HARDWARE ASSISTED SOFTWARE ATTACKS

Zafeiri Stamatia-Varvara

Department of Electrical and Computer Engineering
University of Thessaly

Abstract

Computer system security is now being threatened by hardware Trojans. During the manufacturing process, integrated circuits (ICs) can be subjected to these malevolent alterations, which have the potential to compromise crucial systems or steal important data. This work introduces a novel attack vector where Hardware Trojans can be utilized to support software attacks, whereas earlier research has mostly concentrated on identifying Hardware Trojans. The suggested technique entails using the Hardware Trojan's extra capability to get around security measures and inject malicious code onto the target system. The findings demonstrate that the suggested method can seriously damage the target system and is highly effective at getting beyond existing security measures like firewalls and intrusion detection systems. The results of this study highlight the necessity for ongoing research into the prevention and detection of Hardware Trojans and have significant implications for the security of computer systems and the protection of sensitive data.

Keywords: Hardware trojans, hardware, Hardware involved attacks, software attacks, Hardware assisted, software attacks.

1. Introduction

The purpose of this project is to analyze hardware-assisted software attacks and how to recognize and defend against hybrid attacks. There aren't many (exact) cases of hardware vulnerabilities or assaults that have been publicly reported. The current state of the economy significantly increases the susceptibility to Trojan horse attacks[1][2]. The IC life cycle increasingly relies on unreliable parties and entities for IC design and production procedures. The majority of current ICs must be produced in unsafe fabrication facilities due to economic considerations. Modern IC design also frequently uses electronic design automation (EDA) software tools from various vendors, outsourced design and test services, and intellectual property (IP) cores that are provided by unreliable third-party providers [3].

2. PC System Stack

A PC computer system, in its simplest form, is made up of a stack of software, namely software programs, software operating systems, and VMM Hypervisor, and a stack of hardware, namely the hardware platform and the bios, which are intimately tied to the hardware. The components of the hardware platform are the CPU, Memory, and Peripherals. Hardware components frequently include firmware capabilities as technology develops, opening the door to new kinds of software attacks.

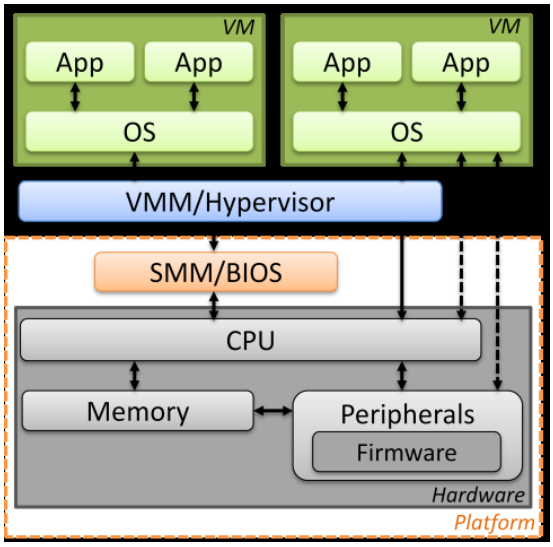

Fig.1 Layers of Hardware [4]

3. Common attack scenarios until now

The following are the most frequent software-related attack scenarios:

• A computer program that is horizontally attacking another computer program

• A privileged OS software piece is directly attacked by a non-privileged software application.

• A third party targeting the system's software directly from a distance, whether they are privileged or not.

• When the software hypervisor layer is the target of a remote assault, the "VM escapes" from the VM and enters the software hypervisor layer or peer VM software components.

Any software component with a lower level of privilege launching direct attacks on SMM [5]/BIOS [6] software or Because BIOS/SMM software only reveals related points of entry locally to the system and not in a remotely accessible manner, remote attacks targeting BIOS/SMM software are typically not taken into consideration.

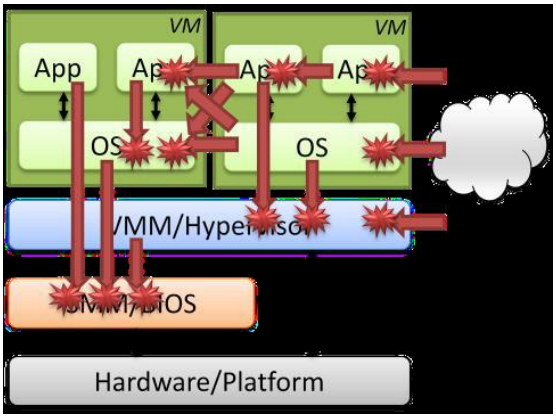

Fig. 2 Attack scenarios [4]

4. Why the change

Although the security industry is now focusing on software vulnerabilities, assaults on hardware are a possibility. The reason for this is because major software domains that affect security, like operating systems, network services, web browsers, and virtualization hypervisors, are often in a considerably better security posture now than they were previously. If you already have full access to the OS, there is no reason to move lower in the stack because it takes more effort on the part of the attacker to discover and implement a security exploit against specific targets than it did in prior generations to have the same net effect.

This indicates that major software areas that are often used by attackers quickly develop low-investment security flaws. Attackers will have to invest more money in order to find the medium-investment security vulnerabilities; as a result, the supply of those vulnerabilities will eventually run out, necessitating more money from attackers if they want to continue.

This foreshadows a forced migration into lower layers given that all the layers at and above the OS are gradually improving their security posture and are heavily used by attackers.

The realization that hardware-involved attack scenarios are under-represented in the community's current knowledge pool—particularly in terminology, attack characterization, and defense/mitigation guidance—came as researchers[7] and attackers[8] turned their attention to attacks including hardware elements.

5. Obtaining Hardware Access

Every computer system's foundation is made up of hardware, which has the potential to directly affect important system resources, indirectly access resources through sideband channels, store arbitrary software executable code that can be called automatically, and proxy data from unreliable external sources.

An attacker can access hardware in one of five ways before launching a software assault. [4]

5.1 Inadvertently allowed through by a layer of software with greater privileges

While attempting to grant regulated or limited access to hardware, a higher-privileged software layer may end up being unduly liberal, or the hardware it granted access to may have additional, unacknowledged capabilities. In other cases, the privileged software layer's hardware access functionality can be a holdover from pre-production debugging requirements.

5.2 Explicitly let through by a layer of software with greater privileges

Many hardware components, such like graphics or user-mode driver frameworks, are designed to be accessed and used by the local user, who normally operates at the application privilege layer. Especially with regard to USB devices, give the OS the freedom to delegate some device handling to user-mode programs. In each of these scenarios, the OS layer is granting the application layer access to a specific area of hardware in order to aid the management and usage of that hardware.

5.3 Specifically offered by the hardware architectural design

Direct hardware access is made possible by hardware aided virtualization technologies like Intel VT-x, allowing software to execute numerous instructions directly on the CPU without the need for OS (ring 0) intervention.

5.4 It is already believed that the attacker has access

Because the SMM/BIOS software layer is not in a strong architectural position to manage all hardware access, it must always function with the understanding that system resources can be shared with a potentially insecure OS/hypervisor layer. The ability to merely intercept a portion of hardware access and CPU instructions coming from a VM guest is another benefit of hardware assisted virtualization technologies; some VM guest hardware activities simply lack a matching VMM trap or exit that the hypervisor can use.

5.5 Physical proximity between the attacker and the system exists (Untrusted parties, hardware trojans)

The use of externally exposed hardware and different hardware tampering attacks are made possible by physical possession of or access to a PC system. For assaults that use radio hardware (such as WiFi, LTE/Wimax, Bluetooth, or GSM/cellular) as the system entry point, physical proximity is sufficient. Aside from PCs, the community has already witnessed numerous cases of gaming console hacks and embedded system "jailbreaking"[9] where direct access to the system was used to gain some type of software advantage.

Additionally, adversaries may insert Hardware Trojans or backdoors at any time during the design process. IP can be illegally pirated. [3] On the other side, a hardware Trojan is an evil alteration done to a hardware device during the production or supply chain process. The change, usually by adding a back door or even other vulnerability, enables an attacker to take over the device without authorization.

6. Hardware Trojan assisted attacks

An attack that uses a hardware Trojan like a path to carry out malicious activity is referred to as a hardware Trojan enabled attack. A hardware Trojan, for instance, might be used by an attacker to access a device's memory and then exploit that access to steal or install malware.

The alteration of system data is one of the main ways that hardware Trojans can be exploited to support software attacks. A Trojan, for instance, might be programmed to change the values of particular system variables, leading the software to act strangely. This might result in data damage, system crashes, or even the execution of malicious code.

The development of "backdoors" in the system is another potential way that hardware Trojans could be used in software attacks. These backdoors could be used to break into the system without authorization, giving a hacker access to steal confidential information or take over the system.

An attack falling under the category of "hardware-involved software attack" will typically meet the following criteria:

- Begin in a lower-privileged software/layer or be distantly close
- Rely on or be dependent upon hardware operation
- Discover a weakness in a higher-privileged software/layer or even a peer in current software/layer [4].

A significant matter is that after a hardware Trojan occurs, the attacker can use it from a lower-privileged software/layer.

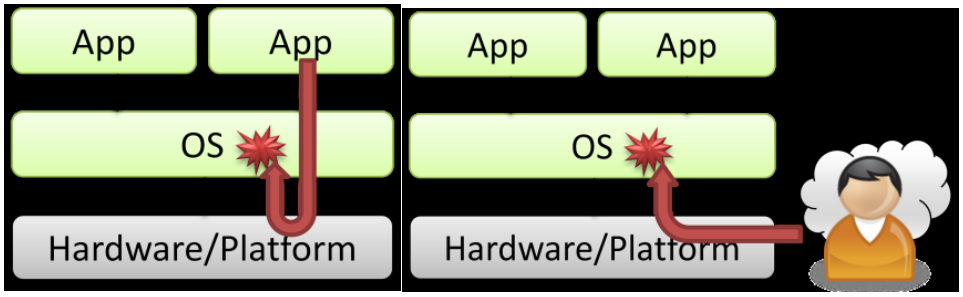

Fig. 3 Hardware assisted attacks scenarios [4]

6.1 Unsuitable Open Access to Hardware

In this attack scenario, a lower-privileged software layer is given improper general hardware access by a higher-privileged software layer (such as an OS or hypervisor). This enables less privileged software to exploit security vulnerabilities by using common hardware access techniques. Attacks like this are commonly called "confused deputy" attacks. [10] However, given that it is already known that giving lower-privileged software access to arbitrary system memory is an immediate vulnerability that doesn't require any hardware involvement, the extent of that memory access and its relevance to hardware involvement may need to be qualified or reviewed. In some circumstances, access to hardware-consumed data structures in normal memory may be relevant**.** For instance, updating the SYSENTER EIP MSR would essentially allow the program to specify any memory address that would be called with ring 0 privileges the next time the SYSENTER instruction was executed.

6.2 Unanticipated Effects of a Particular Hardware Function

The initial step in the attack preconditions is the grant or proxying of particular, limited hardware access to hardware for data from a lower-privileged software layer by a higher-privileged software layer (such as an OS or hypervisor). This is done under the assumption that just certain hardware functionality is exposed and that there are no security issues with access to that particular hardware functionality. The attack then takes advantage of a subsequent or unanticipated effect of that hardware access that was not initially taken into account when the decision to grant access was made. The DMA activities of such device can be affected by the unprivileged software layer. On platforms where they are available, using IOMMU technologies may help to lessen the impact of the assault.

6.3 Hardware Reflected Injection

When an attacking application creates specific malicious data, that data travels (without modification) through higher-privileged software stack layers into the hardware for storage; afterward, a privileged software layer (such as a driver in the OS) obtains the malicious data from the hardware and instantly operates upon, interprets, and otherwise uses the malicious data in an unsafe manner that results in a hardware reflected injection attack. Attacks using hardware-reflected injection have a greater chance of succeeding on software targets that already implicitly trust hardware-generated data.

6.4 Disruption of Hardware Privilege Access Enforcement

Typically directed at the hypervisor layer (or emulation software), this attack involves circumstances where a less-privileged software layer can force the hypervisor to carry out hardware operations (such as instruction execution) or gain access to hardware that would typically be denied to that layer due to hardware privilege enforcement, but will now be permitted because it was carried out with hypervisor privileges.

6.5 Access by an Executing Entity in Parallel

To ensure that no other concurrently running entities (including software and hardware) interfere with the security-sensitive actions taking place on that shared resource, it may be essential to quiesce[11] all executing entities when an authorized software security agent must operate on memory.

If the person performing a security-sensitive action fails to take into consideration the possibility that a parallelly running entity may be able to observe or obstruct the process, an attack may be conceivable. Using a CPU thread to keep track of a different thread's memory cache misses in order to recover data like cryptographic keys.

6.6 External Control of Hardware Device

The safety of the system may be compromised by hardware components that behave maliciously; the question is when and how this happens. Although this assault scenario acknowledges that there may be less privileged attacks coming from outside the system, we have been concentrating on attacks coming from software on the target system. DMA over a Firewire port enables system memory reading and

writing for any Firewire-connected device. During a system restart, a reprogrammed keyboard could be used to simulate human input by pressing keys that modify the BIOS setup. This attack scenario can also occur during radio attacks.

6.7 Incorrect Hardware Use

This kind of attack scenario acts as a catch-all to address use and configuration mistakes made by privileged software layers to make use of hardware in a way that will ensure a secure and privileged environment.

7. How to protect from hardware Trojans

As was already established, these attacks target ICs that have been maliciously modified during design or fabrication in an unreliable design house or foundry using unreliable people, design tools, or components. Such alterations can cause an IC's functionality to behave in an undesirable way or create backdoors or covert channels that allow the leakage of private data[1][12]. These malicious circuits, which target the operating system (OS) of a computer, have been dubbed "hardware Trojans" in the popular press. Ideally, pre-silicon verification/simulation or post-silicon testing should be able to identify any unwanted modifications applied to an IC. Contrary to design flaws, which result from flaws introduced during the manufacturing process, hardware Trojans are implanted knowingly by an adversary to carry out a specific harmful function. [13]

The fact that the malicious code is contained into the hardware itself makes it difficult to identify and prevent hardware Trojan-assisted software attacks. Due of this, it is challenging for conventional security measures, like antivirus software, to identify and eliminate the threat. Researchers are investigating several methods for identifying and avoiding hardware Trojans to address this issue. Utilizing hardware-based security techniques, such as secure boot and secure enclaves, is one strategy. These precautions can aid in preventing the initial installation of malicious code into the hardware and can also be used to find and eliminate any Trojans that may have already been inserted [14].

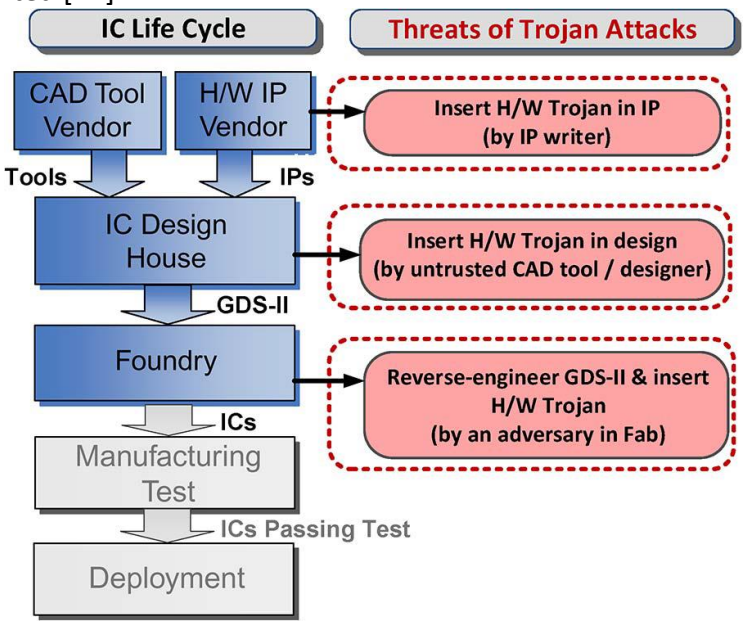

Fig. 4 Threats of Trojan Attacks [13]

The employment of software-based procedures, such as formal testing and verification, provides an alternative strategy. These methods can be used to thoroughly examine the software currently being used on a device, allowing for the detection and elimination of any harmful malware.

7.1 How to Prevent Trojan Insertion

Obfuscation-based methods and layout-filler approaches are the two basic groups into which preventive strategies can be divided. The first class of strategies is centered on hiding a design's structural and functional elements to make it more difficult for an attacker to include Trojans. [15] [16]. The main concept is to modify a circuit's state transition function by using a key-based obfuscation technique. It alters a circuit's behavior so that it operates in two different modes, regular mode and obscured mode [16]. While the desired output is produced in the conventional mode of operation, some input patterns result in improper functional behavior in the obfuscated mode. A difficult-to-detect Trojan is difficult for an opponent to introduce when the internal circuit nodes' rarity is obscured by such a modification.

By filling in empty areas, the second class of techniques seeks to prevent the inclusion of additional circuit components in a design. It's critical to properly conceal them in order to stop an attacker from identifying filled holes in a scheme and replacing them with Trojan circuit. Built-in self-repair (BISA) is a method that uses functional standard cells rather than filler cells to occupy every available area in an integrated circuit (IC). It is difficult for an enemy to locate any space on the device to install Trojans using such a strategy [17]. BISA is a self-authenticating system, therefore it would be simple to spot any attempt to tamper with or alter the BISA design. Finding the empty spaces on the design layout, inserting the BISA cells to fill in the spaces, routing, and connecting these cells so that they are self-authenticating after fabrication are all part of the flow. By establishing an inaccurate signature during self-authentication, BISA offers defense against removal attacks, in which one or more of the filler standard cells are removed to install the Trojan cells. With the same signature difference, it also defends against redesign and resizing attacks. However, filler techniques like BISA cannot stop maliciously changing a set of transistors or adding a circuit that doesn't need more layout room.

7.2 How to Facilitate Trojan Detection

Specially designed on-chip embedded structures can help with Trojan identification, much like they can when testing for flaws and useful defects. It must solve the difficulties brought on by the circuit's infrequent activating nets, process changes, and measurement noise [18][19]. The following is a description of some of the main DFS strategies that try to address them.

- On-Chip Security Monitors: Even with the best measurement device and careful vector generation, a small Trojan (for example, of size 1-100 transistors) in a big multibillion transistor design might generate a scarcely perceptible influence in delay or supply current. This means that the "detection sensitivity," which is calculated as the percentage deviation in latency or current caused by a Trojan, is too low to properly detect a Trojan. The noise brought on by process and environmental changes considerably

exaggerates the issue of lower detection sensitivity [18]. To precisely identify minute delay fluctuations brought on by a Trojan, researchers have suggested arranging circuit lines into ring oscillators [20], [21]. Additionally, they offer a scalable solution for designs of any size, complexity, and Trojans of different shapes and sizes. To minimize the overhead of current sensors, one can make use of the power-gating circuits already present in contemporary designs.

- Eliminating Rare-Triggered Nets: Trojans' covert behavior shows that they link to nets that are difficult to regulate or see. A Trojan, for instance, may have trigger inputs with q > 1, which may be nets with 1) extremely low transition probability; and 2) uncommon combinations.
- Increasing TCR: As previously said, efficient vector production can seek to increase TCR. However, one can achieve the same result by appropriately modifying the design. Based on their final physical placement in the arrangement, scan cells can be rearranged. In a design, layout-aware scan-cell reordering can restrict switching activity in some areas while localizing it in others. According to simulation data, this strategy is particularly successful against dispersed and localized Trojans as well as tiny and large Trojans. This is due to the fact that, even if an attacker distributes the Trojan gates around various sections of the circuit, the reduction in circuit switching employing this technique is far bigger than the reduction in Trojan switching. Voltage inversion can also increase Trojan's contribution to the overall supply current [19].

7.3 Configurable Security Monitors

One method for enabling real-time functionality monitoring utilizing security monitors (SMs) is based on the integration of reconfigurable logic in a specific SoC[22][23]. Finite state machines (FSMs) are implemented in SM in order to monitor the behavior of relevant signals. Through a signal probe network, these signals are sent to SMs (SPN). The configuration of one SM does not interfere with the operation of other SMs or the system as a whole. The tests can be run simultaneously with the regular circuit operation and, if a divergence from expected functionality is found, will initiate the proper countermeasures.

A configuration and control processor can reconfigure SMs to dynamically apply various security checks to identify unanticipated or improper behavior caused by a Trojan, such as access to protected memory space or switching to test mode when the system is operating normally. The functionality of the infected logic is typically implemented by the reconfigurable core, which is then turned off or bypassed.

7.4 Variant-Based Parallel Execution

Another method for determining hardware trust in real-world use at runtime [24] combines multicore hardware with dynamic distributed software scheduling. It entails scheduling and running concurrently functionally equivalent variants (obtained by various compilations or algorithm changes) on various processing elements (PEs) equipped to run these variants, then comparing the outcomes. A fresh PE is engaged if a mismatch is found until a match can be made and the PEs that have Trojans are found. In order to achieve high dependability in the presence

of software flaws, the concept is comparable to N-version programming in software, which generates and runs numerous functionally equal copies of the same program. Even if two PEs have similar Trojans, they are unlikely to be triggered at the same time since they execute different versions of the same code. For particular computer systems, such as manycore processors, it can attain high levels of confidence at the expense of additional computation that may result in performance and energy overhead. The success of such a strategy depends on the effective creation of variants.

7.5 Using Hardware and Software

A software solution that can detect Trojan activation and/or tolerate Trojan effects can offer efficient security in the case of microprocessor-based systems. For instance, as stated in [25], a straightforward verifiable "hardware guard" module external to the processor can be utilized for runtime execution monitoring to detect Trojan activation. With the use of periodic OS checks and live check functionality, it primarily tackles DoS and privilege escalation attacks. Based on SPECint 2006 benchmark programs, it can be implemented with just a 2.2% average performance overhead. Additionally, the BlueChip [26] hybrid hardware/software method, which combines runtime monitoring with a design-time component, is being investigation. With the aid of design verification tests, it looks for any unused circuitry and flags it as suspicious. The questionable circuitry is replaced at runtime with exception logic, which can cause a software exception and circumvent malicious hardware Trojans, enabling the system to function normally. This method is intended to get around hardware Trojans, which function similarly to software Trojans. These Trojans can be injected into an embedded processor's instruction code or hardware IP.

8. Conclusions

Overall, computer security experts are gravely concerned about software attacks aided by hardware Trojans. There are no publicly published guidelines for operationally reducing or proactively addressing these types of assaults, nor are the attacks categorized and documented with a standard nomenclature. Even though it might be challenging to identify and stop these attacks, researchers are making considerable strides toward creating efficient defenses. It is critical to keep researching and creating new defenses against these kinds of assaults as linked devices and the Internet of Things are used more frequently.

References

[1] S. Adee, "The Hunt For The Kill Switch," in IEEE Spectrum, vol. 45, no. 5, pp. 34-39, May 2008, doi: 10.1109/MSPEC.2008.4505310.

[2] Defense Advanced Research Projects Agency (DARPA), ‘‘TRUST in integrated circuits (TIC),’’ 2007. [Online]. Available: http://www.darpa.mil/MTO/solicitations/baa07-24.

[3] Sidhu, S.; Mohd, B.J.; Hayajneh, T. Hardware Security in IoT Devices with Emphasis on Hardware Trojans. *J. Sens. Actuator Netw.* 2019, *8*, 42. https://doi.org/10.3390/jsan8030042

[4] Jeff Forristal , “Hardware Involved Software Attacks”, 2011. http://gauss.ececs.uc.edu/Courses/c6056/lectures/PDF/Forristal_Hardware_Involved_Software_Attacks.pdf

[5] B. Delgado and K. L. Karavanic, "Performance implications of System Management Mode," *2013 IEEE International Symposium on Workload Characterization (IISWC)*, Portland, OR, USA, 2013, pp. 163-173, doi: 10.1109/IISWC.2013.6704682.

[6] PEYTON JONES, S. (2003). 7 Basic Input/Output. *Journal of Functional Programming, 13*(1), 97-102. doi:10.1017/S0956796803000911

[7] Rafal Wojtczuk, Joanna Rutkowska, “Software attacks against Intel(R) VT-d technology”, 2011. https://papers.put.as/papers/firmware/2011/SoftwareAttacksonIntelVT-d.pdf

[8] R. Langner, "Stuxnet: Dissecting a Cyberwarfare Weapon," in IEEE Security & Privacy, vol. 9, no. 3, pp. 49-51, May-June 2011, doi: 10.1109/MSP.2011.67.

[9] Fiona K.E. MDuff, Suzanne D. Turner, Jailbreak: Oncogene-induced senescence and its evasion, Cellular Signalling, Volume 23, Issue 1, 2011, ISSN 0898-6568, https://doi.org/10.1016/j.cellsig.2010.07.004.

[10] D. Ahmad and I. Arce, "The Confused Deputy and the Domain Hijacker," in *IEEE Security & Privacy*, vol. 6, no. 1, pp. 74-77, Jan.-Feb. 2008, doi: 10.1109/MSP.2008.25.

[11] J. Kleinberg, H. Attiya and N. Lynch, "Trade-offs between message delivery and quiesce times in connection management protocols," *Proceedings Third Israel Symposium on the Theory of Computing and Systems*, Tel Aviv, Israel, 1995, pp. 258-267, doi: 10.1109/ISTCS.1995.377024.

[12] M. Tehranipoor and F. Koushanfar, ‘‘A survey of hardware Trojan taxonomy and detection,’’ IEEE Design Test Comput., vol. 27, no. 1, pp. 10–25, Jan.–Feb. 2010.

[13] S. Bhunia, M. S. Hsiao, M. Banga and S. Narasimhan, "Hardware Trojan Attacks: Threat Analysis and Countermeasures," in *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1229-1247, Aug. 2014, doi: 10.1109/JPROC.2014.2334493.

[14] L. Lin, W. Burleson, and C. Paar, ‘‘MOLES: Malicious off-chip leakage enabled by side-channels,’’ in Proc. Int. Conf. Comput.-Aided Design, 2009, pp. 117–122.

[15] J. Rajendran et al., ‘‘Securing processors against insider attacks: A circuit-microarchitecture co-design approach,’’ IEEE Design Test, vol. 30, no. 2, pp. 35–44, Mar.–Apr. 2013.

[16] R. S. Chakraborty and S. Bhunia, ‘‘Security against hardware Trojan attacks using key-based design obfuscation,’’ J. Electron. Testing, Theory Appl., vol. 27, no. 6, pp. 767–785, Dec. 2011.

[17] K. Xiao and M. Tehranipoor, ‘‘BISA: Built-in-self-authentication for preventing hardware Trojan insertion,’’ in Proc. IEEE Int. Workshop Hardware-Oriented Trust Security, 2013, pp. 45–50.

[18] S. Narasimhan, W. Yueh, X. Wang, S. Mukhopadhyay, and S. Bhunia, ‘‘Improving IC security against Trojan attacks through integration of security monitors,’’ IEEE Design Test Comput., vol. 29, no. 5, pp. 37–46, Oct. 2012.

[19] M. Banga and M. Hsiao, ‘‘VITAMIN: Voltage inversion technique to ascertain malicious insertions in ICs,’’ in Proc. IEEE Int. Workshop Hardware-Oriented Trust Security, 2009, pp. 104–107.

[20]] J. Rajendran, V. Jyothi, O. Sinanoglu, and R. Karri, ‘‘Design and analysis of ring oscillator based Design-for-trust technique,’’ in Proc. IEEE 29th VLSI Test Symp., 2011, pp. 105–110.

[21] X. Zhang and M. Tehranipoor, ‘‘RON: An on-chip ring oscillator network for hardware Trojan detection,’’ in Proc. Design Test Eur. Conf. Exhibit., 2011, DOI: 10.1109/DATE. 2011.5763260.

[22] M. Abramovici and P. Bradley, ‘‘Integrated circuit securityVNew threats and solutions,’’ in Proc. 5th Annu. Workshop Cyber Security Inf. Intell. Res., 2009, DOI: 10.1145/1558607. 1558671.

[23] S. Bhunia et al., ‘‘Protection against hardware Trojan attacks: Towards a comprehensive solution,’’ IEEE Design Test Comput., vol. 30, no. 3, pp. 6–17, May–Jun. 2013.

[24] D. McIntyre, F. Wolff, C. Papachristou, and S. Bhunia, ‘‘Dynamic evaluation of hardware trust,’’ in Proc. IEEE Int. Workshop Hardware-Oriented Security Trust, 2009, pp. 108–111.

[25] G. Bloom, B. Narahari, and R. Simha, ‘‘OS support for detecting Trojan circuit attacks,’’ in Proc. IEEE Int. Workshop Hardware-Oriented Security Trust, 2009, pp. 100–103.

[26] M. Hicks, M. Finnicum, S. T. King, M. M. K. Martin, and J. M. Smith, ‘‘Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically,’’ in Proc. IEEE Symp. Security Privacy, 2010, pp. 159–172.