cadence®

A Complete System-Level Security Verification Methodology

By Dr. Nicole Fern, Senior Hardware Security Engineer, Tortuga Logic; Steve Carlson, Director, Aerospace and Solutions Architect, Cadence

Hardware is at the root of all digital systems, and security must be considered during the system-on-chip (SoC) design and verification process. Verifying the security of an SoC design is challenging because of time to market pressure and resource constraints. Resources allocated to the already time-consuming task of functional verification must be diverted to security verification, which requires a significant shift in strategy because security vulnerabilities often exploit unintended or unspecified functionality. Because of these challenges, there is currently no systematic, scalable, and effective methodology for pre-silicon security verification. Tortuga Logic and Cadence have collaborated to provide a security verification platform capable of detecting system-level vulnerabilities requiring minimal disruption to existing functional flows.

Contents

Introduction: Security Essential Across Market Verticals

Computer systems and electronic infrastructure must be resilient in the face of cyberattacks. These systems control many critical aspects of our daily lives ranging from the cars we drive to banking infrastructure. Hardware is at the root of the trust chain across all market verticals, as any digital system runs on silicon. For most applications, security efforts focus mainly on software, with the assumption that hardware is inherently trustworthy and does not open the system to additional vulnerabilities. Recently, this assumption has been challenged by several high-profile system-level exploits rooted in hardware security deficiencies affecting aerospace and defense, automotive, datacenter, and IoT verticals. In all these domains, there have been initiatives to address security; however, designing and verifying secure systems remains a challenging task.

Aerospace and defense

Multiple different Department of Defense (DoD) initiatives striving to develop a portfolio of microelectronics protections and design requirements have been put in place. For example, the DoD Microelectronics Innovation for National Security and Economic Competitiveness (MINSEC) initiative has allocated $2 billion to advance the United States' competitive advantage in microelectronics with a significant emphasis on hardware security.[1] This and other initiatives emphasize the importance of ensuring security and trustworthiness of microelectronics used in military and aerospace applications; however, many challenges remain, such as verifying the security of systems incorporating untrusted third-party IP. Addressing these challenges requires new technologies to detect and prevent security vulnerabilities.

Automotive

Every function in all automobiles made within the last few years is governed by several processors, including the braking system, steering, and cruise control. With increasing wireless connectivity, which has been shown to provide remote attackers entry into the in-vehicle network[2], trusting these processors to perform correctly and be resilient against attacks is critical. In 2015, legislation was introduced in the US Senate ("SPY Car Act of 2015")[3], compelling the NHTSA and FTC to set security standards for cars. An automotive cybersecurity standard, ISO/SAE 21434, is currently under development to address the unique cybersecurity challenges in the automotive domain[4]. Verifying proper isolation of internet-connected functionality, such as infotainment, from safety-critical operations, such as steering and braking, is difficult with today's hardware verification strategies, as evidenced by the recurring security breaches affecting many automobile manufacturers.

Datacenter

Datacenter hardware is optimized for performance, is becoming more heterogenous, and is increasing in complexity. These are all significant factors in the growing hardware security concerns. Datacenter hardware is shared between many different customers, leading to concerns about isolation between applications. Traditionally, the operating system provided acceptable guarantees about process isolation. Recent attacks, such as Spectre and Meltdown[5], Foreshadow[6], and Spoiler[7], however, demonstrate that a bug-free OS and bug-free software utilizing advanced hardware security features, such as secure enclaves, can still be completely compromised. These vulnerabilities, stemming from unexpected side-effects of processor performance features, lie at the hardware/software interface, requiring a shift in verification strategy to detect.

Additionally, the heterogeneity of datacenter hardware is introducing huge concerns about the trust of hardware components and their firmware. One such example is an attack capable of completely replacing firmware on a server baseboard management controller (BMC), which has privileged access to all components on the server motherboard, with malware[8]. The attack is possible because the *hardware configuration* of the communication bus connecting the BMC to the host machine does not authenticate transactions originating from the host machine. There are multiple initiatives, such as Microsoft's Project Cerberus[9] and Google's OpenTitan[10], which focus on building more standardization around the security of server hardware and firmware infrastructure.

IoT

IoT devices continue to grow in popularity, despite numerous security incidents, such as the Mirai botnet[11], BleedingBit[12], and Thangrycat[13], which have demonstrated that the intense time-to-market pressure IoT providers face results in prioritizing adding marketable features and lower product cost over security. What is clear is that IoT vendors need a low-impact, easy-to-deploy, and effective security verification strategy to make sure these devices are secure.

Challenges in Building Secure Systems

There are numerous challenges in building secure systems. These include supply chain concerns, reverse engineering and counterfeiting, and physical tampering and side-channel attacks.

This whitepaper focuses on the challenge faced earliest in the design lifecycle: ensuring the hardware design sent to the fabrication facility is free of functional bugs and security vulnerabilities.

Many designs include *security* features, but it is important to ensure that these are also secure features. Security verification is difficult because of the fundamental asymmetry between attackers and defenders. An attacker only has to discover and exploit a single vulnerability to achieve their goals, whereas the system design and verification teams must anticipate and defend against the complete set of possible threats. Many security vulnerabilities hide within modes of the design not exercised by typical system usage and go undetected during traditional functional verification. Successful security verification requires a shift in thinking. Instead of focusing solely on whether the design functions properly, the verification strategy needs to be centered around two questions:

- What information in my design needs to be protected?
- Where does that information flow and how is that information accessed?

Current methodologies being deployed to address security are simulation/emulation-based verification, manual design review, formal verification methods, and penetration testing. Applying one or all these methods is still insufficient to address hardware security. Moreover, these existing approaches are time-consuming, hard to measure, and are often a burden to engineering schedules. These approaches are at odds with the existing verification and development process, in which engineering teams are focused on meeting schedules, while product security teams are focused on building out unique and differentiated security products. This hurdle must be overcome to get a secure product to market without introducing added costs or delaying products schedules.

There are several existing challenges that need to be addressed to effectively detect and prevent hardware vulnerabilities.

Challenge #1: Security policy capture

Security objectives are based around confidentiality, integrity, and the availability of design assets. These objectives are difficult to formulate using existing tools which have no concept of *information flow*. Writing a directed SystemVerilog test or assertion to verify the confidentiality of an encryption key is extremely time-consuming and difficult to compose. It is possible to record the value of the key as it enters the encryption block, then check if that exact same value appears at the output; but the key can go through an infinite number of simple transformations (for example, exclusive-or with plaintext, bit shift, etc.) from which an attacker who can observe the output can easily recover the original key value. The verification infrastructure knows the expected ciphertext value; however, other transformations that correspond to leakage of intermediate states are difficult to enumerate and check for using directed tests and SystemVerilog assertions. More complex indirect leakages through timing side-channels are impossible to detect using value-based checkers. Similar difficulties are encountered when attempting to capture integrity and availability properties in the verification infrastructure.

Currently, no existing tools used in simulation/emulation-based functional verification can track information flows in the design. Labor-intensive negative tests must be developed to check security-specific corner cases for the highest priority threats, but without the ability to track information flow in the design, negative testing is extremely limited.

Code reviews and penetration testing involve reasoning about information flows, but because these are manual processes, they require an enormous amount of effort, which does not scale as the design size and complexity increases. Some formal verification tools provide the capability to verify information flow properties, but these tools face scalability issues and are unable to easily describe hardware flows while system software is executing.

Challenge #2: Security analysis must scale to SoC level

Security analysis must scale to system-on-chip (SoC) level to detect vulnerabilities arising from the configuration and integration of security IP into the larger system. For example, leakage of sensitive information can occur if the topology of an on-chip interconnect is configured incorrectly, or if there are errors in connecting security-critical signals in the bus interface, such as privilege bits.

Formal verification tools and manual code reviews face scalability issues because of the exponential increase in design complexity resulting from interactions between different functional blocks once they are integrated together. Penetration testing is typically done post-silicon, and any integration errors detected at this time may be difficult to remedy with firmware or software patches alone.

Among existing verification methodologies, simulation and emulation-based testing are the only techniques which scale to SoC-level analysis and are used as the primary verification methodology of the semiconductor industry. Every digital chip designed today has undergone a significant amount of simulation-based verification and before tapeout. Emulation is also widely employed to perform SoC-level testing capable of analyzing firmware and the entire device boot into a commodity OS, such as Linux. The increased capacity that emulation provides is critical for testing low-level software and firmware that will ship with the chip. With respect to security, the main weakness of simulation- and emulation-based verification is that security vulnerabilities often are unknowns, unrelated to core design functionality. Due to modern design complexity, it is impossible to exhaustively test the entire design during simulation or emulation; therefore, some security vulnerabilities are likely to go undiscovered.

Challenge #3: Hardware and software must be analyzed together

To provide flexibility, hardware is designed to be configurable by software and firmware. Security functionality is no exception. There are many hardware security features that are software-configurable either at boot time or dynamically during system execution. Consider a memory protection unit (MPU), which implements access control policies for

different regions of memory. A specific platform can choose to use the MPU to partition the memory space into static regions, dynamically change the regions and permissions, or not use the MPU at all. All choices are valid from the perspective of pure hardware verification, but when considering verification of the entire system, software choices regarding the programming of hardware features have a significant impact on security.

It is extremely important to analyze hardware behavior while the software is executing. Software misconfigurations leading to security vulnerabilities cannot be detected by analyzing the software without a model of the hardware, and analyzing only the hardware is insufficient to catch hardware usage errors as what constitutes a "misuse" of features is context-dependent.

Penetration testing is performed on the entire system, which includes the firmware and software stack. Engineers on the "red team" adopt the role of a potential attacker and perform penetration testing on the product to highlight areas of the system susceptible to real-world attacks; engineers on the "blue team" harden the design against possible red-team attacks. Penetration testing is an effective way to verify specific security-critical hardware/software flows, such as secure boot; however, it is a manual process and is typically done post-silicon to better mimic the conditions under which an attacker will attempt to infiltrate the system.

Visibility into the hardware is extremely limited, and the red team must develop a sophisticated-enough exploit to cause observable problems in the system, such as a crash. A pre-silicon security verification methodology that uses emulation platforms can still run a significant amount of system software but retains full visibility into the hardware design, increasing the number of potential vulnerabilities detected for the same amount of manual effort needed to write software penetration tests.

Formal verification methods explore the complete input and state space of the design when checking for security policy violations, making these techniques an important part of the overall security verification strategy. Formal methods are more complete than simulation- or emulation-based verification, but if the hardware is intended to have multiple configurations controlled by software, where some of the configurations are expected to violate the security policy and others do not, it often becomes prohibitively complex to constrain the formal model to the relevant scenarios. Moreover, even if constraining the model is feasible, the analysis results do not guarantee that the actual software shipped with the system, for example, boot code, correctly programs the silicon to behave as originally expected.

Simulation- and emulation-based verification does not suffer the same plight because security rules can be verified, while the boot code and firmware to be shipped with the system configure the hardware. Any rule violations occurring will be true vulnerabilities, not false positives, and can be traced back to specific points in the software execution. Capacity is also a limitation of formal tools, and for security rules spanning large portions of design functionality, simulation or emulation is often the only option.

Challenge #4: Security verification must have low overhead

Security verification must not add significant overhead because functional verification is already a bottleneck in the silicon design process. Verification resources include engineer hours spent developing test infrastructure and debugging results, and computational resources to run simulations or formal tools. While the process of identifying security objectives and designing and implementing a security architecture cannot be automated, there is tremendous opportunity to leverage the existing effort expended on the functional verification task for security verification. Simulation and emulation are already used heavily in the hardware design flow and scale to system-level hardware/software analysis and are therefore ideal starting points for developing a scalable and effective security verification methodology.

A Scalable and Effective Security Verification Methodology

The challenges outlined in the previous section motivate the following requirements for a security verification methodology:

- Intuitive and efficient mechanisms for security policy specification
- A unified toolset for verifying functional and security properties throughout the design lifecycle capable of scaling to system-level simulation/emulation to perform hardware/software security verification
- An environment that captures the progress of both the functional and security verification effort

To address these challenges, Tortuga Logic and Cadence have collaborated to provide a security verification platform that meets the requirements previously mentioned. To address security policy specification, the platform leverages Tortuga Logic's Sentinel™ language, which enables security rules centered around the concepts of confidentiality, integrity,

and availability to be expressed in a straightforward manner. These rules can be verified using the same infrastructure employed for pre-silicon functional verification. A comprehensive set of Cadence® products exist for functional verification throughout the design lifecycle, and when used in concert with Tortuga Logic's Radix-S and Radix-M security verification software, the functional verification environment can be extended to perform security verification.

Radix-S and Radix-M are two security verification platforms that integrate with functional verification tools to identify security vulnerabilities. Both employ patented information flow tracking technology exposed to the user using the Sentinel language to verify security policies based around confidentiality, integrity, and availability. Radix-S is a software package that scans a system's hardware and software during the pre-silicon design and verification simulation stages to identify and detect vulnerabilities. Radix-M, the second product in Tortuga Logic's Radix series, performs firmware security validation on full SoC designs by leveraging Cadence Palladium Z1™ Enterprise Emulation Platform.

Figure 1 gives an overview of this security verification flow, where blue elements and arrows denote industry-standard components and processes necessary for pre-silicon functional verification, and gray illustrates how security verification using Radix fits into this existing flow. Leveraging the existing functional verification environment for security verification is essential to making security verification feasible under aggressive project schedules.

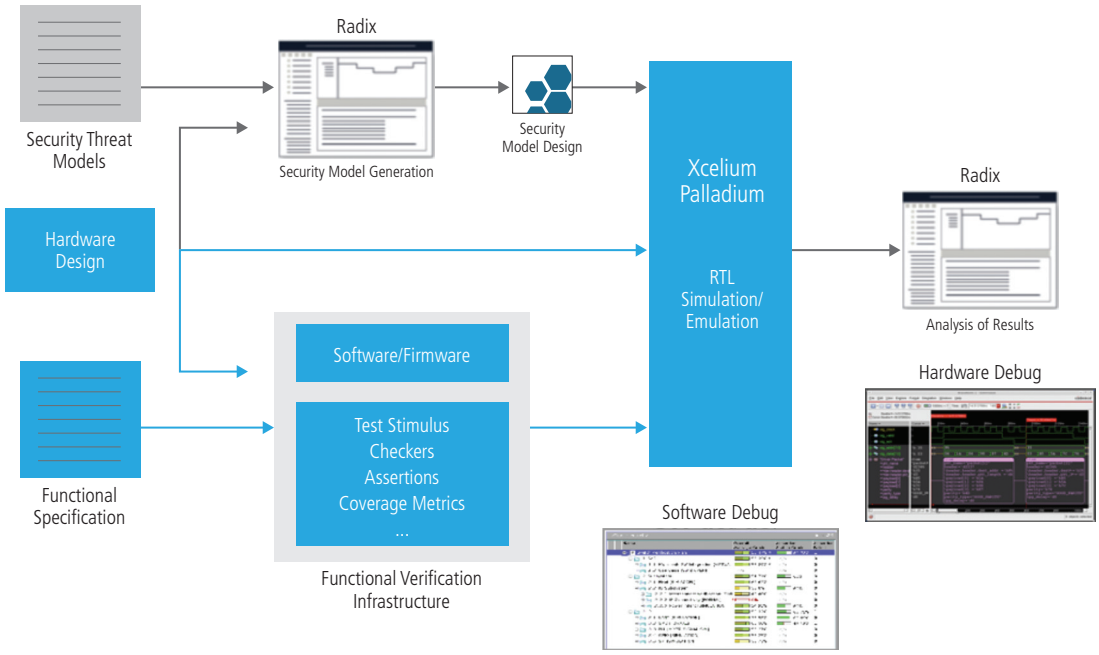

Figure 1: Tortuga Logic and Cadence flow for security verification

Security policy capture

Threat modeling is the process of identifying important data and design states, called assets, which must be protected from an attacker. Assets include, for example, cryptographic key material, registers storing the current execution privilege mode, firmware and boot ROM code, memory access control settings, debug interface, etc. Part of the threat modeling process is reasoning about the environment the system will be deployed in, who is motivated to attack the system, what their capabilities are, and what assets these attackers are interested in. This will guide the process of security specification, which, ultimately, is expressed as a series of requirements for the hardware design which must hold to protect design assets from adversaries.

Successful security policy specification and verification requires a shift in thinking. Instead of focusing solely on whether the design functions properly, the verification strategy must be centered around two questions:

- What information in my design needs to be protected?
- Where does that information flow and how is that information accessed?

Security threat models are expressed in terms of Tortuga Logic's Sentinel security verification rules. Radix uses the design RTL along with the security rules to generate a security model (Verilog) that easily integrates into standard functional verification environments without disruption to existing workflows. The security model is used to check the Sentinel security rules during simulation and emulation and is used only during security verification analysis. The security model does not become part of the design implementation nor impacts design area and performance.

The main difference between Sentinel rules and functional properties/assertions is that Sentinel rules easily capture security policies centered around the security concepts of confidentiality, integrity, and availability due to the Security Model's unique ability to track *information flows* in the hardware design. These rules can be specified at any stage in the design lifecycle and will evolve over time but can be re-used as the verification scope increases from block to sub-system to the entire SoC.

Tortuga Logic tools provide a simple syntax for expressing information flow rules and our security model contains patented technology to detect violations of these rules during RTL simulation and emulation. The basic structure of a Sentinel rule is the following:

source =/=> destination

A Sentinel rule always consists of a source, the "no-flow" operator, =/=>, and a destination. The source and destination can be a single signal, or a set consisting of multiple signals. The rule will fail if information from any signal in the source signal set flows to any signal in the destination signal set. In addition to the no-flow operator, several keywords are provided to increase the expressiveness of the Sentinel language. For example, the "when" keyword can be used to specify conditions that must be held for information flows from the source to be tracked.

System-level security verification using Tortuga Logic's Radix-M and Cadence's Palladium Z1 Enterprise Emulation Platform

Cadence provides a unified simulation and emulation environment ideal for efficient scaling to system-level functional and security verification. Simulation and emulation differ in the size of the design analyzed and the functionality covered during testing. Simulation primarily focuses on verifying hardware functionality at the block and subsystem level and will include comprehensive directed and constrained random tests and checkers for all available hardware features. A few tests may be run during simulation on the entire SoC to check for integration errors but running large portions of the software stack, such as a secure boot flow, is often too time-consuming to run and debug in simulation.

The main use case of emulation is to verify the low-level software stack. Low-level software, such as boot code, firmware, and device drivers, are responsible for configuring and interfacing with hardware and are commonly provided by the chip vendor. It is important to identify bugs in the software stack early on, but even more critical to uncover bugs in the hardware that manifest only under the complex sequences of software operations covered during emulation.

For security verification, the scope of security rules can vary from a single block (e.g., an encryption key should not exit the boundary of the encryption module) to spanning the entire SoC (e.g., test and debug protections). However, even for block-level rules, it is important to verify that the rule still holds while running the software to be shipped with the system if any of the block configuration or operation is controlled or managed by software. Additionally, security-critical functionality, such as the boot flow, must be verified using the actual device boot code.

Tortuga Logic's Radix-S integrates seamlessly with the Cadence Xcelium™ Parallel Logic Simulator and Radix-M supports the Cadence Palladium® Z1 Enterprise Emulation Platform, providing a comprehensive security verification solution. Security objectives expressed using Sentinel can be verified using both existing hardware-centric tests from the simulation infrastructure and software tests. This enables software-configurable hardware security features to be vetted using the exact configuration programmed by the software shipped with the product.

Another important advantage of incorporating Radix into both simulation and emulation flows is that the security rules can be used as a common language for security verification objectives between the security team and the teams involved in simulation and emulation-based verification and low-level software development. This distributes the security verification task more effectively and prevents disconnects between the various teams leading to exploitable vulnerabilities going unnoticed.

Radix contains an interactive platform to aid in the visualization of information flow throughout the design, as seen in Figure 2. This platform includes a waveform viewer that annotates the simulation trace with data about information flows under specific security rules. If any rules fail during simulation, Radix also provides a visualization of a concrete path through the design hierarchy showing information flow, causing a security rule violation. These analysis features make exploring information flow in the design more efficient and are unique to Radix.

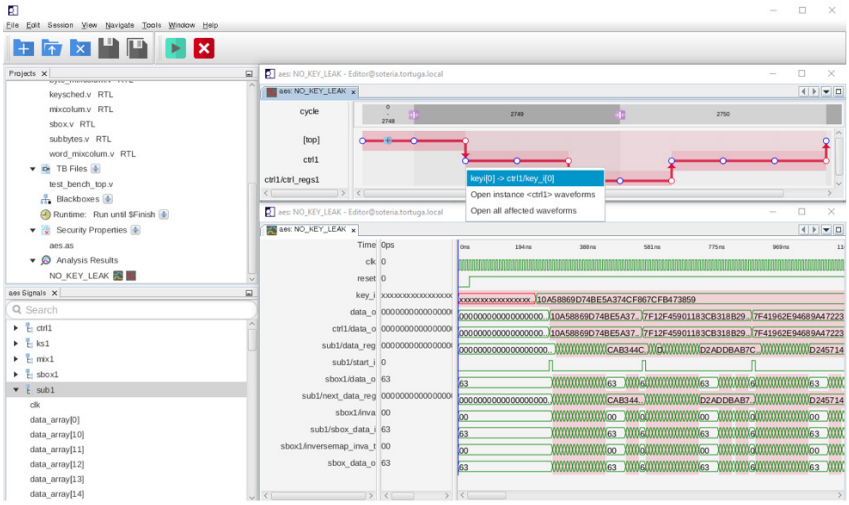

Figure 2: Radix analysis views enabling efficient analysis of security-relevant information flows within the design

Test stimulus for security rule verification and metrics

Prior sections have shown that Sentinel rules can be incorporated seamlessly into the existing verification infrastructure, meaning all manual effort expended developing tests achieving high functional coverage can be re-used for security rule verification. For simulation- and emulation-based security verification, an important question is:

- How effective are existing functional tests at uncovering potential security rule violations?

Security verification methodologies capable of addressing the complexities of system-level hardware/software analysis are rooted in commercial best practices for functional verification. Metrics-driven verification (MDV) is the process of creating an infrastructure in which progress is continuously tracked throughout various stages in the design lifecycle against verification goals.

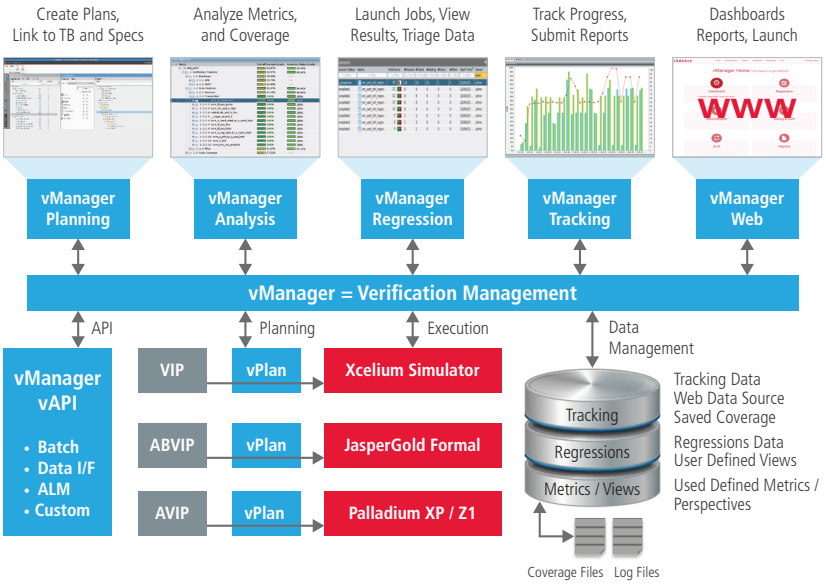

Figure 3: Cadence vManager platform

A key component of metrics-driven verification is coverage metrics. Metrics are an important component for gauging the progress, completeness, and effectiveness of the verification and validation effort. Good metrics provide quantifiable feedback on how well the test stimulus exercises the design, and correlate with the probability of detecting bugs. Achieving higher coverage means greater confidence in design correctness.

In functional verification, the desired modes to be checked are known, whereas events leading to violation of security policy are often unknown and may involve complex interactions between different functional components, making the development of security metrics difficult.

Currently, there are no widely adopted metrics for security verification. Just as good line/statement coverage is a bare minimum for traditional verification, very high functional coverage is a bare minimum for security verification. We have seen in practice that a comprehensive suite of functional tests, when combined with security-centered negative tests, intelligent fuzzing, and constrained random testing, already activate undesired information flows in the design. The limiting factor in the detection of these flows is the complexity of developing value-based checkers with the ability to detect transformations of the source signal and indirect information flows.

Use of Tortuga Logic Sentinel rules alleviates the need for complex checkers, and a set of security rules verified using Radix and a mature functional test suite always results in significantly improved security coverage due to the unique ability of Radix to detect direct and indirect information flows.

Case Study: Identifying leakage of cryptographic key material using customer's existing functional test suite

Many designs contain hardware blocks which accelerate cryptographic algorithms. While an algorithm may be secure in theory, the implementation, which includes the hardware and the software controlling the operation, can result in the leakage of sensitive information.

A Tortuga Logic customer design contained a symmetric cipher block that could decrypt either an encrypted key or encrypted data. The decrypted plaintext data can leave the block, but the key may not appear at block outputs in any form. A block diagram similar to the customer's architecture is shown in Figure 4 and illustrates the expected and illegal flows.

The addition of the key decryption operational mode added a security feature but resulted in more complex control logic, because there are various multiplexers and routing logic necessary to select from the various key and data sources, and either save the output of the decryption block to the key storage or export it through the "plaintext" output. Increased complexity often results in implementation errors that can lead to security vulnerabilities.

The illegal flow present in the customer's logic is shown in red in Figure 4. This flow resulted from a quirk in the implementation of the routing logic that packages the output of the cryptographic core into bus transactions. The flows manifested under a specific sequence of software operations that occurred several times during the directed functional tests provided by the customer for this block.

The illegal flow was not detected by the customer, despite comprehensive functional testing of this cryptographic block, under which the flow did exist for a handful of regression tests.

One reason the flow went undetected is that the flow added extra unintended functionality and did not impact the functional operation of the decryption core. In the block diagram, the expected flow and usage of an encrypted key are shown in green. The vulnerability adds an extra illegal flow, shown as a red arrow, but does not suppress the expected flow and is not detected unless a special test were written with this specific vulnerability in mind.

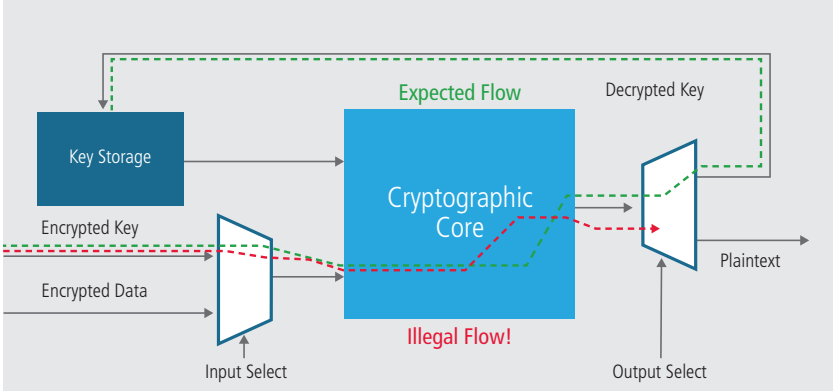

Figure 4: Illegal key flow in customer design detected by Radix

Detecting this illegal flow using Radix only required writing a single simple security rule checking if the Encrypted Key flows outside the module boundary:

Encrypted Key =/=> Plaintext

Without additional tests targeting key leakage vulnerabilities, this Sentinel security rule was able to detect the illegal flow early on during a 4-week services engagement with the customer. This case study illustrates the power of combining Radix with pre-existing functional verification environments.

Radix was able to quickly identify a vulnerability in part because the customer had a comprehensive set of regression tests for the design block being analyzed. Because Radix is a simulation/emulation-based technique, the quality of the test stimulus impacts the effectiveness of Radix in identifying security rule violations, making it critical to employ best practices for functional verification.

Conclusion

Hardware is becoming more complex, customized, and ubiquitous. This is driving security features into hardware and creating more attack vectors that exploit hardware vulnerabilities. The existence and exploitation of these hardware vulnerabilities can increase the time to market, reduce chip vendor trust, and lead to costly lawsuits and chip recalls. Ensuring hardware designs and the associated software that configures and uses hardware features are free from vulnerabilities before tapeout is a challenging task. An effective security verification methodology must have low overhead in terms of manual effort and compute resources, but also scale to SoC-level analysis and be able to address both hardware and software together.

In this whitepaper, we have outlined a methodology for system-level security verification addressing these unique challenges based around Tortuga Logic and Cadence products. Tortuga Logic's Sentinel language provides an efficient mechanism to express security policy centered around confidentiality, integrity, and availability of design assets. Tortuga Logic's Radix in combination with Cadence's RTL simulation and emulation platforms enable efficient system-level security verification by leveraging the existing verification environment to detect violations of security rules.

References

1. http://www.nationaldefensemagazine.org/articles/2018/6/14/official-pentagon-investing-billions-into-microelectronics
2. https://www.sandiegouniontribune.com/news/education/sdut-ucsd-professor-cyber-hacking-2015aug28- story.html
3. https://www.congress.gov/bill/114th-congress/senate-bill/1806/all-info
4. https://www.sans.org/cyber-security-summit/archives/file/summit-archive-1525889601.pdf
5. https://meltdownattack.com/
6. https://foreshadowattack.eu/
7. https://arxiv.org/abs/1903.00446/
8. https://www.theregister.co.uk/2019/01/24/bmc_pantsdown_bug/
9. https://azure.microsoft.com/en-us/blog/microsofts-project-olympus-delivers-cloud-hardware-innovation-at-scale/
10. https://cloud.google.com/blog/products/gcp/titan-in-depth-security-in-plaintext
11. Antonakakis, Manos, et al. "Understanding the Mirai Botnet." USENIX Security Symposium. 2017.
12. https://go.armis.com/bleedingbit/
13. https://thrangrycat.com/