, programmable fuses, and executable firmware. Note that these policies are merely illustrative and do not represent the security policy set of any specific company or design.

1) During boot, keys transmitted by the crypto engine cannot be observed by any IP other than its intended target.
2) An on-chip fuse can be updated for silicon validation but not after production.

Note that the access restrictions specified by a policy can be a function of both the point in the execution (*e.g.*, boot vs. normal) or point in the system life-cycle (*e.g.*, validation vs. production).

B. Security Along SoC Design Life-Cycle

Fig. 1 provides a high-level overview of the SoC design life-cycle. Each component of the life-cycle, of course, involves a large number of design, development, and validation activities. Here we summarize the key activities involved along the life-cycle, that pertain to security.

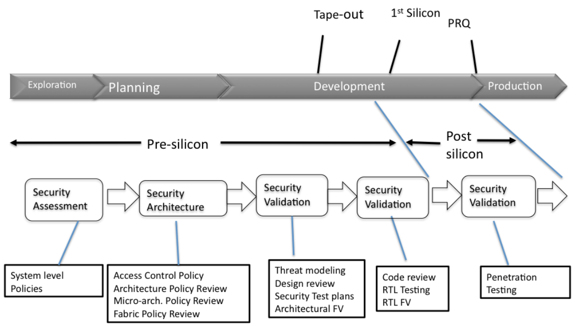

Fig. 1. SoC Design Life-Cycle

Risk Assessment. Security requirements definition is a key part of product planning, and happens concurrently with (and in close collaboration with) the definition of architectural features of the product. This process involves identifying the security assets in the system, their ownership, and protection requirements, collectively defined as *security policies* (see below). The result of this process is typically the generation of a set of documents, often referred to as *product security specification* (PSS), which provides the requirements for downstream architecture, design, and validation activities.

Security Architecture. The goal of a security architecture is to design mechanisms for protection of system assets as specified by the PSS. It includes several components, including (1) identifying and classifying potential adversary for each asset; (1) determining attacker entry points, also referred to as threat modeling; and (3) developing protection and mitigation strategies. The process can identify additional security policies — typically at a lower level than those identified during risk assessment (see below) — which are added to the PSS. The security definition typically proceeds in collaboration with architecture and design of other system features, including speed, power management, thermal characteristics, etc., with each component potentially influencing the others.

Security Validation. Security validation represents one of the longest and most critical part of security assurance for industrial SoC designs, spanning the architecture, design, and post-silicon components of the system life-cycle. The actual validation target and properties validated at any phase, of course depends on the collateral available in that phase, *e.g.*, we target, respectively, architecture, design, implementation, and silicon artifacts as the system development matures. Below we will discuss some of the key validation activities and associated technology. One key component of security validation is to develop techniques to subvert the advertised security requirements of the system, and identify mitigation measures. Mitigation measures for early-stage validation targeting architecture and early system design often include significant refinement of the security architecture itself. At later stages of the system life-cycle, when architectural changes are no longer feasible due to product maturity, mitigation measures can include software or firmware patches, product defeature, etc.

III. Security/Debug Conflicts

In this section, we consider one key challenge in security assurance for modern SoC designs, *viz.*, challenges between security and debug/validation requirements. Debug and validation typically requires observability of internal design states of the system. In particular, post-silicon validation requires instrumentation of the design with significant additional circuitry, referred to as Design-for-Debug or DfD, which enables the debugger to observe the internal design behavior during silicon execution. These DfD features, obviously, incur a significant security risk, *e.g.*, it is also possible for attackers to exploit these features to gain access to critical internal assets. Even for pre-silicon validation there are coverage monitors, assertion checkers, etc. built into RTL which might be exploited for security violation. The security/debug trade-off is about enabling debug while ensuring protection of secure assets.

The security/debug trade-off is just an instance of the more general trade-off between security and interoperability. High confidentiality and integrity requirements can be typically realized by restricting functionality of the design, *e.g.*, an obvious way to protect an asset from unauthorized access is to prohibit access to it. Unfortunately, over-zealous access restriction can have a significant adverse impact on the usability of the product and even make it vulnerable to denial-of-service attacks. This is one reason why security policies can become highly subtle: they need to ensure the the system functions properly while still providing robust protection against malicious access. However, debug requirements are different from most other usability requirements in this respect because they do not directly affect functional behavior of the design but only their validation methodologies. Since observability requirements from debug depend upon the potential errors in the design which cannot be predicted a priori, it is difficult to create robust security policies to account for debug requirements. Finally, an "irony" in the trade-off between security and debug requirements arises from the fact that one key component of the debug and validation involves checking security properties themselves: it is not uncommon for validation of security properties to be stumped by the constraints imposed by security on observability.

To the best of our knowledge, there has been no comprehensive solution to address the security/debug trade-offs. Other papers [5] have looked at the challenges that must be addressed by any such solution. Below we summarize some key aspects of the challenge.

HVM Considerations. High-volume manufacturing test is the process of identifying manufacturing defects during production. This is done by placing the fabricated silicon in a tester, where it is exercised with a large number of test vectors. The test patterns are generated by accounting for the functional definition of the design under test, the target faults, a fault model, the fabrication process technology, etc. The accuracy of coverage inferred from the results of these tests is highly sensitive to the test patterns being applied and the fidelity of the silicon design with respect to its pre-silicon netlist model. Consequently, it is important that irrespective of security constraints and access control restrictions, the test patterns work the same way as much as possible on silicon designs as expected from pre-silicon models, and their results accurately observed.[1] Furthermore, it must be possible to have a simple access to the IP being exercised with the test, without requiring too many workarounds.

Reusability: A key source of complexity in the current state of the practice is the need to manually identify assets and accesses for different products and usage scenarios. This job is highly tedious and error-prone. Consequently, solution to the problem must provide a reusable infrastructure for systematically identifying and classifying assets and analyzing usage scenarios.

Late Variability: DfD is notorious for late changes in requirements and implementation. Indeed, DfD requirements can change during IP design, SoC integration, or even after a silicon stepping; the latter can happen on realization that observability or control of certain signals is critical for a future stepping. Consequently, any solution for addressing security challenges with DfD must be easy to adapt with such changing requirements. In particular, it should be possible to quickly validate an updated DfD architecture against a given set of security policies and identify vulnerabilities.

Self-Securability It is obvious that any architecture introduced to address the security-validation trade-off must be self-securing and must not introduce additional security back-doors (or complexity with debug).

Architecture: It is critical for any architecture that permits the trade-off to be centralized. The reason is that a decentralized architecture (both for security and DfD) is

[1]We say "as much as possible" since it is not possible to have exact equivalence, *e.g.*, if the system uses dummy keys in debug mode and test result depends on value of keys.

difficult to follow and can accidentally break or introduce vulnerability. To circumvent this possibility, it is critical that the architecture can be viewed as a centralized IP which can itself be effectively analyzed for possible violation of either security or debug requirements.

IV. Hardware/Software Challenges

Another critical challenge in the development of security solutions for modern computing devices is the tight coupling between hardware and software components. Traditionally, general-purpose, programmable computing systems, *e.g.*, desktops, laptops, and servers, were built upon a standard hardware architecture such as X86, MIPS, SPARC, etc. with a published, standardized instruction set. This allowed decoupled design analysis of hardware and software components. Of course there were embedded systems, *e.g.*, automobile control, medical instruments, etc., with tightly coupled hardware/software modules; however, these systems were designed to implement a small set of use-cases and the individual modules were relatively simpler with well-defined security requirements. This clear demarcation started to break down with the advent of mobile systems like smartphones and tablets, and the barrier has broken down completely with the advent of the large diversity of IoT devices. These devices are highly complex, programmable, and have computing power of the scale higher than a general-purpose computer of a few years back. On the other hand, they also exhibit the tight coupling of hardware and software which was the hallmark of embedded systems of the past. Indeed, the very definition of embedded systems has changed from "a system with hardware and software components targeted for a specific function" [6] to "any computing system that is not a desktop, laptop, or server" [7].

Why does this tight hardware/software coupling cause challenges to security? Many security vulnerabilities arise because of errors or misconfiguration at the *interface* of hardware and software components. For example, an error in the device/driver interface may result in a buffer overflow for a specific execution usage which can be exploited by a malicious attacker to induce a code-injection attack. The trouble with such vulnerabilities is that since they occur as a result of the breach of contract between hardware and software they cannot be detected during either hardware or software validation and a robust co-validation framework is necessary to exhibit such problems. On the other hand, traditional validation technologies are typically inadequate for hardware/software co-validation. Hardware and software components are often abstracted differently, with different formalisms, tools, and methodologies involved. To exacerbate the issue, hardware and software components are typically developed concurrently by different teams or vendors, and continue evolving along the entire system design/validation life-cycle. The key challenge then is: "How can we validate (evolving) software that is being developed to execute on a hardware platform whose design itself is undergoing change at the same time?"

The continuous evolution problem alluded to above is, of course, a general problem for validation of modern computing devices and not necessarily confined to security. To address this issue, current industrial practice involves developing prototypes of the hardware design at different levels of abstraction specifically for the purpose of co-validation. Some prototyping frameworks include (1) virtual platforms, *i.e.*, an abstract software model of the hardware platform [8], (2) emulation and FPGA models, and (3) adapting a previous-generation silicon as a platform for testing the next-generation software (often called an "$\mathrm{N}-1$ silicon" solution). While many of these prototyping solutions work well for functional validation, they are unfortunately inadequate for security validation. In particular, what makes the co-validation problem particularly vexing for security is that security compromises typically happen as a consequence of very specific corner-case violations. On the other hand, most prototyping solutions diverge from the actual design in some way, either because of abstraction (*e.g.*, for virtual platforms) or to adapt the design to the platform constraints (*e.g.*, for emulation, FPGA, and $\mathrm{N}-1$ models). Thus, crucial hardware behavior is typically missed in these co-validation platforms leading to security compromises.

A final point in hardware/software co-validation must be added for the role of formal verification. Formal verification involves use of mathematical analysis to ensure that the system satisfies its desired properties. Typically formal methods incur significant cost, in terms of human expertise or computational resources involved. However, security is critical enough to justify investment of resources in these techniques. But formal verification also becomes inadequate for ensuring security properties involving hardware/software interfaces for the same reason as functional validation using prototyping platforms: scalability of formal verification requires development of abstractions (of both hardware and software components), resulting in crucial security-critical corner-cases being ignored by analysis.

The upshot of the above deficiencies is that security properties involving hardware/software interfaces can be practically validated only during post-silicon validation and debug when the silicon implementation of the current-generation system is ready in some form. Indeed, such validation is only effective in *later stages of post-silicon validation* when both the hardware and software components are mature. However, errors found so late are difficult to fix, and typically result in patches and point-fixes which may expose the system to yet more security vulnerabilities.

V. Specification Challenges

A third challenge in developing security assurance for modern computing devices is the complexity and continuous evolution of requirements and specifications along the system life-cycle. Ideally, specifications ought to be available at the beginning of the design phase of the system and guide design, implementation, and validation activities. Unfortunately, this is rarely the case for a number of reasons. Below we summarize some key issues that cause churn in specification definitions.

Changing Customer and Usage Requirements. The system life-cycle from exploration to production involves a long period of time. Even for a mobile or IoT device built on aggressive schedule, the design and development process takes more than a year. Product requirements can change within this period of time. For example, the launch of a competitor product may result in additional feature requirements to justify additional value to the developing system. Furthermore, a device is typically designed with anticipation of launch and successful business of other components in the overall ecosystem. For example, a smartphone design optimized for a specific operating system "bets" on success of smartphones running that operating system. If the launch of the operating system is delayed, or it is unsuccessful in the market then the specification of the smartphone must change to optimize for other, more successful operating systems.

Discovery of New Security Vulnerabilities. Although security requirements for a target device is defined as part of risk assessment during the design exploration phase, such analysis is based on known security exploits at that time. New vulnerabilities may be discovered, potentially through successful on-field attacks in related devices. Consequently, the security requirements and protection mechanisms need to be updated to account for such threats.

Late Identification of Hardware Bugs. Hardware bugs found late in design and during post-silicon validation are typically patched by firmware and software updates. This is because modifying hardware is more expensive than developing a software patch, *e.g.*, a bug found in silicon, if fixed in hardware, would require an additional silicon spin. Such patches then become part of the system functionality and carried over from one product to the next. This means that the specification of hardware (and in fact, software) gets changed: some functionality originally specified to be implemented in hardware is later adapted to be implemented through a combination of hardware, firmware, and software. Such distribution of functionality, however, hardly get reflected in the specification definition which consequently becomes obsolete.

Such churns in specification, however, have profound effect on the design, implementation, and validation activities. In particular, validation planning typically starts concurrently with design of hardware and software components using the specification documents as guide. Consequently, such plans need to be updated when the specification changes. However, it is infeasible to perform detailed validation planning in response to each requirement change, particularly in later stages of the system life-cycle. To exacerbate the problem, many of the specification changes are not even propagated to the specification documents, making the latter obsolete at the later stages of design development.

VI. Upcoming Security Challenges in the IoT Regime

The challenges discussed in the preceding section pertain to SoC design development and validation, and apply to modern embedded, mobile, as well as IoT systems. IoT systems, however, come with some unique challenges of their own, consequent of their form factor, distributed nature, and applications. In this section we mention a few of those challenges. An IoT system involves a large number of small, smart devices, also called *edge devices* that communicate with each other and to a "cloud" of servers and datacenters through a communication network involving routers and gateways. In its simplest form, the idea is for the edge devices to obtain sensory data from the environment (*e.g.*, ambient temperature for a thermostat, heart rate for a fitness tracker, etc.) which are accumulated, consolidated, and passed to the datacenters in the cloud to perform analysis (*e.g.*, determining whether the temperature ought to be increased or decreased for a home climate system, or an analysis of fitness and health for a health monitoring system, etc.). The cloud is responsible for performing the global analytics and the results are passed along to the edge devices as directives to perform appropriate action (*e.g.*, reduce the temperature if the analytics suggests it ought to be cooler for that time of day at that season of the year). The security challenge in this context arises from the fact that the collection of so much of often highly personalized or intimate information makes the IoT systems susceptible to security threats. Unfortunately, the nature of the threat is still not sufficiently understood (see below). Nevertheless, based on our experiences with mobile systems as well as the understanding of the basic usages of IoT, following are some of the baseline challenges that must be addressed.

Long, Complex Life-Cycle. IoT systems have a long life, compared to the life of a traditional desktop, laptop, or mobile system. For example, a smart home, car, or multiplex can have a life-time of 20-30 years or more. Security requirements, designs, and attacks can change considerably within this time-frame. It is therefore critical for security to be designed with on-field configurability and facilities to upgrade design, implementation, or even the complete specification. As an example of the drastic requirements this might introduce, note that it is possible for quantum computing to be introduced in the years 2030-2050, which will break many of the cryptographic algorithms used in current computing devices. Consequently, one must design IoT systems to ensure that any cryptographic implementation in IoT devices must either be resistant to attacks by adversaries having access to quantum computing or be upgradable on-field.

Power, Performance, an Form-factor Constraints. IoT edge devices perform under aggressive constraints on power; additional constraints may be introduced by unique form factor. For example, a smart implant or wearable must operate with extremely low energy budget. Many traditional security primitives cannot be used in IoT edge devices because of these constraints, *e.g.*, it is often not possible to insert complex security monitors or encryption primitives in hardware. Alternative approaches must be devised to ensure security in the face of these constraints.

Unanticipated Usages and Attacks. IoT systems are connecting ”things” that were never originally intended to be connected, *e.g.*, light bulbs, refrigerators, etc. Consequently it is difficult to imagine potential attacks in these interactions. For example, it is difficult to determine what a potential attack can be in a system where the refrigerator can communicate with the car to direct the grocery items to be collected. IoT designs must therefore be flexible to develop mitigation and protection mechanisms for grossly unanticipated attacks.

Note that the above challenges are not comprehensive. In fact, we do not know all the potential challenges to IoT as we move into the regime. However, they give a flavor of the kind of challenges that we must address in order for IoTs to become robust and trustworthy.

VII. Conclusion

In this paper, we discussed some security challenges in computing devices as we move into the regime of Internet-of-Things. Some of them are reincarnations of old challenges, *e.g.*, hardware/software validation, interoperability of functionality with security, etc. Some are unique and new. Nevertheless, all of these challenges (and many more) must be taken into account in order to make the IoT regime trustworthy. Security flaws in IoT systems can have catastrophic consequences, precisely because the scale and complexity of these systems. Unfortunately, as we discussed through several examples, we do not have an effective solution or even a good understanding of the nature of different challenges. On the other hand, such an environment provides an excellent opportunity for impactful, collaborative research. We believe that a robust security solution for IoT regime will be achievable only through strong collaboration among researchers across different disciplines, potentially breaking many of the topic silos we currently have in computing science.

References

[1] D. Evans, “The internet of things - how the next evolution of the internet is changing everything,” *White Paper. Cisco Internet Business Solutions Group (IBSG)*, 2011.

[2] J. Goguen and J. Meseguer, “Security Policies and Security Models,” in *Proc. 1982 IEEE Symposium on Security and Privacy*, 1982, pp. 11–20.

[3] A. Basak, S. Bhunia, and S. Ray, “A Flexible Architecture for Systematic Implementation of SoC Security Policies,” in *Proceedings of the 34th International Conference on Computer-Aided Design*, 2015.

[4] S. J. Greenwald, “Discussion Topic: What is the Old Security Paradigm,” in *Workshop on New Security Paradigms*, 1998, pp. 107–118.

[5] S. Ray, J. Yang, A. Basak, and S. Bhunia, “Correctness and Security at Odds: Post-silicon Validation of Modern SoC Designs,” in *Proceedings of the 52nd Annual Design Automation Conference*, 2015.

[6] S. Heath, *Embedded Systems Design*. EDN Series for Design Engineers, 2003.

[7] “Embedded Systems,” in *PC Magazine*, 2012.

[8] L. Lei, F. Xie, and K. Cong, “Post-silicon Conformance Checking with Virtual Prototypes,” in *DAC 2013*, 2013.