Received 16 April 2013; revised 15 August 2013; accepted 7 October 2013. Date of publication 23 October 2013;
date of current version 7 May 2014.

Digital Object Identifier 10.1109/TETC.2013.2287186

Fabrication Attacks: Zero-Overhead Malicious Modifications Enabling Modern Microprocessor Privilege Escalation

NEKTARIOS GEORGIOS TSOUTSOS[1] (Student Member, IEEE), AND MICHAIL MANIATAKOS[2] (Member, IEEE)

[1]Department of Computer Science and Engineering, New York University Polytechnic School of Engineering, Brooklyn, NY 11201 USA
[2]Department of Electrical and Computer Engineering, New York University Abu Dhabi, Abu Dhabi 628 4313, UAE
CORRESPONDING AUTHOR: M. MANIATAKOS (michail.maniatakos@nyu.edu)

ABSTRACT The wide deployment of general purpose and embedded microprocessors has emphasized the need for defenses against cyber-attacks. Due to the globalized supply chain, however, there are several stages where a processor can be maliciously modified. The most promising stage, and the hardest during which to inject the hardware trojan, is the fabrication stage. As modern microprocessor chips are characterized by very dense, billion-transistor designs, such attacks must be very carefully crafted. In this paper, we demonstrate zero overhead malicious modifications on both high-performance and embedded microprocessors. These hardware trojans enable privilege escalation through execution of an instruction stream that excites the necessary conditions to make the modification appear. The minimal footprint, however, comes at the cost of a small window of attack opportunities. Experimental results show that malicious users can gain escalated privileges within a few million clock cycles. In addition, no system crashes were reported during normal operation, rendering the modifications transparent to the end user.

INDEX TERMS Hardware trojans, fabrication attacks, microprocessors, zero overhead, malicious modification, privilege escalation.

I. INTRODUCTION

Modern microprocessors are ubiquitously deployed in a wide variety of applications: From personal computers, laptops, tablets and cellphones for personal use, to space and automotive applications. It has been reported that the average household in the United States includes on average 40 microprocessors, premium class automobiles carry at least 75 embedded processors, while even low-profile vehicles can still include at least 50 processors [1].

Therefore, ensuring the integrity of a microprocessor is paramount, as security breaches can range from simple information leakage to life-threatening situations. Microprocessor designers incorporate extended security features in latest designs, in an effort to protect the system from external attacks. Due to the globalized supply chain, however, the final design may be tampered with during the design cycle and eventually fail to satisfy the security properties set forth by the designers.

Due to the vast choice of intellectual property (IP) cores, circuit designers as well as system integrators can focus on the development of system architectures instead of manually designing, testing and implementing common functional modules. The wide usage of IP cores, however, comes at the cost of decreased security. Before eventually reaching the system integrator, an IP core has traveled through many stages and is modified by various design houses [2]. There are plenty of design stages for attackers to insert malicious logic in the IP core throughout the whole IP transaction process. Such modifications, commonly refered to as hardware trojans, are purportedly done without the knowledge of the IP consumer. The additional functionality can be exploited by an attacker in order to cause catastrophic results, in case the functional module is embedded into mission-critical devices. Recently, silicon scanning revealed a backdoor in military chips [3], allowing the attacker to disable all the security of the chip. This discovery emphasizes the need for malicious

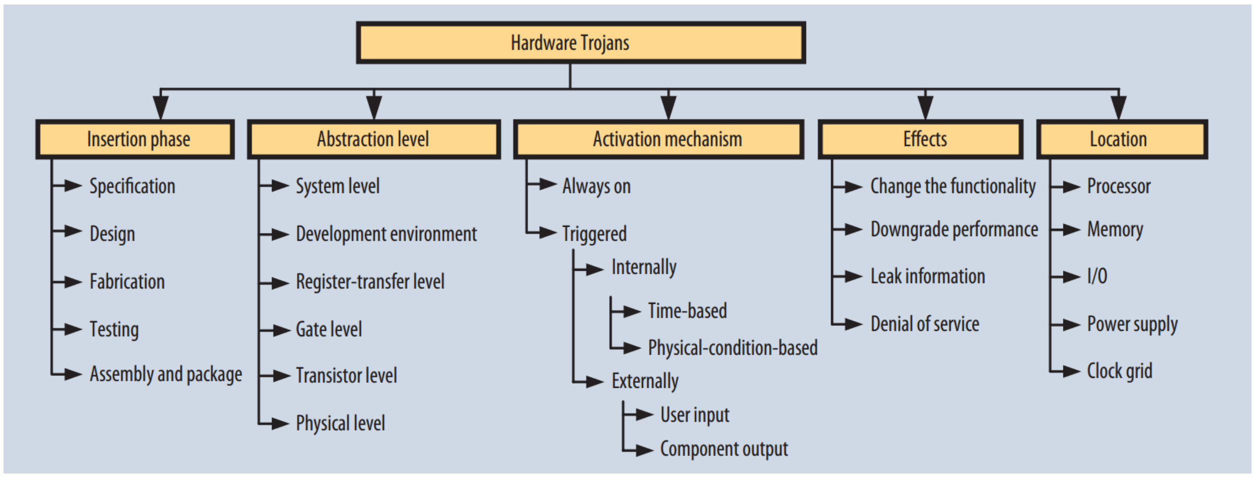


FIGURE 1. **Trojan taxonomy, as presented in [4].**

modification research, as compromised chips are already deployed in the field.

Especially in the domain of modern microprocessors, as discussed previously, malicious modifications can have extensive impact: The dramatic increase of mobile devices, consisting of high-performance, multi-core microprocessors, provide a wide radius of attack surface.

Due to the microprocessor complexity, however, introducing malicious modifications is a very tedious process, as microprocessor designs are very compact with very limited, if any, area for extra malicious circuitry. Therefore, it is very difficult to maliciously insert a non-negligible amount of hardware during the fabrication stage of the microprocessor, as discussed extensively in Section II.

In this paper, we present zero overhead malicious modifications that can be successfully implanted during the fabrication stage of the microprocessor (assuming an untrusted foundry). These alterations, discussed in Section III, can be as simple as an addition of a buffer, a change in the size of a wire, or even a resistive particle that bridges two adjacent wires. The goal of these modifications is to create a temporary malfunction at the system that an attacker can exploit in order to execute unauthorized code or read protected memory locations. The most prominent way to execute unauthorized instructions is through privilege escalation attacks, discussed in Section IV. Examples of such zero overhead modifications are presented in Section V, for both high-performance and high-throughput microprocessors, representing personal computer and embedded microprocessors respectively. Using the experimental platforms presented in Section VI, we demonstrate the quality of the presented malicious modifications in the experimental results, Section VII. Finally, conclusions of the presented work appear in Section VIII.

II. RELATED WORK

Malicious modification insertion and detection has risen as a contemporary topic of interest. An extensive taxonomy of hardware modifications (called ''Hardware trojans'') appears in [4]. The taxonomy, presented in Fig. 1, sorts hardware trojans based on five different attributes: Insertion phase, abstraction level, activation mechanism, effects and location. These attributes provide an indication about several trojan properties, such as area and power overhead, potential impact etc. For example, gate sizing cannot be applied at the Register-Transfer-Level. Similarly, trojans inserted at the fabrication stage must be minimal in terms of area and power overhead.

Researchers are currently focusing their efforts on side channel attacks, such as power or wireless channels [5]–[8], targeting small integrated circuits (ICs). There has been, however, a limited amount of work targeting contemporary, high-performance microprocessors.

Microprocessor trojans targeting an i8051 microprocessor executing a cryptographic algorithm appear in [9], [10], and [11]. In [9], the authors present a set of modifications targeting mostly execution of unauthorized code through architecture changes. Examples include interrupt handler location change, operation code modification and introduction of a transparent one-instruction set computing unit. The modification overhead ranges from 0% to 17.9%, presented on an FPGA. Similarly, [10] presents a 5% area overhead addition of modifications that enable information leakage. The trojan is triggered by the instructions and the key leaks through the transmission channel. Finally, [11] discusses denial-of-service (DoS) attacks through instruction skipping or addition of a functions that force individual chips to fail. All aforementioned solutions, however, target an old microprocessor and since no operating system is used, the solutions can not be ported to latest commercial, high-performance microprocessors.

A more sophisticated public-key encryption circuit attack has been presented in [12]. The authors attack a circuit by turning off portions of the circuit, enabling a key-leaking attack. The malicious hardware still incurs a non-negligible overhead of 406 gates, and can only applied to RSA designs and not generic microprocessors.

The closest work to a modern microprocessor modification has been presented in [13]. The authors present a modification of a high performance microprocessor, where extra hardware is inserted during the design stage, providing the attacker with an extended stage of attack (login backdoor, stealing password etc.). The area overhead, however, is a few thousand gates (the minimum overhead for login attacks is 1,341 gates). Therefore, due to the high area overhead (which is more than a few transistors), the modification cannot be applied during the fabrication stage of the processor and can only be inserted during the design or specification.

III. ZERO OVERHEAD MODIFICATIONS

Given the design phases shown in Fig. 1, the size of the added hardware limits the phase where the trojan can be implanted. As most of the hardware attacks presented in Section II incur a non-negligible overhead (the footprint of the smallest solution requires 406 gates), the addition of circuits of that size during the fabrication stage is difficult. Modern IC floorplanning tools can achieve very high utilization, leaving very small area for addition of extra circuitry. And even if such empty area can be found, the extra hardware has to be connected to very specific parts of the integrated circuit, requiring available interconnection layers that can be extremely long. Furthermore, the attacker must be very careful not to interfere with the critical path of the circuit, rendering the design non operational. All these factors further increase the difficulty of adding trojans during the fabrication stage.

Adding trojans during the design or specification stage is an appealing option, but because the design goes through many iterations and different design teams, the trojan might be detected easily. Moreover, the RTL designer delivers the final design to the physical layout team, and the chance that the extra circuitry will be detected by other people is even higher. Therefore, for all the discussed reasons, we believe that trojans implanted during fabrication are the hardest to detect and are the best candidates to actually make it to a real chip.

To the best of our knowledge, there is no demonstrated attack at the fabrication phase of a modern microprocessor. The main reason for this is the very limited control and potential of fabrication stage modifications. The authors of [4] suggested that the clock trees or the power supply are potential candidates for fabrication stage attacks of high complexity designs. Tampering with the clock tree or the power supply, however, can have an immediate catastrophic impact on the design. The modifications must be minimal and transparent in order to be effective.

In this work, we demonstrate modifications at the physical level of the design that enable malicious users to attack a microprocessor. The modifications are minimal, practically incurring zero area overhead. Such modifications include simple gate alterations, gate additions, gate sizing or interconnection tampering. The goal of the malicious modifications is to introduce a faulty condition (delay fault, coupling fault, bridging fault etc.) that can be exploited by the attacker.

Since the presence of faults can have a catastrophic impact on the microprocessor, the attacker needs to ensure that these faults are activated only under certain conditions: Specific stream of patterns exciting the delay fault, temperature or thermal variations that could enable a bridging fault etc. Ideally, the malicious modifications will create temporary malfunctions that will have no functional impact to the end user. Branch prediction malfunction, supervisor mode temporarily enabled for simple user, unnecessary Translation Lookaside Buffer (TLB) flushing are examples of faults that do not affect the functionality of the microprocessor.

An example of a zero overhead microprocessor modification appears in Fig. 2, where the TLB of the Alpha 21264 microprocessor is presented. In this case, a delay fault is introduced at the most significant bit of the Address Space Number (ASN). A delay fault can be introduced during fabrication by modifying the transistor sizes or the wires between the memory elements storing the next address space identifier, and the location where the comparison occurs. Alternatively, a buffer between the two aforementioned locations can be inserted. In Section V-A.1, we explain how this delay fault can be exploited in order to gain control of the microprocessor.

Since a typical modern microprocessor contains billion transistors, the exploration space of zero overhead malicious modification is practically infinite. Therefore, we define specific properties of the modifications, in order to be able to identify the best modification depending on the attacker's needs. Each property can have a qualitative value in the range *very low, low, medium, high* or *very high*, and the proposed properties are:

1) **Feasibility**: Represents the extent of the physical layer modifications required in order to implant the hardware trojan. Simple wire sizing is an example of high feasibility, while the addition of a few gates greatly limits the feasibility of the trojan during the microprocessor's fabrication stage.
2) **Controllability**: This property represents how often the conditions required for trojan activation are fulfilled (including carefully crafted code that will accelerate the process). If the expected series of microprocessor inputs typically appear every few billion cycles, the modification is classified as highly controllable. In the case that the trojan activation is heavily dependent on the non-determinism of the system load, the operating system context switching etc., the modification has very low controllability.
3) **Visibility**: Represents the effect the modification has on the microprocessor (and the end user) when the necessary conditions are satisfied and there is no attacker. Examples of very low visibility include a branch misprediction or no operation instruction execution.

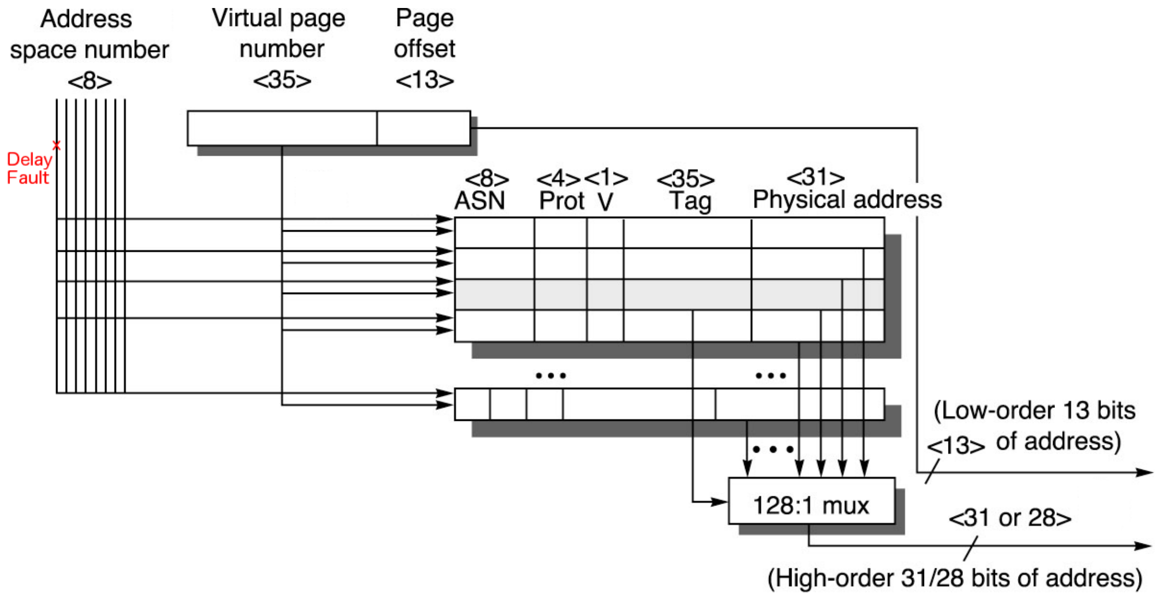


FIGURE 2. **Alpha 21264 memory structure [14].**

Potential catastrophic impacts, such as invalid memory access, are typical cases of very high visibility.

4) **Potential**: Every modification is characterized by different potential, depending on what are the extra features granted to the attacker in case the attack goes through. Gaining root access can provide extreme power to the attacker, while a writing to the memory of another user process might hinder the attacker's efforts to gain system control.

We will use the presented metrics in order to classify the presented microprocessor modifications. In this work, all demonstrated attacks aim for privilege escalation, presented in the next section, offering very high payload potential.

IV. PRIVILEGE ESCALATION ATTACK

In order to ensure proper and secure resource management, modern microprocessors incorporate distinct privilege levels, preventing unauthorized processes to gain access to system or other user resources. The most typical privilege configuration separates the kernel from the user processes, while more advanced configurations incorporate several privilege levels, such as kernel, executive, supervisor, user etc. In case lower privilege level processes request a legitimate access to system resources, the operating system can grant a higher privilege level temporarily.

A *privilege escalation attack* is a type of attack where an unauthorized user gains elevated access to system or other user resources. Privilege escalation attacks typically exploit software or hardware bugs and oversights, as well as poor software configurations. Once elevated privileges are granted, the attacker can access files, view private information (such as encryption keys), modify system files or install unwanted software.

There are two types of privilege escalation:

- **Vertical privilege escalation**: This attack allows the attackers to grant themselves higher privileges. For instance, injecting and executing code at the kernel space, or performing kernel operations that allow unauthorized code execution.
- **Horizontal privilege escalation**: During this attack, the malicious user maintains the same privilege levels, but can gain access to resources belonging to other processes or users sharing the same privilege levels.

The concept of privilege escalation attacks has been extensively studied in the software domain [15], where the attackers exploit programming or hardware bugs to stage privilege escalation attacks [16]. In this research, we consider vertical privilege escalation at the hardware level. This implies that the attacker is trying to gain kernel or supervisor access to resources in order to exploit the underlying vulnerability.

A. PRIVILEGE ESCALATION ATTACK PREREQUISITES

Staging a privilege escalation attack requires the existence of distinct privilege levels, such as kernel, supervisor, user etc., at the microprocessor architecture specifications. Furthermore, software mechanisms (usually at the operating system layer) that escalate privileges during normal operation are also needed.

Almost all modern microprocessors implement privilege levels and provide the necessary mechanisms to handle privileges to the operating system. Specifically:

- **x86**: Privilege levels first appeared with the 80386 [17] in the x86 architecture, with 4 distinct privilege levels (“Rings”): Ring 0 has the highest privileges (kernel mode), while Ring 3 has the least privileges (application mode). Faults in a ring affect only rings of the same or less privileges. In order to assist virtualization, however, contemporary microprocessor extensions introduced a new privilege level (Ring-1), intended to be used by the hypervisor.

- **Alpha**: The latest Alpha microprocessors also implement 4 privilege levels, similar to the x86 architecture: [18]: Kernel, executive, supervisor and user.
- **OpenRISC**: The OpenRISC architecture specification requires the existence of 2 distinct privilege levels, namely Supervisor and User. The levels exist in all available implementations of the OpenRISC processor and are used to protect sensitive kernel structures as well as special purpose registers from unauthorized user access.
- **ARM**: ARM architectures have 1 unprivileged (User Mode) and 3 privileged modes: Supervisor, Interrupt and Fast Interrupt mode. The difference between these modes is the priority the executable code receives when in interrupt and fast interrupt mode.
- **MIPS**: Initially, MIPS required only 2 operating modes (User and Kernel). The addition of many I/O devices in modern SoCs, however, has created the need for implementing two additional modes: Supervisor (I/O access) and EJTAG mode (for debugging purposes).

B. TECHNIQUES TO ESCALATE PRIVILEGES

There are different ways to escalate the process privileges. We should note that such options are heavily instruction-set and architecture dependent; thus, the attacker should have some knowledge about the maliciously modified platform.

The most straightforward way to escalate privileges is to execute an instruction that serves this purpose. In the event that the temporary malfunction allows the attacker to execute any command, the malicious process can get kernel privileges and compromise the system.

Not all instruction set architectures, however, include instructions that directly elevate privileges. Privilege escalation usually occurs in an as-needed basis. Therefore, another way to attack the system to gain escalated privileges is to write directly to the kernel structures. These unauthorized accesses alter the privileges of the attacking process, masking it as a kernel process or altering the allocated virtual address space. This type of payload requires extensive knowledge of the operating system kernel, as well as dynamic information of the attacking process.

Finally, another way to escalate privileges is to overwrite locations where there is a priori known information. Examples of such information include interrupt handlers, shared libraries and operating system specific code (return-to-libc [19] is a common attack using the shared C libraries). This payload requires information about the kernel, but it is independent of where the attacking process is located in memory.

V. DEMONSTRATED ATTACKS

In this section we demonstrate zero overhead modification on modern microprocessors. We have chosen two very different microprocessor models in order to cover the gamut of commercial microprocessors. Furthermore, the chosen microprocessors implement most of the features available in the latest commercial microprocessors (such as out of order execution, speculative execution etc.), rendering them accurate, publicly-available representatives of contemporary microprocessors. Since the demonstrated attacks focus on the architectural features of a single core and not on the chip periphery, single processor models are utilized in this study.

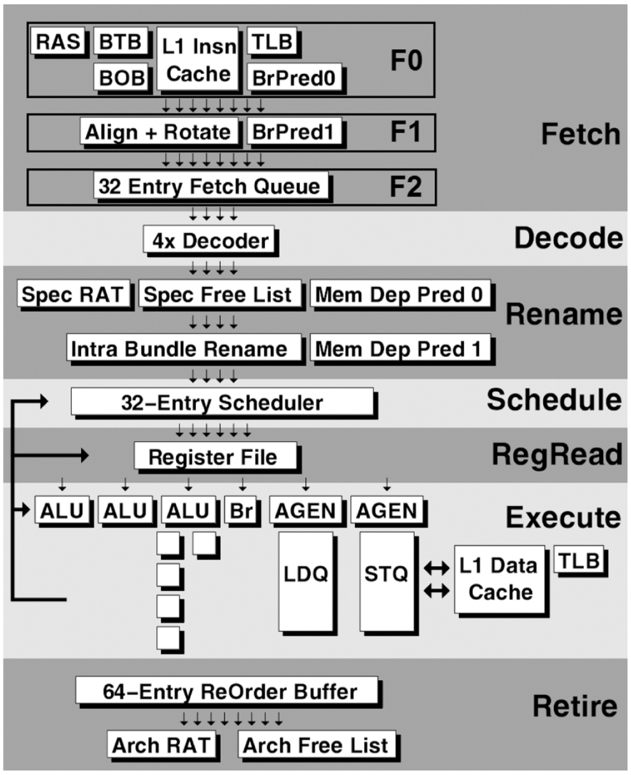


FIGURE 3. Block diagram of Alpha 21264 [21].

- **Alpha 21264**: The Alpha is a concrete example of complex, high-performance microprocessors like the Intel Core or the AMD K10. Featuring a 12-stage pipeline, out-of-order and speculative execution, as well as hybrid 3-stage branch prediction [20], the Alpha's physical design complexity is enormous, making it very difficult for attackers to insert transparent, zero overhead modifications and successfully deploy an attack. The block diagram of the Alpha 21264 appears in Fig. 3.
- **OpenRISC 1200**: The OpenRISC microprocessor represents the embedded processors, like the MIPS or the ARM families. Simple pipelining and in-order, low power execution are common features of such microprocessors. Embedded microprocessors tend to be small and dense, increasing the effort required for fabrication phase attacks. The block diagram of the OpenRISC 1200 appears in Fig. 4.

The following two sections describe carefully crafted, physical layer modifications on the described microprocessors, targeting privilege escalation attacks.

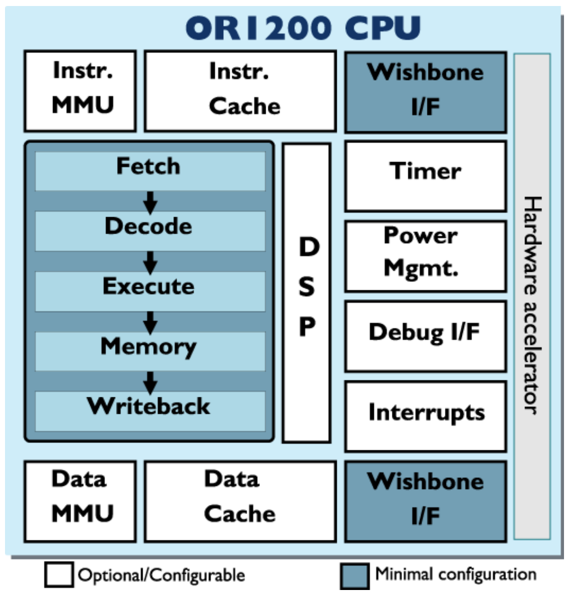


FIGURE 4. **Block diagram of the OpenRISC 1200 [22].**

A. ALPHA 21264

1) ATTACK DESCRIPTION

The trojan is implanted in the Data Translation Lookaside Buffer (DTLB). The DTLB structure of the Alpha 21264 is presented in Fig. 2. The DTLB contains 256 entries and it is fully associative. The translation begins by sending the virtual address to all tags. Once a (valid) match is identified, it goes through a 128:1 multiplexer, augmented with the physical page frame to form the final physical address. The Alpha DTLB also contains a protection permission (“Prot”) and valid (“V”) bits.

The address space number (ASN) is an identifier that groups and identifies addresses belonging to the same process, and ensures that the TLB is not flushed on a context switch. As TLB flushes are expensive, use of address space identifiers greatly increases the performance of the microprocessor. This is the reason why both Intel and AMD (starting with the Nehalem VT-X [23] and SVM [24] respectively) implemented a similar feature, where the address space tag is built in the TLB and dedicated hardware checks tags for validity. This significantly increased the performance of the x86 architecture, as TLBs are designed to operate completely in hardware, with extremely low latency.

The trojan itself is a modification of the interconnection (through wire sizing) between the memory elements storing the next address space identifier, and the location where the comparison occurs, as shown in Fig. 2. The modification is inserted at the most significant bit of the address space number, introducing extra delay and generating delay faults under certain conditions.

Given the maliciously inserted delay, the most significant bit of the address space identifier may not capture the intended value during process context switch and will end up corrupted. For example, when the process to be executed has an identifier of 128 (`10000000`, given 8-bit address space identifiers), then a delay fault will convert the identifier to 0 (`00000000`; process ID 0 is reserved for kernel processes). Therefore, the next instruction of the modification-aware process will use a different address space than the one allocated by the operating system (Section V-A.3 discusses the properties and potential limitations of the attack). Thus, if the next instruction is a carefully crafted memory store that modifies the kernel structures, then a modification-aware process can access and/or write to unauthorized memory address space.

2) ATTACK PAYLOAD

Once the address space identifier is corrupted, the attacker can utilize different options in order to gain escalated privileges. In this work, we modify the the operating system-specific special kernel functions (called Privileged Architecture Library code - PALcode in the Alpha instruction set architecture).

Alpha PALcode provides a hardware abstraction layer that can be used to implement hardware special functions, such as [25]:

- Handling of complex sequencing instructions
- Translation lookaside buffer management (flush/load)
- Interrupt and exception dispatching etc.

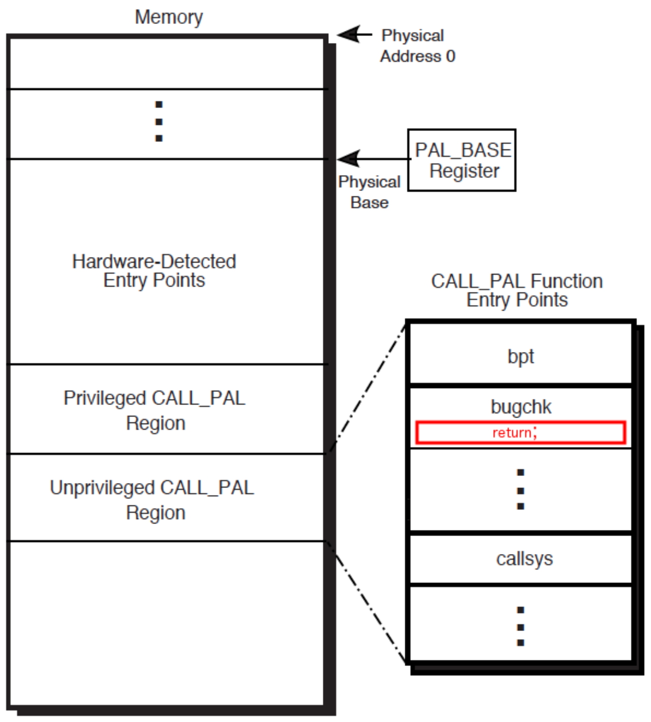

Memory
Physical Address 0
PAL_BASE Register
Physical Base
Hardware-Detected Entry Points
Privileged CALL_PAL Region
Unprivileged CALL_PAL Region
CALL_PAL Function Entry Points
bpt
bugchk
return;
callsys

FIGURE 5. **Alpha 21264 PALcode entry points into memory.**

The Alpha architecture lets these functions be implemented in standard machine code, that is resident in the data segment of memory. Fig. 5 presents the physical memory layout of the machine with PALcode installed in a location defined by the PAL_BASE register. This register is written by the operating system, and is known a priori for each operating system.

Because PALcode is resident in main memory, not all physical memory is available to the operating system code.

Since the PALcode resides in main memory, as soon as the attacking process can access the kernel memory address space, it can overwrite part of the PALcode. As only one instruction can be executed in the foreign address space (while the delay fault is still effective), a potential attack mechanism is to abruptly return to the user process while still in kernel mode. This is presented on the right hand side of Fig. 5: The attacking process overwrites an instruction belonging to the execution area of the PALcode instruction ''bugchk''; the new instruction is a simple ''return''. Therefore, when the ''bugchk'' instruction is later executed by the malicious process, the system will return to the user code, still running in PALcode environment privileges (in non-malicious code, a special instruction is used to return from PALcode). The PALcode environment has the most privileges in the system, as it has complete control of the machine state and allows all functions of the machine to be controlled. Thus, after the malicious instruction is installed after the address space identifier corruption, a simple ''bugchk'' instruction call suffices to gain escalated privileges.

The ''bugchk'' instruction was specifically chosen for two reasons:

- It is an unprivileged CALL_PAL instruction, therefore it can be invoked by any process, but runs with elevated privileges.
- It is only used for program debugging, so it will not be executed during normal system operation. Any software attempting to execute it will receive an invalid response.

3) ATTACK PROPERTIES

Feasibility: In order to apply this modification, a delay needs to be added in a specific wire. Since this can be as easy as modifying the width of a wire or changing the size of a transistor, the feasibility of the presented modification is *high*.
Controllability: There are a number of conditions that need to be satisfied in order to be able to successfully deliver a privilege escalation attack through the added TLB delay. Specifically:

1) The attacking process must receive a specific process ID (*in this case 128*).
2) When the address space identifier is corrupted, the ''new'' identifier must belong to a process with escalated privileges (*in this case 0*).
3) Context switch should occur directly before that one instruction that will attack the system, in order for the delay to be effective.

Therefore, due to the high number of conditions needed to deploy the attack, the controllability of the modification is classified as *low*.
Visibility: There are cases where this modification can have a catastrophic impact on the microprocessor execution. Modification-unaware processes that access the memory directly after context switch may end up with corrupted memory contents. However, this is rare, as context switch usually happens during I/O access and in modern operating systems process usually do not use the whole quantum given by the scheduler [26]. Indeed, experimental results presented in Section VII show that the workload used was not corrupted. However, as this could eventually lead to problems in other configurations and operating systems, on-going work explores potential masking the effect of different address space identifiers in modification-unaware processes. Due to the possibility of this case and the fact that it has never appeared in our extensive simulations, the modification visibility is classified as *medium*.
Potential: Since the presented payload manages to provide supervisor privileges to the malicious user process whenever the PALcode command 'bugchk' is executed, the attacker can execute unauthorized code at will and access system resources. Therefore, the potential of this modification is *very high*.

B. OPENRISC 1200

1) ATTACK DESCRIPTION

In the case of the OpenRISC processor, the trojan is inserted in the *chip select signal*, allowing privileged access to the ''special purpose registers'' (SPRs) of the microprocessor. These special registers control a wide range of functions, including CPU status and control, MMU configuration, TLB and cache entries, etc. [27]. Special assembly instructions can be used to read or write to these registers. Since only the operating system kernel should be allowed to access the special purpose registers, however, the OpenRISC implementations include provisions to check if the processor runs in supervisor mode, before the corresponding request goes through.

In this work, we have identified a single point of failure that renders the special purpose register protection vulnerable to zero overhead modifications. A simplified abstraction of the supervisor mode protection mechanism that controls access to the special register file is presented in Fig. 6. This abstraction shows that the ''chip select'' signal is activated if the ''read'' is excited, or if both the ''write'' and ''supervisor mode'' signals are excited.

The zero overhead trojan applicable in this case is a bridging fault between the ''write'' and ''supervisor mode'' signals, part of the security mechanism described previously. The bridging fault, essentially, ''shorts'' together the two wires under some resistance and allows one signal to dominate the other with some probability. In practice, bridging faults with resistance under 500 Ohms are not uncommon [28]. A schematic and an electronic microscope image are presented in Fig. 7.

Given the maliciously inserted bridge, the output of the SPR ''chip select'' signal is no longer controlled in a deterministic manner by the ''supervisor mode'' signal, and in practice only the ''read'' and ''write'' signals are effective (Fig. 6). As a result, in the case the ''write'' signal is excited by a process running with simple user privileges,

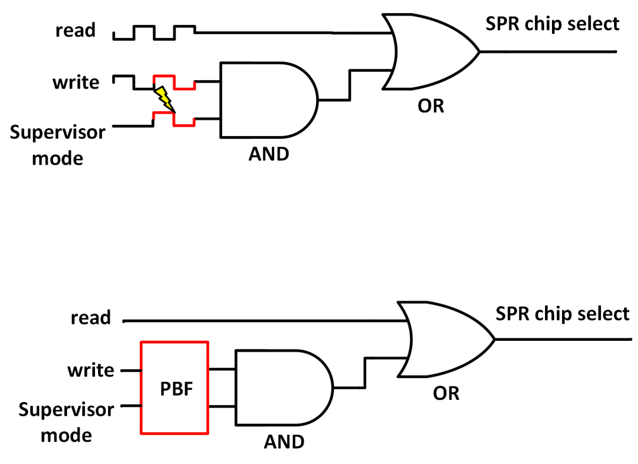


FIGURE 6. **Abstract representation of "chip select" used to protect access to special registers of the OpenRISC (such as TLB entries). The bridge fault is simulated with a Primitive Bridge Function (PBF).**

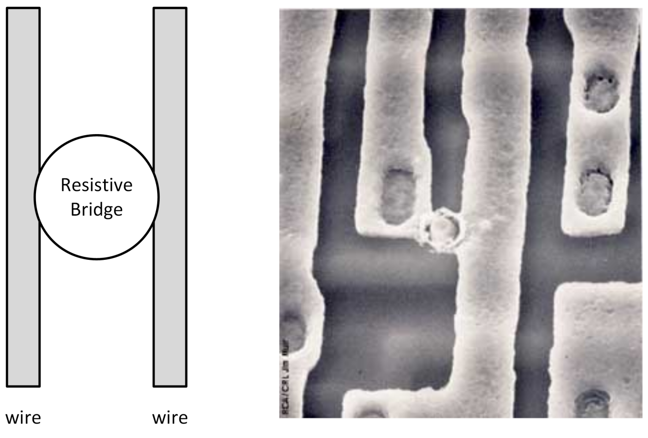


FIGURE 7. **Resistive Bridging Faults: Schematic and microscope image from [29].**

that process will be able to write in the SPRs, effectively bypassing the security controls in place. It should be noted that, in order for the malicious bridging fault to be effective, the "write" signal needs to dominate the "supervisor mode" signal: This can be achieved by exciting the "write" signal in succession, causing high frequency transitions in the bridge that eventually excite the "supervisor mode" signal.

As soon as the bridging fault is activated, the "chip select" signal is enabled and the user process that carefully excited the "write" signal will be able to modify any one of the special purpose registers in the processor. A potential payload of this attack would be to overwrite the SPR that stores the supervisor bit value, so that subsequent attacks from the malicious process will always be successful.

The fault can be inserted by the attacker on the photomask used during the lithography stage. With some effort, the attacker will be able to insert a bridging fault with medium to high resistance, in order to make the fault detectable only under very specific conditions (namely the high switching frequency of the "write" signal, as well as environmental conditions affecting bridge resistance).

2) ATTACK PAYLOAD

Once the "chip select" signal is corrupted, the attacker can overwrite any SPR. As an intermediate step, and in order to decrease the number of subsequent tries, the attacker may overwrite the SPR that maintains the current supervisor mode bit for the malicious process context.

In the demonstrated modification, we use the bridging fault in order to modify one TLB entry that is stored in the special purpose register file: An artificial, but still "valid", TLB row, is generated. This row matches a virtual address of the attacking user process with a carefully selected physical address. Furthermore, the attacker is also able to modify the access permissions that discriminate user and kernel processes, ultimately allowing access to different memory address spaces (either of the kernel or of another process). During the deployed attack, the attacking process executes an infinite loop that continuously writes and reads from a TLB entry. The high frequency of writes and reads, excite the "write" signal described in Fig. 6 and activate the bridging fault; the write operation to the TLB is eventually successful and the process controls the corresponding physical address in memory.

3) ATTACK PROPERTIES

Feasibility: In order to implement this trojan, a resistive bridging fault needs to be added between two wires. This can be achieved by the attacker with small effort, by modifying the photomask during the lithography step of the process (see Fig. 7). Therefore, the feasibility of the attack is *high*.
Controllability: As soon as the bridging fault is inserted in the microprocessor, the attacker can launch an attack with high probability, since such an attack is essentially controlled by generating a high frequency succession of read and write operations. Most of these operations will be unsuccessful, but eventually a bridging fault will be excited and cause the "write" signal to dominate the "supervisor mode" signal (as in Fig. 6). The *voting model* for bridging faults mandates that the driving gate whose network is capable of providing (or draining) more current will ultimately determine the state of the capacitive load and the logical values of the driven gates [30]. Thus, in order to satisfy this condition, the "write" signal needs to be excited at a high frequency. Another condition that also affects the probability of success is the bridging resistance itself, which is partially affected by environmental conditions (the higher the resistance, the more difficult will be to dominate the adjacent wire).

Based on the above, the probability of success is analogous to the frequency that one signal is excited with continuous high and low inputs; this high frequency pattern will eventually activate the bridge and the attack will be successful. Because the inputs can be directly controlled by the attacker through carefully crafted code, we score the controllability of this attack as *high*.

Visibility: While the attack targets the malicious switching of the "supervisor mode" signal through "write" switching, the opposite effect (i.e., "write" switching when "supervisor mode" toggles) may appear as well. In that case, there is a chance that a faulty value will be written to the special register file because of a latched "address" and "write enable" signal, eventually causing a catastrophic failure. As the bridging fault, however, is carefully injected and the resistance of the bridge is not very low, the fault is activated only when a strong signal is driven to one of the two adjacent wires coupled together due to the fault. A signal excited with very high frequency is stronger than a signal not excited often. This is why this attack is only activated when rapidly exciting the "write" signal in Fig. 6. Experimental results showed that the "supervisor mode" signal is not toggling with high frequency, rendering the modification rarely visible to the user. In practice, the FPGA running a full linux distribution did not crash in any of the experiments. Thus, visibility is conservatively classified as *medium*.

Potential: Because the payload allows the attacker to bypass the supervisor mode restriction and access the SPRs, the security impact to the microprocessor is the highest. The attacker is able to control physical memory as well as the operating system kernel. Therefore, the potential of this malicious modification is *very high*.

The summary of all the properties of the two demonstrated attacks appear in Table 1.

TABLE 1. **Summary of the properties of the presented attacks.**

Attack	ASN corruption	SPR corruption
Platform	Alpha 21264	OpenRISC 1200
Feasibility	Very high	High
Controllability	Low	High
Visibility	Medium	Medium
Potential	Very High	Very High

VI. EXPERIMENTAL SETUP

Ideally, the presented attacks should be demonstrated directly on an ASIC during fabrication process. As access to these resources is extremely limited, we emulate the behavior of the proposed modifications in order to demonstrate the feasibility of the attacks. Specifically, we use a full-system simulator for the Alpha 21264, and an FPGA platform for the OpenRISC 1200.

A. MICROPROCESSOR MODELS

1) ALPHA 21264

Gem5 [31] is the functional simulator used to evaluate the presented attack. Gem5 is a modular, discrete event driven computer system simulator platform written in C++. Gem5 can execute an unmodified Linux kernel, with full device support, at very high speeds.

The Linux kernel used in this study is the 2.4 version, optimized for the alphaev67 platform (codename for the Alpha 21264A, a smaller version of the Alpha 21264). Without loss of generality, the Linux kernel source code has been modified to match the ASN to the process IDs. This modification allows the attacking process to identify their ASN by reading the OS-assigned process ID. Ongoing work explores avoiding this modification by maintaining an "ASN to process ID" translation table.

Similar to the BIOS functionality in the x86 architecture, a firmware layer is needed to correctly initialize the machine state. Therefore, we use the Alpha Linux Miniloader (MILO), in order to load the Linux kernel. MILO builds the page tables, turns on virtual addressing, installs PALcode and initializes the kernel.

2) OPENRISC 1200

The Digilent Atlys FPGA is used to implement the `ORPSoCv2` system on chip, which incorporates the OpenRISC 1200 processor core along with all necessary peripherals. The OpenRISC processor is an open source Verilog design that can be synthesized on FPGAs as well as ASIC designs [22].

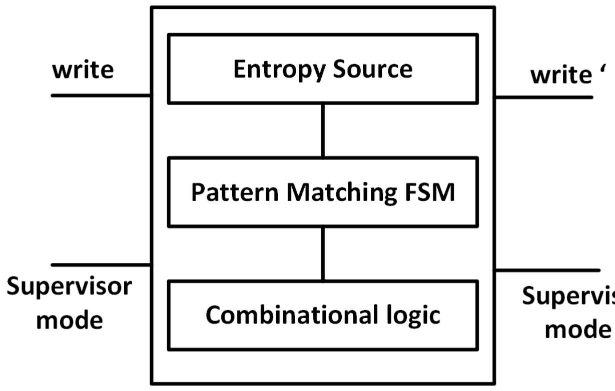
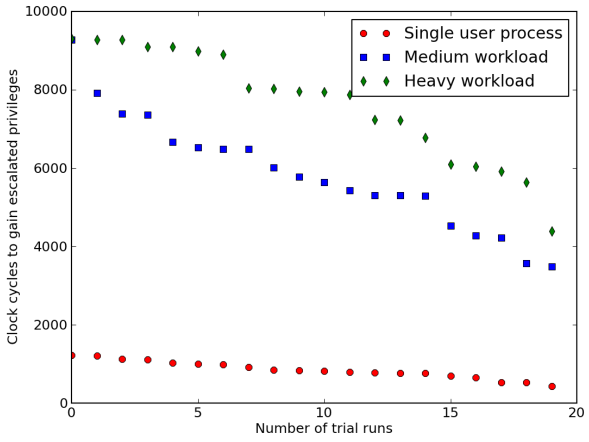


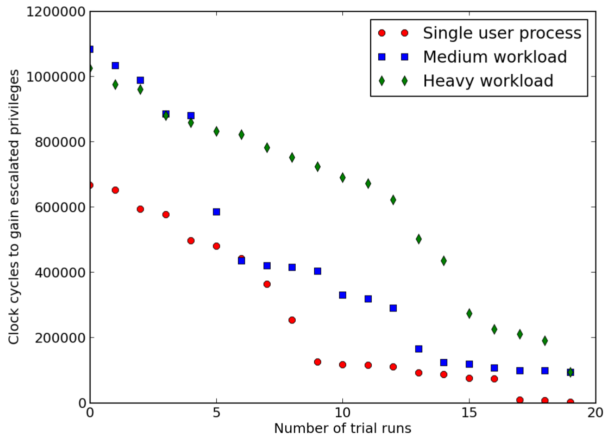
FIGURE 8. **Primitive Bridge Function used to emulate the fabrication fault in the FPGA.**

The fabrication attack is emulated using a *Primitive Bridge Function*; a block diagram of this function is presented in Fig. 8. Historically, bridging faults in TTL logic (i.e. Transistor to Transistor Logic) are simply emulated using an OR logic gate [32], [33]. In CMOS, however, this model is not effective and special Primitive Bridging Functions implementing a voting model are required. As these models are specific to the underlying technology [34] and cannot be used in a more abstract FPGA implementation, a technology independent approach is used in this study.

As expected, the bridge fault model *should generate corrupted inputs* to the AND logic gate with non-negligible probability (Fig. 6). The model implements a fault which is dominated by the signal with the highest frequency of excitements (i.e. the "write" signal in Fig. 8), and this signal remains correct even in the presence of the fault. On the other hand,



(a)



(b)

FIGURE 9. Clock cycles to gain escalated privileges for synthetic benchmarks (random sample of 20 trials). (a) Alpha 21264. (b) OpenRISC 1200.

the second signal (i.e. "Supervisor mode") occasionally will be incorrect (i.e. flipped). The implemented Primitive Bridge Function (PBF) utilizes a source of entropy (i.e. pseudo-randomness) in an effort to model (a) the probability of high frequency signal excitements dominating the adjacent signal, (b) the probability that the attacker injected the fault properly, (c) environmental conditions that facilitate the attack, as well as (d) for all other unknown variables. In addition, the PBF incorporates a pattern matching FSM that recognizes when the rapidly excited signal is present, along with the necessary combinational logic.

The OpenRISC hardware was compiled using the tools provided with the OpenRISC/ORPSOCv2 release, as well as the Xilinx ISE 14.4 framework. For the operating system and necessary software, the latest OpenRISC Linux kernel 3.8 was used, along with the entire GNU toolchain cross-compiled for the OpenRISC ISA. Both the compiled kernel and the user programs were merged into bitstream files that were used to program the FPGA. All attacks and experiments were launched from inside the Linux *sh* shell running live on the FPGA.

B. WORKLOAD

Two different types of workload are utilized for demonstrating the feasibility of the presented privilege escalation attacks:

- Synthetic workload, that consists of infinite loops of carefully crafted code in order to generate the right conditions to trigger the trojan and gain control of the system.
- Real-world benchmarks (such as SPEC2000 and Polarssl), where the attacking code described previously is injected in random locations of the algorithm. Therefore, while the benchmarks executes their useful workload, a continuous privilege escalation attack is taking place in the background. Twelve different benchmarks are used in this study, namely bzip2, mcf, gap, gzip, cc, parser (from SPEC2000), as well as AES-256-CBC, DES-CBC, CAMELLIA-256-CBC, MD5, SHA1 and SHA256 (from polarssl). The cryptographic benchmarks help demonstrate attacks where the privilege escalation assists leaking the cryptographic keys, by gaining supervisor privileges and bypassing inter-process memory protections.

  The SPEC benchmarks, representing typical high-performance computing workload, were used for the Alpha 21264 analysis. The Polarssl benchmarks, representing typical embedded computing workload, were used for the high-throughput OpenRISC microprocessor.

VII. RESULTS AND DISCUSSION

This section discusses the timing perturbations of the presented attacks for the synthetic and the real-world benchmarks (i.e. SPEC2000, polarssl). For each experiment *a random sample* of 20 trials was used each time. The number of trials is sufficient since *all trials* are successful within a small number of cycles, (and the attacker needs to succeed *at least once*, and not *every time*).

A. TIME TO GAIN PRIVILEGES

Alpha: Fig. 9 presents the number of clock cycles needed for synthetic benchmarks to get escalated privileges, for 20 different trials. For clarity, the trials are sorted in decreasing order of clock cycles. As a reminder, a synthetic benchmark only contains the necessary code to alter the PAL-code code and successfully execute a "bugchk" instruction. Fig. 9 shows that if the benchmark is the only (user) process in the system, then a privilege escalation attack can take place within a few thousand cycles. As more processes are added to the system competing for the CPU (10 processes for medium workload and 100 for heavy workload),

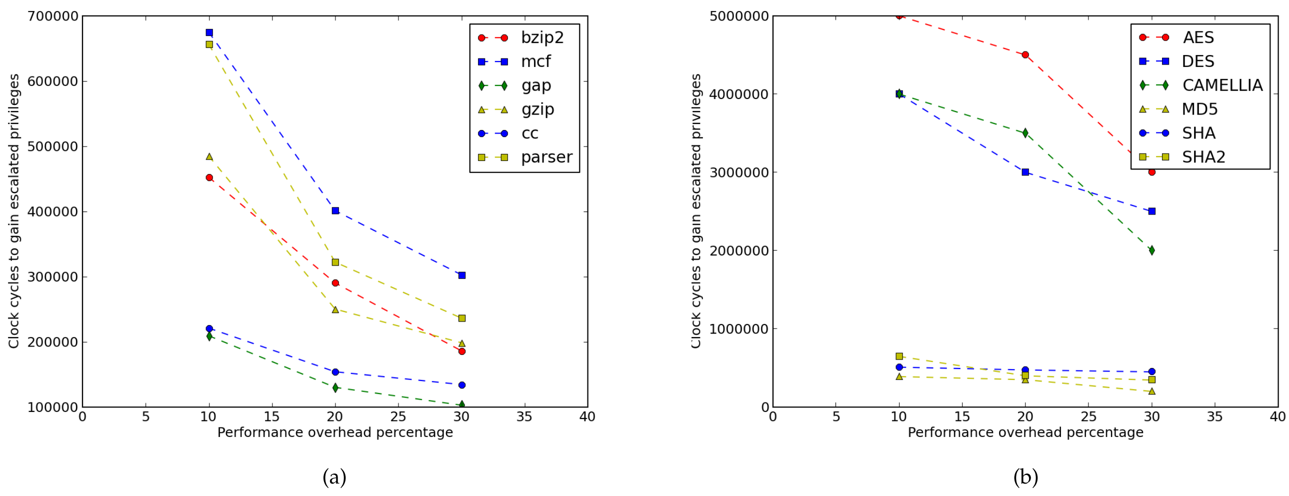


FIGURE 10. **Clock cycles to gain escalated privileges for SPEC2000 and Polarssl benchmarks respectively (average of random sample of 20 trials), for given performance overhead (i.e. overhead from execution of malicious instructions). (a) Alpha 21264. (b) OpenRISC 1200.**

the number of clock cycles needed to successfully deliver the attack greatly increases, as the process runs less often in the microprocessor. An interesting observation though, is that adding an order of magnitude more processes (100 instead of 10) does not significantly increase the number of clock cycles before privilege escalation. This is attributed to the fact that with both medium and heavy workloads there is very intensive switching, so the synthetic workload has many windows of opportunity to deliver the attack.

OpenRISC: Similarly to the Alpha synthetic experiments described above, Fig. 9(b) shows 20 trials to elevate privileges (sorted in descending order for clarity) for varying system load. The results for the OpenRISC indicate that as the process CPU quota decreases, more cycles are required to succeed in the attack, as expected. A difference between the Alpha 21264 and the OpenRISC attacks, is that for the latter, the higher the CPU utilization, the higher the probability the malicious process will be able to trigger the fault and escalate privileges. The Alpha attack relies more on context switching. Nevertheless, in all experiments, the malicious process was able to escalate privileges within about a million cycles, which indicates that the attack is indeed applicable.

B. SPEC AND POLARSSL BENCHMARKS PERFORMANCE OVERHEAD

The number of extra instructions maliciously added to an existing benchmark affect how many clock cycles are required to receive escalated privileges. Fig. 10 presents the average number of clock cycles needed to deliver the attack for different number of *performance overhead* (i.e., overhead from the dynamic execution of malicious instructions), for 20 trials. Performance overheads of approximately 10%, 20% and 30% are examined.

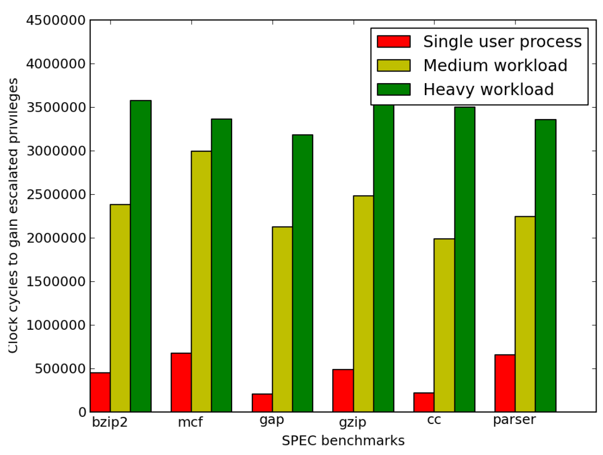
Alpha: As expected, there is a linear decrease of the time the attack successfully takes place for increasing performance overhead, for all given benchmarks. It should be noted that the average time-to-attack figures vary for different benchmarks; this is attributed to different numbers of I/O accesses. Since a few thousand cycles is a negligible fraction of time in real-world attack deployment, the performance overhead of the SPEC benchmarks can be sustained in very small percentages.

OpenRISC: The OpenRISC benchmarks also demonstrate that a real-world attack is effective within a few million cycles, depending on the performance overhead applied to the underlying benchmark. The cycles required to successfully elevate privileges also depends on how demanding the benchmark is in terms of resources. This explains why the hash algorithms require about 0.5 million cycles while the heavy block ciphers require between 2 and 5 million cycles. This set of results demonstrates that the presented attack on the OpenRISC processor is applicable in heavy duty applications (such as a block cipher).

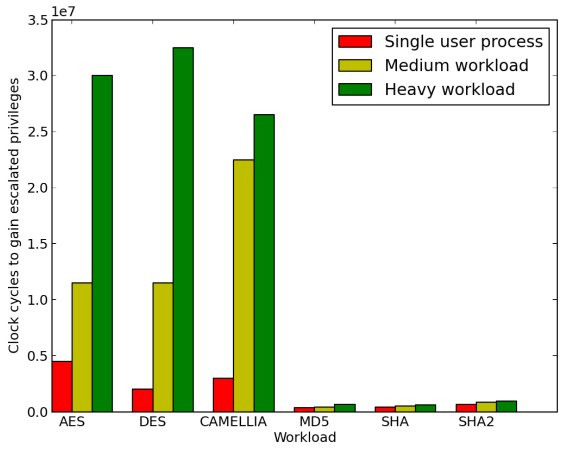
C. SYSTEM LOAD IMPACT ON TIME-TO-ATTACK

The final set of results discusses the behavior of the attack in the presence of resource-competing processes. Fig. 11 presents the number of clock cycles required to gain escalated privileges, for the polarssl and SPEC2000 benchmarks. The results are the average of 20 trial runs, for 10% performance overhead.

Alpha: The first observation is that when the SPEC2000 benchmark is the only user process in the system, then a privilege escalation attack can take place within a million clock cycles. However, as more processes are added, this number increases dramatically: More processes mean less CPU time for the benchmark. Furthermore, the benchmark individual I/O demand is forcing them to yield to other processes,



(a)



(b)

FIGURE 11. Clock cycles to gain escalated privileges for SPEC2000 and Polarssl benchmarks respectively (average of random sample of 20 trials), for different system activity. (a) Alpha 21264. (b) OpenRISC 1200.

decreasing the probability for all 3 conditions presented in Section V-A.3 to be satisfied at the same time. Therefore, the attack may require several million cycles to successfully deliver the payload. In our experiments, the use of typical input for the SPEC2000 benchmarks provided enough time for the attack to take place. However, the attack might not go through given small workloads.

OpenRISC: For the OpenRISC, the results are similar to the performance overhead results, indicating that the higher the system load, the smaller the probability the bridging fault will be excited, allowing escalation of privileges. A significant observation is that even though the system runs with a potential bridging fault, the operating system did not crash even after hours of continuous operation. This suggests that the assumptions about the activation conditions of the fault (i.e. successive write attempts to SPRs) are extremely unusual under normal operation and only the attacker may trigger such conditions.

D. DETECTING FABRICATION ATTACKS

The presented attacks demonstrate that even minor modifications, when properly exploited, can give extreme power to the attacker. This emphasizes the need for trojan detection techniques. In [35], the authors suggest a method for reducing the problem of comparing 2 designs to a boolean satisfiability problem, while in [36] the authors suggest an entire framework to formally verify IP hardware properties. In addition, the authors of [37] propose an equivalence checking method paired with ATPG to identify the suspicious signals.

At the side channel analysis frontier, the authors of [38] suggest a method for detecting malicious modifications using non-destructive measurements for power and delay; in [39] the authors employ gate level characterization techniques, while [40] also discusses a detection method using path delay measurements. From the above methods, those that do not require access to the RTL and can be applied to the physical design directly, can be a good starting point in detecting the rogue TLB signal in the Alpha 21264, described in this work, as well as the proposed bridging trojan in the OpenRISC processor.

VIII. CONCLUSION

In this paper, we demonstrate zero overhead microprocessor modifications that can be exploited in order to escalate system privileges. These modifications can be effectively applied during the fabrication stage of the chip, as their application requires minimal invasion. Two different microprocessors are used to demonstrate these malicious alterations: (i) The Alpha 21264, resembling commercial high-performance microprocessors, and (ii) the OpenRISC 1200, representing embedded processors. The attacks are presented and characterized using newly defined properties, namely: feasibility, controllability, visibility and potential. Experimental results corroborate that the proposed attacks are indeed applicable, despite heavy system load, and the attacker can exploit them in order to escalate system privileges. Furthermore, due to the fact that the modifications have no overhead and they manifest only when the attacker triggers specific conditions (no system crash occured during extensive microprocessor simulation), discovering these malicious modification can be a very tedious process.

REFERENCES

[1] R. N. Charette. (Accessed: 2 Nov. 2013). *This Car Runs on Code* [Online]. Available: http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code/0#

[2] *Defense Science Board Task Force on High Performance Microchip Supply*, U.S. Defense Sci. Board, Washington, DC, USA, 2005.

[3] S. Skorobogatov and C. Woods, "Breakthrough silicon scanning discovers backdoor in military chip," in *Cryptographic Hardware and Embedded Systems* (Lecture Notes Comput. Sci.), vol. 7428, E. Prouff and P. Schaumont, Eds. Berlin Heidelberg, Germany: Springer-Verlag, 2012, pp. 23–40.

[4] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Trustworthy hardware: Identifying and classifying hardware Trojans," *IEEE Comput.*, vol. 43, no. 10, pp. 39–46, Oct. 2010.

[5] Y. Jin and Y. Makris, ''Hardware Trojans in wireless cryptographic ICs,'' *IEEE Design Test Comput.*, vol. 27, no. 1, pp. 26–35, Jan./Feb. 2010.
[6] M. Tehranipoor, H. Salmani, X. Zhang, X. Wang, R. Karri, J. Rajendran, *et al.*, ''Trustworthy hardware: Trojan detection and design-for-trust challenges,'' *IEEE Comput.*, vol. 44, no. 7, pp. 66–74, Jul. 2011.
[7] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, ''Security analysis of logic obfuscation,'' in *Proc. ACD/EDAC/IEEE Design Autom. Conf.*, 2012, pp. 83–89.
[8] Y. Alkabani and F. Koushanfar, ''Extended Abstract: Designer's hardware Trojan horse,'' in *Proc. IEEE Int. Workshop Hardw.-Oriented Security Trust*, Jun. 2008, pp. 82–83.
[9] Y. Jin, M. Maniatakos, and Y. Makris, ''Exposing vulnerabilities of untrusted computing platforms,'' in *Proc. IEEE 30th Int. Conf. Comput. Design*, Sep./Oct. 2009, pp. 91–96.
[10] D. Hély, M. Augagneur, Y. Clauzel, and J. Dubeuf, ''Malicious key emission via hardware Trojan against encryption system,'' in *Proc. IEEE 30th Int. Conf. Comput. Design*, Sep./Oct. 2012, pp. 127–130.
[11] T. Reece, D. B. Limbrick, X. Wang, B. T. Kiddie, and W. H. Robinson, ''Stealth assessment of hardware Trojans in a microcontroller,'' in *Proc. IEEE 30th Int. Conf. Comput. Design*, Sep./Oct. 2012, pp. 139–142.
[12] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, ''Trojan detection using IC fingerprinting,'' in *Proc. IEEE Symp. Security Privacy*, May 2007, pp. 296–310.
[13] S. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou, ''Designing and implementing malicious hardware,'' in *Proc. 1st USENIX Workshop Large-Scale Exploits Emergent Threats*, 2008, pp. 1–8.
[14] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*. San Mateo, CA, USA: Morgan Kaufmann, 2009.
[15] N. Provos, M. Friedl, and P. Honeyman, ''Preventing privilege escalation,'' in *Proc. 12th USENIX Security Symp.*, vol. 12. 2003, pp. 231–242.
[16] L. Davi, A. Dmitrienko, A. Sadeghi, and M. Winandy, ''Privilege escalation attacks on android,'' in *Information Security*. New York, NY, USA: Springer-Verlag, 2011, pp. 346–360.
[17] J. Uffenbeck, *The 80×86 Family: Design, Programming, and Interfacing*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1997.
[18] Digital Equipment Corporation, ''Alpha 21264 microprocessor hardware reference manual,'' Compaq, Maynard, MA, USA, Tech. Rep. EC-RJRZA-TE, 1999.
[19] H. Shacham, ''The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86),'' in *Proc. ACM Conf. Comput. Commun. Security*, 2007, pp. 552–561.
[20] M. Maniatakos, N. Karimi, C. Tirumurti, A. Jas, and Y. Makris, ''Instruction-level impact analysis of low-level faults in a modern microprocessor controller,'' *IEEE Trans. Comput.*, vol. 60, no. 9, pp. 1260–1273, Sep. 2011.
[21] N. J. Wang, J. Quek, T. M. Rafacz, and S. J. Patel, ''Characterizing the effects of transient faults on a highperformance processor pipeline,'' in *Proc. Int. Conf. Dependable Syst. Netw.*, 2004, pp. 61–70.
[22] OpenCores.org. (Accessed: 2 Nov. 2013). *OR1200 OpenRISC Processor* [Online]. Available: http://opencores.org/or1k/OR1200_OpenRISC_Processor
[23] G. Neiger, A. Santoni, F. Leung, D. Rodgers, and R. Uhlig, ''Intel virtualization technology: Hardware support for efficient processor virtualization,'' *Intel Technol. J.*, vol. 10, no. 3, pp. 167–177, 2006.
[24] Advanced Micro Devices, ''AMD secure virtual machine architecture reference manual,'' AMD Corporation, Sunnyvale, CA, USA, Tech. Rep. 33047, 1999.
[25] R. Sites, *Alpha Architecture Reference Manual*. Bedford, MA, USA: Digital, 1998.
[26] J. Aas, *Understanding the Linux 2.6.8.1 CPU Scheduler*. Silicon Graphics Inc., Fremont, CA, USA, 2005
[27] OpenCores.org, *Openrisc 1000 Architecture Manual*, OpenCores.org, Dec. 2012.
[28] M. Renovell, P. Huc, and Y. Bertrand, ''The concept of resistance interval: A new parametric model for realistic resistive bridging fault,'' in *Proc. 13th IEEE VLSI Test Symp.*, Apr./May 1995, pp. 184–189.
[29] Thomson Lab Services, Indianapolis, IL, USA. (Accessed: 2 Nov. 2013) *Scanning Electron Microscope (SEM)* [Online]. Available: http://labservices.technicolor.com/www.thomsonlabs.com/EN/Home/TLS/Material/Scanning%2bElectron%2bMicroscope%2bSEM.htm
[30] P. Engelke, ''Resistive bridging faults—Defect-oriented modeling and efficient testing,'' Ph.D. dissertation, Faculty of Technol., Albert-Ludwig-Univ., Freiburg, Germany, 2009.
[31] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, *et al.*, ''The GEM5 simulator,'' *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.
[32] B. Chess, C. Roth, and T. Larrabee, ''On evaluating competing bridge fault models for CMOS ICs,'' in *Proc. 12th IEEE VLSI Test Symp.*, Apr. 1994, pp. 446–451.
[33] C. Roth, ''Simulation and test pattern generation for bridge faults in CMOS ICs,'' M.S. dissertation, Computer Eng. Dept., Univ. California, Santa Cruz, CA, USA, 1994.
[34] B. Chess and T. Larrabee, ''Logic testing of bridging faults in CMOS integrated circuits,'' *IEEE Trans. Comput.*, vol. 47, no. 3, pp. 338–345, Mar. 1998.
[35] G. Shrestha and M. S. Hsiao, ''Ensuring trust of third-party hardware design with constrained sequential equivalence checking,'' in *Proc. IEEE Conf. Technol. Homeland Security*, Nov. 2012, pp. 7–12.
[36] T. Reece, D. B. Limbrick, and W. H. Robinson, ''Design comparison to identify malicious hardware in external intellectual property,'' in *Proc. IEEE 10th Int. Conf. Trust, Security Privacy Comput. Commun.*, Nov. 2011, pp. 639–646.
[37] E. Love, Y. Jin, and Y. Makris, ''Proof-carrying hardware intellectual property: A pathway to trusted module acquisition,'' *IEEE Trans. Inf. Forensics Security*, vol. 7, no. 1, pp. 25–40, Feb. 2012.
[38] S. Wei and M. Potkonjak, ''Scalable hardware Trojan diagnosis,'' *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 6, pp. 1049–1057, Jun. 2012.
[39] M. Potkonjak, A. Nahapetian, M. Nelson, and T. Massey, ''Hardware Trojan horse detection using gate-level characterization,'' in *Proc. 46th ACM/IEEE DAC*, Jul. 2009, pp. 688–693.
[40] Y. Jin and Y. Makris, ''Hardware Trojan detection using path delay fingerprint,'' in *Proc. IEEE Int. Workshop HOST*, Jun. 2008, pp. 51–57.

NEKTARIOS GEORGIOS TSOUTSOS received the 5-year Diploma degree in electrical and computer engineering from the National Technical University of Athens, Greece, in 2008, and the M.Sc. degree in computer engineering from Columbia University, New York, in 2010. After a brief career as a security professional, he is currently pursuing the Ph.D. degree in computer science from New York University. His research interests include computer security, computer architecture, forensic analysis with hardware support, and hardware Trojans.

MICHAIL MANIATAKOS is an Assistant Professor of electrical and computer engineering with New York University Abu Dhabi and a Research Assistant Professor with the NYU Polytechnic School of Engineering. He received the B.Sc. and M.Sc. degrees in computer science and embedded systems from the University of Piraeus, Greece, in 2006 and 2007, respectively, and the M.Sc., M.Phil., and Ph.D. degrees from Yale University in 2008, 2009 and 2012, respectively. He is the Director of the Modern Microprocessor Architectures Laboratory, NYU Abu Dhabi, and is the author of multiple publications in the IEEE Transactions and conference papers. His research interests include robust microprocessor architectures, hardware security, and heterogeneous microprocessor architectures.