SHIELDSTRAP: Making Secure Processors Truly Secure

Siddhartha Chhabra and Brian Rogers and Yan Solihin
North Carolina State University
{*schhabr, bmrogers, solihin*}*@ncsu.edu*

***Abstract*— Many systems may have security requirements such as protecting the privacy of data and code stored in the system, ensuring integrity of computations, or preventing the execution of unauthorized code. It is becoming increasingly difficult to ensure such protections as hardware-based attacks, in addition to software attacks, become more widespread and feasible. Many of these attacks target a system during booting before any employed security measures can take effect. In this paper, we propose SHIELDSTRAP, a security architecture capable of booting a system securely in the face of hardware and software attacks targeting the boot phase. SHIELDSTRAP bridges the gap between the vulnerable initialization of the system and the secure steady state execution environment provided by the secure processor. We present an analysis of the security of SHIELDSTRAP against several common boot time attacks. We also show that SHIELDSTRAP requires an on-chip area overhead of only 0.012% and incurs negligible boot time overhead of 0.37 seconds.**

I. Introduction

As computing systems become more distributed, it is increasingly more difficult to guard systems against *unauthorized physical tampering* by attackers. One example is companies that have mobile computers (e.g., laptops, PDAs) which contain sensitive data, and employees are allowed to carry them at will. The laptops may be stolen by attackers, or even tampered with by the employees themselves without the knowledge of the companies. Another example is game consoles in which the owners may try to circumvent the copyright protection mechanisms. In addition, the damage that can result from a successful attack is also increasing: loss of revenues from game sales and theft of valuable company or consumer secrets (credit card numbers, product information, etc.).

Compared to traditional, software-based security attacks, physical tampering is unique in that it can be used to bypass software protections in the system. Even systems that store their files encrypted on the disk still keep the data in plaintext form in main memory. Through physical tampering, attackers can dump the content of the memory, attach a bus snooper between the processor and the main memory in order to snoop plaintext data, or even alter the values stored in the main memory or communicated between the processor and main memory. Some examples of physical attacks that have been documented in detail include mod-chips, which are widely available for popular video game consoles [1], [2], [3], [4], [5], [6], and attacks on processors used for ATM machines [7]. These attacks demonstrate the feasibility and surprising ease of bypassing security protection through physical tampering.

Recognizing both the increased opportunity and motivation for attackers to compromise the *privacy* and *integrity* of data and computation, researchers have proposed *secure processor* architectures [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20]. These architectures assume that the processor die provides a reasonable security boundary. Thus they protect any data stored off-die (including the main memory) with encryption and integrity verification. Unfortunately, these studies describe such mechanisms for steady state execution after the system is up and running. They do not address the threat of physical attacks during system boot. In this paper, we address the problem of how to securely boot a secure processor system.

The goal of secure bootstrapping is to ensure that the system has booted using valid boot components including the basic input output system (BIOS) and operating system (OS). Recognizing the vulnerability of the bootstrapping process to the overall security of systems, prior work has proposed various secure bootstrapping mechanisms, such as AEGIS by Arbaugh et al. [21], Trusted Computing Platform [22], and others [23]. These approaches all follow a *chained integrity verification* approach where each layer in the boot process verifies the integrity of the next layer before passing control to it. However for each approach the root of trust is the BIOS code, which in most computer systems is located off-chip. Hence, they cannot ensure the integrity of the bootstrapping process if physical tampering of the BIOS code is a possibility. Thus, they are inappropriate for implementation on a secure processor.

There have also been proposed solutions which do assume that physical attacks to off-chip components are possible. For example, in ARM TrustZone [24], security extensions to the ARM architecture, the entire boot code is moved on chip, eliminating the possibility of physical tampering to alter the boot code. Additionally, any application that needs to be secured must be kept on-chip in a *secure* RAM. While this solution is secure against the physical attacks we consider, it is too heavyweight for use in many systems. Especially for more general-purpose systems such as PCs and gaming systems, the cost in terms of die area of keeping the entire BIOS on-chip (which can be as large as 1MB) as well as the code and data of applications (possibly several MBs) is prohibitive. For this wide range of systems, a more lightweight and robust solution is needed.

It is clear that the root of trust for a secure bootstrapping solution should be located on-chip to provide strong security. However, an on-chip root of trust does not alone guarantee secure bootstrapping. We find that a class of attacks which we classify as a type of *Time-Of-Check To Time-Of-Use (TOCTTOU)* attacks are still possible. The main idea is that after a boot component has been verified, an attacker with physical access to the system can tamper with the component

978-1-4244-5028-2/09/$25.00 ©2009 IEEE

before it is actually fetched and executed by the processor. Hence, the processor will now execute an image that is different from the image that was originally verified. Therefore, as another requirement, a secure bootstrapping mechanism should enforce that a boot component cannot be modified (without detection) after it is initially verified during secure boot and before it is executed by the processor. AEGIS by Arbaugh et al. and TCG do not meet this requirement and hence are susceptible to such TOCTTOU attacks. ARM TrustZone, on the other hand, meets this requirement since verified components never leave the processor chip, however again this is an expensive solution that is infeasible for many systems. As a result, the currently existing solutions to secure booting are inappropriate for implementation on secure processors.

Contributions. In this paper, we propose SHIELDSTRAP, an architecture for protecting the integrity of the bootstrapping process of secure processors. SHIELDSTRAP is designed with three principle design goals: **security, complexity and flexibility**

Security: SHIELDSTRAP is designed to provide protection from sophisticated hardware-based attacks during system booting. Our solution to secure booting is based on the observation that any component off of the processor chip is relatively easy to compromise with hardware-based attacks. Whereas on-chip components are significantly more difficult to attack and can be made more difficult by various manufacturing techniques such as special coating [25]. Hence, the root of trust for the booting process should be located on the processor chip. We further motivate this security model in Section III. SHIELDSTRAP uses a two-phased approach to boot the system to a trusted state: a *verification phase* and a *booting phase*. In the verification phase, SHIELDSTRAP verifies the integrity of the BIOS using on-chip components (our root of trust) before passing control to the BIOS to start the actual booting phase. In the booting phase, we also follow a chained integrity verification approach where each layer verifies the integrity of the next layer in the boot chain before passing control to it. A failure at any point will prevent the system from booting. SHIELDSTRAP offers distinct security advantages over the previously proposed approaches for secure booting. First, SHIELDSTRAP protects systems from even sophisticated hardware attacks during booting. Hardware attacks against a boot component will be detected since that component's integrity verification during our secure boot process will fail. This is accomplished with only lightweight additions to the processor and small changes to the boot procedure. On a reset (hard/soft), the processor instead of directly executing the BIOS, jumps to code stored in an on-chip memory which we call the SHIELDSTRAP ROM (ST-ROM). The ST-ROM is responsible for carrying out the verification phase to establish the integrity of the BIOS before handing over control to the BIOS to start the booting phase. This is the first work to bridge the gap between the vulnerable initialization of a system to the secure steady state execution environment provided by secure processors.

Secondly, SHIELDSTRAP leverages secure processor support to prevent TOCTTOU-style attacks during booting. After the integrity of a boot component is verified, integrity information about that component is retained on-chip such that any subsequent tampering with the component will be detected when the component is executed by the processor at a later time. Lastly, similar to current solutions, SHIELDSTRAP continues to provide protection against the most widespread software attacks that target the boot components. We provide a security analysis of SHIELDSTRAP to motivate its security against several known boot-time attacks.

Complexity: We show how our SHIELDSTRAP mechanisms can be combined with typical memory encryption and authentication mechanisms provided by a secure processor to protect a system and the programs it executes all the way from power-on to steady-state application execution. In addition to enhancing the overall system security, this novel use of memory encryption and authentication for implementing a secure boot mechanism significantly reduces its complexity with respect to the on-chip storage overheads. The boot components can now be safely evicted to off-chip memory where they automatically come under the protection of the memory encryption and authentication mechanisms. This enables SHIELDSTRAP to have a very small on-chip area overhead making it suitable for use in both embedded and general-purpose systems. Our work explores the interaction of secure memory with a secure booting mechanism. We present a detailed analysis of the complexity and overheads of a secure booting mechanism with and without secure memory support. We also provide an area estimation for the amount of on-chip hardware that is required for SHIELDSTRAP and we find that this overhead is reasonable at less than 0.1% of the chip area. We also provide an evaluation to show that SHIELDSTRAP adds a negligible overhead of 0.37 seconds during system booting (which is not typically a latency-critical process), compared to a base system with no secure booting.

Flexibility: Finally, SHIELDSTRAP is flexible, allowing for hardware and software reconfiguration without requiring changes for the end-user. For example, the user can install a new operating system (software reconfiguration) or install a new expansion card, such as a graphics card (hardware reconfiguration), without causing boot failures. We also explore the use of a group signature scheme, Direct Anonymous Attestation [26], to allow the system to use the BIOS from various vendors while ensuring the integrity of the secure booting mechanism, thereby preventing tying the processor manufacturer to specific BIOS manufacturers. However, our scheme places some requirements on the boot components designed to run on a secure system using the SHIELDSTRAP architecture. In particular, SHIELDSTRAP requires the boot components to be signed by their respective manufacturers before installation on the system. We discuss the particular requirements of each component in section IV.

The rest of the paper is organized as follows. Section II discusses the related work on secure booting, Section III describes our assumed attack model. Section IV describes our proposed SHIELDSTRAP architecture in detail. To evaluate SHIELDSTRAP, we first present a qualitative security discussion in Section V-A. Then we present quantitative results to summarize the area overheads and boot-time performance overheads of SHIELDSTRAP in Section V-B. Finally, we conclude in Section VI.

II. Related Work

Despite the growing importance of the problem, current solutions suffer from critical limitations in terms of security, cost, and flexibility. We categorize the related works in to three categories.

In **Category 1**, we have solutions that have a low complexity but do not provide complete security. This is due to the fact, that these solutions rely on an off-chip component like BIOS to from the root of trust for the booting process [21], [23] which makes them vulnerable to hardware attacks. MIT AEGIS [17], on the other hand, proposes a secure boot mechanism where the public key of the security kernel manufacturer is embedded on the processor chip, and is used to authenticate the OS when entering trusted execution mode. Other components involved in the boot process (e.g. BIOS, expansion ROMs) are not verified and thus are vulnerable to boot-time attacks. In particular, MIT AEGIS cannot provide defense against boot attacks targeting the BIOS (e.g. Modchip attacks), the expansion ROMs and MBR (e.g. rootkits persisted in expansion ROMs and MBR [27]). In essence, the attack model for MIT AEGIS does not consider hardware attacks on boot components to be a possibility. MIT AEGIS also lacks flexibility because the embedded OS-vendor public key prevents installing OSes from different vendors. In comparison, SHIELDSTRAP places the root of trust on-chip and verifies all components during the bootstrap process to protect the secure bootstrap process against even hardware attacks targeting any of the boot components. In addition, SHIELDSTRAP is also flexible in allowing hardware and software upgrades by decoupling the secure bootstrap mechanism from the system configuration.

In **Category 2**, we have solutions that offer low security and in addition have high complexity in terms of the hardware required. TCG [22] and Microsoft Microsoft's Next-Generation Secure Computing Base (NGSCB) fall in this category. TCG still relies on the BIOS to establish the root of trust and is optional for end users and NGSCB despite reducing the reliance on off-chip components continues to be optional for the end users allowing adversaries to bypass the security mechanisms. In addition, these approaches are hardware and software intensive, for example, NGSCB requires changes to the CPU, chipset, USB I/O and GPU components, in addition to requiring a new OS component called the nexus. SHIELDSTRAP requires much more simple hardware and software changes and is an architectural mechanism which cannot be bypassed by users. IBM Cell BE architecture [28] supports the *Runtime Secure Boot* feature, which is used to check the integrity of application code as it is loaded or as it runs, but not the bootstrap of the system. The Cell security model, hence, does not protect against boot time hardware attacks. Modchips for Sony Playstation 3 using an IBM Cell processor will soon be available [29].

In **Category 3**, we have solutions that provide defense against hardware attacks but are hardware intensive and even infeasible to be used for general purpose systems. ARM TrustZone [24] falls in this category. In TrustZone, a new processor mode is introduced, called the Secure Monitor Mode, which is supported by the introduction of a new security bit (S-bit). The caches, TLB and MMU are tagged with the S-bit and partitioned to provide isolation between secure and non-secure applications. A secure application is executed from the on-chip Secure RAM and cannot be evicted from the chip to provide protection against TOCTTOU-style attacks. TrustZone also moves the root of trust on-chip into a ROM which stores the first level boot loader. This solution targets consumer products such as mobile phones and PDAs with small boot code requirements (8-16 KB) and applications with small code footprints. However, general purpose and gaming systems have typical BIOS sizes of 256KB and 1MB respectively, and generally execute much larger applications. The TrustZone approach is impractical for these types of systems, while SHIELDSTRAP requires much more lightweight on-chip storage overheads of less than 0.1% on-chip area overhead.

We also note that there have been a variety of studies on secure processor architectures [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], which consist of hardware-based solutions for ensuring the confidentiality and integrity of computations even in the face of relatively sophisticated hardware attacks. While the proposed secure processor architectures enforce the confidentiality and integrity of computations at run-time, it is assumed that the system has been booted and initialized securely by including the booting process in the Trusted Computing Base (TCB). This assumption however can render the system vulnerable unless a secure booting mechanism is in place which is resistant to even hardware-based attacks as SHIELDSTRAP is.

The following figure summarizes the solutions to secure booting in terms of security and cost, and we show where SHIELDSTRAP fits.

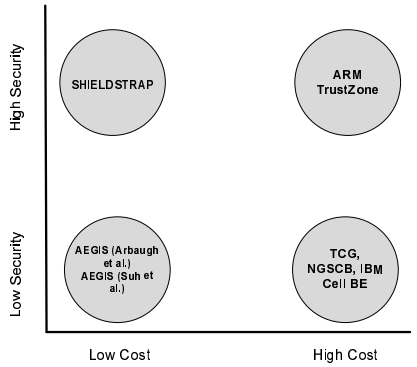

Fig. 1. Comparison of Secure Booting mechanisms with SHIELDSTRAP

III. Attack Model and Assumptions

As in all prior studies on secure processor architectures which consider hardware attacks, our attack model identifies two regions of a system. The secure region consists of the processor chip itself. We assume that the attackers cannot tamper with code or data stored on-chip (e.g. in registers or caches). The non-secure region consists of all off-chip structures including the buses, memory, disk, BIOS and expansion cards. Any data or code stored in any of these structures can be observed and modified by the attackers.

In this paper, we focus on attack scenarios where the attacker has physical access to the system and can perform

hardware-based attacks, including the attack scenarios where the owner of the system is the attacker itself. For instance, this is the case in gaming systems where the owners hack the system to bypass the security mechanisms used by the manufacturers to enforce Digital Rights Management. This enables the owners to use the system beyond the intended purposes for which the system was originally designed. Our proposed technique, however, is not restricted in application to gaming systems and is equally applicable to other embedded or general purpose systems. Finally, we also consider software boot attacks to be equally likely and continue to defend against them similar to prior solutions.

IV. SHIELDSTRAP ARCHITECTURE

A. SHIELDSTRAP

In light of the problems with current approaches to secure bootstrapping and the growing need for such a mechanism, we propose our solution to secure booting, SHIELDSTRAP. SHIELDSTRAP is specifically designed to provide protection against hardware attacks and continues to provide effective defense against software attacks that target the boot components. SHIELDSTRAP uses an on-chip memory, which we call the *SHIELDSTRAP ROM (ST-ROM)* as the core root of trust to start our bootstrap verification process. We make no assumptions about the integrity of any of the off-chip components (including the BIOS). The proposed solution is based on the observation that the security attacks in prior solutions were made feasible because off-chip components, namely the BIOS, were included in the TCB. Moving the root of trust on-chip into the ST-ROM addresses this vulnerability. The proposed architecture necessitates several changes to the current boot process, and we discuss these later in this section.

The booting process with the SHIELDSTRAP architecture consists of two phases, the *verification phase* and the *booting phase*. On a hard or soft system reset, the verification phase is started where the ST-ROM reads the BIOS from an off-chip ROM and verifies its integrity by checking its signature. If the signature is verified successfully, the system transitions to the booting phase where control is passed to the BIOS to start the booting process. In the booting phase, we follow a chained integrity verification where each layer verifies the integrity of the next layer in the boot sequence before passing execution control to it. This requires each component involved in booting, namely the BIOS, the expansion ROMs, the primary and secondary boot blocks and the OS image to be signed by their respective manufacturers. An exception to this requirement is the BIOS, which can either be signed by the manufacturer itself or by the processor manufacturer depending on the type of system in question as discussed later in this section. Figure 2 outlines the SHIELDSTRAP approach to secure booting.

Secure processors are equipped with hardware memory encryption and authentication mechanisms. Memory encryption provides protection against *passive* attacks, where the attacker tries to observe data communicated in plaintext between the processor and main memory. This is done by encrypting and decrypting data and code as it moves on and off the processor chip. Memory authentication is achieved

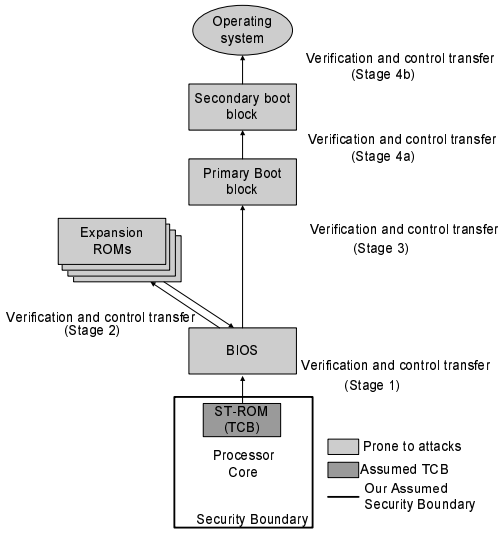

Fig. 2. SHIELDSTRAP: Secure bootstrap mechanism

by building a tree of MAC (Message Authentication Code) values over the memory (called a *Merkle* tree). The root of the tree is stored securely in an on-chip register and never goes off-chip. A block loaded from memory is verified for integrity by checking it against the chain of MAC values up to the root. SHIELDSTRAP leverages these mechanisms to significantly reduce the design and storage complexity of the secure booting solution. The boot components do not have to be retained on-chip for verification or execution and can be safely evicted to the main memory where they automatically come under the integrity and confidentiality protection provided by the secure processor. In effect, SHIELDSTRAP effectively eliminates the window between the verification and start of execution of a boot component *vulnerability window* , where an attacker with physical access to the system can change the boot component, thereby eliminating any possibility of TOCTTOU attacks. However, processors have the ability to read directly from the BIOS ROM chip allowing the code to bypass the memory hierarchy and thereby bypass the secure processor protection. Hence, we propose to make it mandatory to shadow the BIOS ROM and not provide it as an option. With shadowing always enabled, the BIOS will first be copied to the main memory before it can be executed by the processor to boot the system. This will ensure that the BIOS always comes under the integrity protection mechanism of the secure processor substrate used by SHIELDSTRAP. Most systems use this optimization to speed up accessing the ROMs and it is recommended to set this option on anyway. Hence, this requirement does not restrict the design of a system in any way. We describe one TOCTTOU attack in section V-B and show how SHIELDSTRAP defends against such attacks. We next discuss the requirements for each of the boot components.

1) SHIELDSTRAP-ROM (ST-ROM): SHIELDSTRAP uses on-chip storage, the ST-ROM, as the root for establishing trust. The ST-ROM lies inside the security boundary of the processor chip. In this section, we highlight the requirements and operation of the ST-ROM.

The ST-ROM stores the following components:

- Public key of the BIOS or processor manufacturer

- RSA [30] and SHA [31] implementations
- ST-code

Public key of the BIOS or processor manufacturer

One may assume that we would want to authenticate the BIOS by having the ST-ROM verify the BIOS signature using the public key <e,n> of the BIOS manufacturer which is stored on the BIOS itself and is signed with the private key of a Certificate Authority (CA). The public key of the CA would be stored on the ST-ROM. However, we note that simply proving that the BIOS is an authentic one that has been signed by the CA may not be the most desirable solution. The reason is that the CA will likely sign a large variety of software, and which software is signed is not under the control of any particular manufacturer. Thus, a modding attack, where the modchip is equipped with another BIOS that has been signed by the same CA, could be used to supply a BIOS to the processor that will pass the verification check of the ST-ROM, but does not contain the correct code to verify the rest of the components in the boot sequence.

Instead we propose that the ST-ROM should directly store the public key that will be used for establishing the authenticity of the BIOS. This ensures that when the ST-ROM loads and verifies the BIOS against its digital signature, the BIOS is guaranteed to be one which was intended to run on the system and which contains the necessary code to verify the next components in the booting. An obvious choice is to store the public key of the BIOS manufacturer in the ST-ROM. This public key can then be used to authenticate the BIOS signed by its manufacturer. We believe storing the public key of the BIOS on-chip will be suitable for systems where the system manufacturer also manufactures the BIOS. For example, in gaming systems the system manufacturers produce the BIOS as well, since the BIOS is used for specialized tasks such as DRM enforcement. However, for other secure systems, requiring the public key of the BIOS manufacturer to be stored in the on-chip ST-ROM will reduce the flexibility of the system by tying the processor manufacturer to a particular BIOS manufacturer. In order to not trade flexibility for security, we propose using the processor manufacturer's key to verify the authenticity of the BIOS. This in-turn would require the BIOS manufacturers to send their code to the processor manufacturer for signing before it could be used by the system. At boot time the processor's public key stored in the ST-ROM will be used to verify the digital signature off the BIOS. We believe that this requirement is reasonable as processor manufacturers implementing a secure boot mechanism would anyways want to have some form of control over the system configuration. A similar model is currently being employed by the Windows Logo Program [32], where the driver manufacturers are required to get a certification from Microsoft before the driver could be installed on the operating system without warnings.

Group Signature scheme : In order to further decouple the processor manufacturers from the BIOS manufacturers, we explore the use of a Group signature scheme like Direct Anonymous Attestation (DAA) [26] for verifying the authenticity of the BIOS. In DAA, a single *group public key* can correspond to multiple private keys, that is, a single group public key can be used to verify signatures signed by multiple private keys. For our purposes, the processor manufacturer can embed the group public key in the ST-ROM and each of the BIOS manufacturer, desiring to be compatible with the processor, can obtain a private key from the processor manufacturer to sign the BIOS code. This scheme retains the flexibility advantage of the scheme where the processor manufacturer's public key is used to verify the BIOS, but at the same time, it also avoids the extra indirection, where the BIOS manufacturers need to get their BIOS signed by the processor manufacturers.

RSA and SHA implementations

We propose to verify the signatures of the boot components using software stored on the ST-ROM. This requires the ST-ROM to store an implementation of a digital signature algorithm. We use RSA for signature verification with the SHA algorithm used for generating the hash of the component. Our choice of RSA over Digital Signature Standard [33] was driven primarily by two factors: storage and speed. As we propose storing software implementations of the algorithms on-chip, the algorithm chosen should be space efficient. Also, it is desirable that the algorithm is faster for the verifier than for the signer. Since each boot component is only signed once by the manufacturer, but the signature is verified on each boot attempt, this helps to lower the execution time overheads due to our secure bootstrapping mechanisms. RSA with SHA compares favorably to DSS with SHA in terms of both storage and verification speed, hence our choice of RSA plus SHA. Performing the signature verification in software minimizes additional hardware needed for the proposed SHIELDSTRAP architecture, but at the same time will be slower than a corresponding hardware implementation. However,we show in our results that using a software approach adds an insignificant performance overhead to the boot time compared to a base system with an unprotected boot process.

ST-Code

On a system reset, the processor jumps to execute the ST-ROM code. The ST-ROM code which we call the SHIELDSTRAP Code (ST-Code) is a simple algorithm that carries out the verification phase. The ST-Code starts by reading the signature of the BIOS. A digital signature is generally not computed over the entire range of code/data itself, but instead a much smaller representative form of the code/data is signed. The representative form we use is the running hash of the code/data. This means that a hash should first be computed on the BIOS code and this hash must then be signed with the private key of either the BIOS manufacturer or the processor manufacturer depending on the flexibility requirements of the system as discussed above. The ST-Code will read the BIOS location by location and compute a running hash on it. Once the hash for the BIOS code is obtained, the ST-Code proceeds to verify the computed running hash. The ST-Code uses the public key stored in the ST-ROM to get the *expected hash* from the signature. If the running hash matches the expected hash, the ST-Code concludes that the BIOS has been verified successfully, and control is passed to the BIOS to start the booting phase. The operations carried out for verifying the BIOS signature can be summarized as follows:

$$\begin{aligned} public\_key &\Leftarrow < e, n > \\ expected\_hash &\Leftarrow bios\_signature^{e} mod\ n \\ running\_hash &\Leftarrow HASH(BIOS\ code) \\ running\_hash &\ ?= expected\_hash \end{aligned}$$

where, <*e,n*> is the public key stored in the ST-ROM, *expected_hash* is the hash obtained from the signature of the BIOS using the public key and *running_hash* is the hash computed by ST-Code over the BIOS that is being verified.

In the event of an integrity failure at any layer, the bootstrap process is halted. We note that preventing the system from booting up in the event of failure and not recovering (as in [21]) may lead to denial of service. However, for the attack model we consider where the attacker has physical access to the system, causing a boot failure on integrity check failures will prevent the attack from being successful. This forces the attacker (the user/owner) to restore the original hardware configuration of the system, thereby preventing the system from being used for purposes other than those intended.

Figure 3 shows the minimal architectural modifications required for SHIELDSTRAP. Over a secure processor architecture, we only need to add an on-chip memory, the ST-ROM, which forms the root of trust for the bootstrap process.

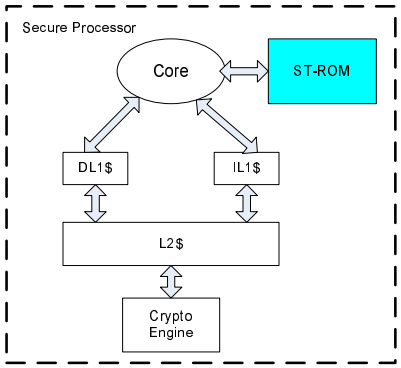

Fig. 3. SHIELDSTRAP: Architecture changes

2) SHIELDSTRAP: Boot component requirements : **Requirements of BIOS manufacturer**

SHIELDSTRAP requires either the BIOS or the processor manufacturer to sign the BIOS code using their private key. Additionally all further BIOS upgrades should also be signed by the BIOS or processor manufacturer. Signed upgrades are necessary to ensure the system boots up properly on legitimate BIOS upgrades. The ST-ROM will store the corresponding public key of the BIOS or processor manufacturer and verify the digital signature of the BIOS before handing over control to the BIOS to start the normal booting process of the system.

Requirements of other boot components

The other components involved in the booting chain include the expansion ROMs, the primary boot block, the boot loader (in multi-boot systems) and the OS image. SHIELDSTRAP requires the manufacturers of these boot components to sign their components and include their public key signed by a certificate authority with the component. The boot components involved in verifying other components should also include the public key of the certificate authority to verify the signed public key of the next component in the boot chain. We chose this approach as opposed to requiring the boot components in one layer of the boot chain to store the public keys of the components in the next layer to favor flexibility without sacrificing security. If a boot component stores the public key of the next layer's components, it limits the flexibility of the system, as a user cannot add new software or hardware components without requiring the lower layer component to be upgraded as well to include the public key of the new component manufacturer.

V. Evaluation

We present a qualitative and quantitative evaluation of SHIELDSTRAP in this section. The primary evaluation of a secure bootstrap mechanism is the ability to defend against boot time attacks, which can either be hardware or software attacks. Hence, we first analyze SHIELDSTRAP qualitatively, by presenting a detailed security analysis of the proposed SHIELDSTRAP secure boot architecture in section V-A.

SHIELDSTRAP adds an on-chip ST-ROM that forms the root of trust for the booting process. Hence, it is critically important for the on-chip area overhead to be within reasonable limits for the proposed scheme to be practical for implementation. Also, despite the fact that booting is not a time critical process, it is desirable that the proposed mechanism does not add an undue amount of time to the boot process. Hence, we qualitatively analyze SHIELDSTRAP by presenting an area and performance analysis in section V-B.

A. SHIELDSTRAP Security Analysis

Table I summarizes the main categories of attacks that can be conducted by an attacker during boot time to subvert the boot process and use the system for purposes for which it was not intended. Since the verification phase checks the BIOS against its signature using the trusted, on-chip ST-ROM, the attacker cannot conduct any attacks before the verification phase is complete, as it will result in a boot failure due to a signature mismatch. Once the verification phase is complete, control can be passed to the now trusted BIOS which is executing under protection from the secure processor. Thus, any attacks against the BIOS after this point will be detected by the secure processor mechanisms. As such, the BIOS can be trusted to verify the next components in the boot process against their signatures, and subsequently pass control to them as they are now verified and running under the protection of the secure processor. This procedure continues until all boot components have been verified and executed and the OS has been loaded.

B. SHIELDSTRAP Area and Performance Evaluation

1) Area Overhead: To evaluate the on-chip area required for the ST-ROM, we implemented SHA and RSA in software with space efficiency as the primary goal. The object code sizes for SHA and RSA that will be stored on the ST-ROM measure 8266 bytes and 4894 bytes respectively. Along with the implementations of these cryptographic algorithms, the ST-ROM also needs to store the public key of the BIOS

TABLE I
SHIELDSTRAP: DEFENSE AGAINST POSSIBLE BOOT ATTACKS.

Type of Attack	Description	SHIELDSTRAP's response
Pre-boot attacks (Hardware attacks)	This category of attacks includes attacks where the hardware boot components namely BIOS and expansion cards are have been replaced prior to booting. (e.g. modchips [1], [2], [3], [4], [5], [6])	Signature failure during the verification phase. Action: Boot failure.
BIOS Flashing Attacks (Hardware attacks)	This category of attacks include attacks where the BIOS is flashed with an alternate BIOS, possibly from a legitimate manufacturer (e.g. the Cromwell BIOS). TSOP flashing attack conducted on earlier versions of Xbox falls in this category. The BIOS was stored on a chip called TSOP, whose write-access could be enabled by soldering a few points on the motherboard.	Signature failure during the verification phase. Any BIOS flashing other than a legitimate upgrade from the BIOS manufacturer triggers a signature verification failure, preventing the system from booting. Action: Boot failure.
Vulnerability window attacks (TOCTTOU attacks) (Hardware attacks)	The attack could be thought to proceed as follows. Verification phase completion is signalled by the processor when it starts to execute the BIOS in the booting phase. A snooping device attached on the control bus could snoop this action from the processor, and at this time redirect fetch requests from the processor to a modchip containing an alternate BIOS. This attack would trick the processor into executing a malicious BIOS, which it thinks it has already verified as trusted.	Detected by the underlying secure processor mechanisms as an integrity failure. Action: System will halt and signal alert.
Software Attacks	This category of attacks includes attacks where the boot components are used to persist rootkit code. (e.g. PCI rootkits [34], MBR rootkit [27], [35]	Signature failure during the booting phase. Action: Boot failure.

which takes up another 1024 bits (128 bytes), and the ST-Code which uses the above implementations to read in the BIOS and carry out the verification phase. Hence, the total area required for ST-ROM is at most 13288 bytes (13KB), so we propose adding a 16KB on-chip ST-ROM.

The on-chip ROM can be implemented using either the MOS NOR ROM implementation having a cell size of 9.5λ x 7λ or the MOS NAND ROM implementation having a cell size of 5λ x 6λ, where λ is 1/2 x *feature size*. The dimensions for the two implementations have been taken from [36]. We assume a MOS NOR ROM implementation for ST-ROM to derive an upper-bound on the area overhead. Using Mosis scalable CMOS rules [37], and a feature size of 90 nm, the area taken by the 16KB ST-ROM is 0.017 mm^2. On an Intel Pentium 4 with a die area of 143 mm^2 [38], this amounts to an on-chip area overhead of 0.012%. Hence, SHIELDSTRAP requires minimal on-chip area overheads and should be feasible to incorporate onto the processor chip.

2) Experimental Setup: We use SESC [39], a cycle accurate, execution driven simulator, to model a secure processor substrate for this research. The secure processor simulated is based on counter mode encryption and Merkle tree authentication mechanisms. We model a 2GHz, 3-issue, out-of-order processor with split L1 data and instruction caches. Both caches have a 32KB size, 2-way set associativity, and 2-cycle round-trip hit latency. The L2 cache is unified and has a 1MB size, 8-way set associativity, and 10-cycle round-trip hit latency. For counter mode encryption, the processor includes a 32KB, 16-way set-associative counter cache at the L2 cache level. All caches have 64B blocks and use LRU replacement. We assume a 1GB main memory with an access latency of 200 processor cycles. The encryption/decryption engine simulated is a 128-bit AES engine with a 16-stage pipeline and a total latency of 80 cycles, while the MAC computation models HMAC [40] based on SHA-1 [31] with 80-cycle latency [41]. The default MAC size is 128 bits.

We simulate the cryptographic operations for SHIELDSTRAP using an open source cryptographic library, Crypto++ version 5.5.1 [42], which provides the implementation for RSA and SHA algorithms used for signature verification. The RSA algorithm is based on Public-Key Cryptographic Standards (PKCS) version 1.5 [30], which defines the RSA encryption standard. The SHA algorithm is implemented based on the standard published by NIST [31]. We use key lengths of 1024-bit for RSA and SHA-256 which generates a 256-bit authentication code used for signing purposes. As discussed previously, the choice of these cryptographic algorithms was based on two factors other than the proven security strengths of these algorithms: Storage and speed.

3) Boot-Time Overhead: We modeled the SHIELDSTRAP booting process using the Crypto++ cryptographic library. We assume a system with a 256KB BIOS, two expansion cards with 32KB of expansion ROM on each, Master Boot Record of 512 bytes, a boot loader with a size of 200KB (most common boot loaders have a a size ranging from 150KB - 200KB), and an OS image of size 2MB (the compressed image size for linux kernel 2.6.9). We implemented code to model the verification phase of SHIELDSTRAP (e.g. reading the boot components, computing their running hash value, and verifying their signature), and we have timed this process on an Intel Pentium 4 processor running at a frequency of 2GHz. The system has split L1 data and instruction caches. Both caches have a size of 32 KB. The L2 is unified and has a 1MB size. We find that SHIELDSTRAP will add a negligible overhead of 0.37 seconds on average relative to the boot process on a system with no protection.

Further, to approximate the impact of the interaction of SHIELDSTRAP and a secure processor on the boot-time overhead, we ran the code modeling SHIELDSTRAP under SESC. SHIELDSTRAP adds an overhead of less than 0.8% over AEGIS by Arbaugh et al. which primarily comes from the secure processor overheads for memory encryption and authentication.

We also analyzed the breakdown of these minimal overheads. As expected, the OS takes the most time for verification (up to 80% of the total verification time) since it is the largest boot component. We believe that the already insignificant overheads of our scheme will be even lower for specialized systems like gaming and embedded systems as they generally employ a significantly reduced version of the operating system kernel compared to those used in general purpose systems.

VI. Conclusions

Despite the increasing number of attacks which target systems during the boot phase in order to subvert various security mechanisms employed by the system, there have been few studies on secure booting mechanisms. In addition, prior secure booting approaches are vulnerable in the scenario where the user of the system is considered the attacker and can conduct hardware-based attacks. We propose SHIELDSTRAP, a novel secure booting mechanism that is secure even against relatively sophisticated hardware attacks. Our key insight in SHIELDSTRAP is that we move the root of trust for verifying the integrity of all boot components onto the processor chip which is our natural security boundary. We analyze how SHIELDSTRAP is secure against common boot attacks. We also show that SHIELDSTRAP requires an acceptably small amount of on-chip hardware, and that it adds an insignificant amount of execution time overhead to the normal boot process.

References

[1] americanxboxmodchips.com, *http://www.americanxboxmodchips.com/*.

[2] http://www.modchip.com, 2005.

[3] mod-chip.com, *http://www.mod-chip.com/*.

[4] modchipoutlet.com, *http://www.modchipoutlet.com/*.

[5] modchipstore.com, *http://www.modchipstore.com/*.

[6] wii-modchips.com, *http://www.wii-modchips.com/*.

[7] M. G. Kuhn, "Cipher Instruction Search Attack on the Bus-Encryption Security Microcontroller DS5002FP," *IEEE Transactions on Computers*, vol. Oct, no. 10, 1998.

[8] B. Gassend, G. Suh, D. Clarke, M. Dijk, and S. Devadas, "Caches and Hash Trees for Efficient Memory Integrity Verification," in *Proc. of the 9th International Symposium on High Performance Computer Architecture*, 2003.

[9] T. Gilmont, J.-D. Legat, and J.-J. Quisquater, "Enhancing the Security in the Memory Management Unit," in *Proc. of the 25th EuroMicro Conference*, 1999.

[10] IBM, "IBM Extends Enhanced Data Security to Consumer Electronics Products," *http://domino.research.ibm.com/comm/pr.nsf/pages/news.20060410_security.html*, April 2006.

[11] D. Lie, J. Mitchell, C. Thekkath, and M. Horowitz, "Specifying and Verifying Hardware for Tamper-Resistant Software," in *Proc. of the 2003 IEEE Symposium on Security and Privacy*, 2003.

[12] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. MItchell, and M. Horowitz, "Architectural Support for Copy and Tamper Resistant Software," in *Proc. of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2000.

[13] B. Rogers, S. Chhabra, Y. Solihin, and M. Prvulovic, "Using Address Independent Seed Encryption and Bonsai Merkle Trees to Make Secure Processors OS- and Performance-Friendly," in *Proc. of the 36th Annual International Symposium on Microarchitecture*, 2007.

[14] W. Shi, H.-H. Lee, M. Ghosh, and C. Lu, "Architectural Support for High Speed Protection of Memory Integrity and Confidentiality in Multiprocessor Systems," in *Proc. of the 13th International Conference on Parallel Architectures and Compilation Techniques*, 2004.

[15] W. Shi, H.-H. Lee, M. Ghosh, C. Lu, and A. Boldyreva, "High Efficiency Counter Mode Security Architecture via Prediction and Precomputation," in *Proc. of the 32nd International Symposium on Computer Architecture*, 2005.

[16] W. Shi, H.-H. Lee, C. Lu, and M. Ghosh, "Towards the Issues in Architectural Support for Protection of Software Execution," in *Proc. of the Workshop on Architectural Support for Security and Anti-virus*, 2004.

[17] G. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, "AEGIS: Architecture for Tamper-Evident and Tamper-Resistant Processing," in *Proc. of the 17th International Conference on Supercomputing*, 2003.

[18] G. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, "Efficient Memory Integrity Verification and Encryption for Secure Processor," in *Proc. of the 36th Annual International Symposium on Microarchitecture*, 2003.

[19] C. Yan, B. Rogers, D. Englender, Y. Solihin, and M. Prvulovic, "Improving Cost, Performance, and Security of Memory Encryption and Authentication," in *Proc. of the International Symposium on Computer Architecture*, 2006.

[20] J. Yang, Y. Zhang, and L. Gao, "Fast Secure Processor for Inhibiting Software Piracy and Tampering," in *Proc. of the 36th Annual International Symposium on Microarchitecture*, 2003.

[21] W. Arbaugh, D. J. Farber, and J. M. Smith, "A Secure and Reliable Bootstrap Architecture," in *Proc. 1997 IEEE Symposium on Security and Privacy*, 1997.

[22] TCG, "TCG PC Client Specific Implementation Specification For Conventional BIOS," *https://www.trustedcomputinggroup.org/sspecs/PCClient/TCG_PCClientImplementationforBIOS_1-20_100.pdf*, April 2006.

[23] N. Itoi, W. A. Arbaugh, S. Pollack, and D. M. Reeves, "Personal Secure Booting," in *Proc. of the Sixth Australian Conference on Information Security and Privacy*, 2001.

[24] ARM, "ARM TrustZone," *http://www.arm.com/products/esd/trustzone_home.html*, 2004.

[25] Maxim/Dallas Semiconductor, "DS5002FP Secure Microprocessor Chip," *http://www.maxim-ic.com/quick_view2.cfm/qv_pk/2949*, 2007 (last modification).

[26] E. Brickell, J. Camenisch, and L. Chen, "Direct Anonymous Attestat," in *ACM Conference on Computer and Communications Security*, 2004.

[27] PandaLabs, "Quarterly Report PandaLabs," *pandalabs.pandasecurity.com/blogs/images/PandaLabs/2008/04/01/Quarterly_Report_ PandaLabs_Q1_2008.pdf*, 2008.

[28] IBM Corporation, "The Cell Broadband Engine processor security architecture," http://www-128.ibm.com/developerworks/power/library/pa-cellsecurity/, 2006.

[29] OZMODCHIPS, "Sony Playstation 3 modchips," http://www.ozmodchi- ps.com/ps3-playstation-3-modchip-p-68.html, 2008.

[30] FIPS Publication 197, "RSA Cryptography Standard," http://www.rsa.com/rsalabs/node.asp?id=2125, 1993.

[31] FIPS Publication 180-1, "Secure Hash Standard," National Institute of Standards and Technology, Federal Information Processing Standards, 1995.

[32] Microsoft, "Windows Logo Program," *http://www.microsoft.com/whdc/ winlogo/default.mspx*.

[33] FIPS Publication 186, "Digital Signature Standard (AES)," National Institute of Standards and Technology, Federal Information Processing Standards, 1994.

[34] John Heasman, "Implementing and Detecting a PCI Rootkit," *http://www.ngssoftware.com/research/papers/Implementingi_And_ Detect ing_A_PCI_Rootkit.pdf*, 2006.

[35] D. Soeder and R. Permeh, "eEye BootRoot," *http://research.eeye.com/html/tools/RT20060801-7.html*, 2005.

[36] J. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits: A Desing Perspective*. Prentice-Hall, 2002, vol. II.

[37] MOSIS, *http://www.mosis.com/*.

[38] K. Krewell, "Intel cancels pentium p4," *Microprocessor Report*, 2004.

[39] J. Renau et al, "SESC," *http://sesc.sourceforge.net*, 2004.

[40] H. Krawczyk and M. Bellare and R. Caneti, "HMAC: Keyed-hashing for message authentication," http://www.ietf.org/rfc/rfc2104.txt, 1997.

[41] T. Kgil, L. Falk, and T. Mudge, "ChipLock: Support for Secure Microarchitectures," in *Proc. of the Workshop on Architectural Support for Security and Anti-Virus*, Oct. 2004.

[42] W. Dai, "Crypto++ version 5.5.1," http://www.cryptopp.com/, 1995.