Abstract—The hardware-software boundary within SoC-based integrated systems has been proven to be vulnerable to attacks because neither hardware nor software defense methods can, by themselves, secure the whole system. A highly secure SoC architecture is proposed for holistic system defense based on the concept of a “hardware anchor”, which bridges the communication between OS and hardware. The anchor helps an OS kernel to actively monitor its extensions and can also track bus traffic within the SoC platform to prevent untrusted third-party IP modules from performing malicious operations. Simulation results with an x86 emulator (Bochs) and Linux OS demonstrate the effectiveness of the proposed architecture with low performance overhead.

I. INTRODUCTION

The problem of malicious modifications to software and hardware infrastructure threatens the security of computer systems, especially those built upon SoC platforms and firmware extensions. Various security methods have been proposed to ensure the trustworthiness of different layers within computer systems, but they suffer from two main drawbacks. First, hardware and software currently do not safely co-exist with their needed, but untrustworthy, extensions. Because of pressure to shorten time-to-market and increase the reusability of the designed systems, SoC designers/integrators rely heavily on third-party resources. Under the current SoC-based system design flow, only the kernel firmware and the core processor/microprocessor that are designed in-house go through full functionality and security testing and, thus, are trusted. Peripheral IP modules and firmware extensions are often not fully tested to save cost. Even worse, it is often the case that a thorough testing of third-party IP modules and firmware extensions is impossible because hardware modules or software programs are provided as black boxes.

For OSes, extensions make up a large portion of the kernel code base (approximately 70% in Linux [1]). These extensions are convenient for extending the kernel functionality and allowing a system to communicate with I/O devices without the need to reboot the system or recompile the kernel. In spite of that, these extensions are not trustworthy because they are developed by third-party companies.

Most of these hardware and software extensions are benign and beneficial to the host system, but some can be malicious or vulnerable to attacks. This paradoxical situation can threaten system trustworthiness. In hardware, third-parties IP cores can contain malicious logic (e.g., a Trojan or backdoor). At the OS layer, malicious extensions (rootkits) represent a dangerous avenue for system exploitation as they run in kernel mode.

The second drawback of current hardware and software security mechanisms is that they are not integrated. Current solutions are layer-specific, make wrong assumptions about the trustworthiness of other layers, and have players that do not cooperate and communicate with peers at other layers. Hardware security approaches rarely leverage system level context and work independently from what is being accomplished at the software layer. That is, most of the hardware security methods try to detect and prevent the insertion of hardware Trojans in a stand-alone way and assume that firmware and OS are trustworthy [2]. As a result, these security methods suffer from the limitation of scalability and are unable to be applied to large-scale circuits such as SoC platforms.

Security approaches employed at the OS or virtual machine (VM) layers do not cooperate with the hardware, nor do they share with it intelligence about attacks. Also, current VM-based security solutions [3], [4] do not rely on any collaboration with the guest OS (considered easily compromisable) and assume a trustworthy hardware.

In order to increase the robustness of computer systems’ defense mechanisms, especially systems constructed on an SoC platform, we are currently developing a cross-boundary holistic security defense architecture that relies on cooperation between a trusted OS and processor within the computer system. The communication channel and cooperation protocol is based on a hardware anchor, which can dynamically monitor whole-system operations and enforce both dynamic hardware and software security policies.

II. SW/HW CO-PROTECTION

The main component of the proposed platform is the hardware anchor, which provides a communication channel between the OS and hardware for SW-HW cooperation. Through the anchor the OS provides the hardware with data (or intelligence) on key system-level events and also enforceable security policies. The anchor is also responsible for processing OS-level data and policies and enforcing applied security policies.

The hardware anchor embedded inside the SoC platform also gathers intelligence about architecture-level events by monitoring the data passing through the bus. It enforces hardware-level security policies to maintain a safe-coexistence of the trusted processor and untrustworthy third-party IP cores. The anchor is composed of two parts. The first component is a cross-layered interface with the OS represented by a new software interrupt by which the OS can call the processor’s attention and pass data and policies to the anchor. The second component is a hardware module responsible for processing OS and architecture-level data and enforcing applied security policies.
The cross-layered interface is implemented by modifying the processor core so that the processor can recognize and deal with this new software interrupt. The OS leverages this new instruction to pass to the anchor information about system-level events, such as module installation/uninstallation, creation of processes and files, function addresses and types (exported and non-exported by the kernel), allocation and deallocation of data structures and dynamic memory. For example, the OS downcalls the anchor to provide memory boundaries of extensions whenever they are installed in the system. The modified processor will then record this information and impose on all extension’s operations at hardware level that could violate the applied security policies. The anchor will check whether or not these operations (e.g., a load/store), violate the applied security policies. Examples of OS-level policies are: (i) Modules cannot write into kernel code segment, (ii) Modules cannot write into kernel data segment, except to their own address spaces, (iii) Modules can only invoke kernel exported functions, (iv) Memory range \((x, y)\) cannot be read by modules, (v) Module \(X\) can bypass security policy \(Y\), etc. The OS is also equipped with a library of security policies that can be mixed and matched during kernel installation.

The hardware anchor also monitors operations of hardware IP modules attached to the SoC bus guided by security policies developed at the hardware level. The SoC bus arbiter is enhanced to monitor bus read/write operations initiated by 3rd-party IP modules preventing IP modules from visiting restricted data illegally and, thus, enforcing the applied security policies within the SoC platform.

The OS also receives intelligence from the anchor about the status of hardware level security enforcement. For example, the anchor will inform the OS through a new exception that a particular security policy was violated and which entity caused the violation. Even though the anchor prevents policy violation, the OS can also take other security measures such as uninstalling/quarantining offending modules, applying a more restrictive security policy, or even removing a security policy in the event of false positives or performance issues.

III. Experimentation Results [5]

A proof-of-concept prototype for this new paradigm was designed and implemented using Bochs Intel x86 emulator [6] running Linux Ubuntu 10.04 (kernel version 2.6.32) [5]. Experimental evaluation successfully confined modules into their own address spaces and contained their interactions with other parts of kernel code and data. The prototype’s security was assessed with a set of real kernel rootkits which were stopped before any malicious actions were performed and with benign modules that could run normally.

All experiments were executed on an Intel quad core 3.8 GHz with 16 GB RAM running Linux Ubuntu 12.04. Each performance experiment was executed ten times and the results were averaged. Figure 1 shows the performance analysis for five benchmarks from SPEC CPUINT2006. The overhead was measured for two different situations. In the first (OS Downcalls), the system was running the modified version of the OS containing all downcall issuing. Here the goal was to evaluate the overhead to the OS for a CPU intensive benchmark. The second setting (Downcall handling) had the same configuration as the first, but now the downcalls were being processed by the emulator’s handlers. The downcall issuing overhead at the OS is very low (less than 2%). Downcall processing caused a slowdown of 1.12X. The 12% overhead is low when it is considered that certain types of applications that require a high level of security (e.g., a power grid or a server at a security agency) can trade performance for security.

IV. Conclusions

We proposed a software-hardware co-protection architecture which enhances a traditional SoC platform with a hardware anchor that facilitates the communication between OS and hardware. The anchor allows an OS to apply dynamic security policies to be enforced at the hardware for protection against untrustworthy extensions. The hardware anchor also works closely with OS intelligence to regulate operations initiated by third-party IP cores so that any inserted hardware Trojans can be caught online before they can do harm to the whole computer system. Simulation results support our claim that the proposed framework can provide end-users with a highly secure SoC platform-based computer system with low performance overhead. Our future work includes the implementation of the framework SoC platform with SPARC-compatible processors and peripheral modules. We will also study the trade-off between security and performance.

REFERENCES