University of Piraeus

School of Information and Communication Technologies

Department of Digital Systems

Undergraduate Program of Study/Postgraduate Program of Studies

MSc Digital Systems Security

Malicious Hardware

MSc Dissertation

of

Ntouska Katerina

Supervisor Professor: Xenakis Christos

Piraeus, February 2022

Η σελίδα αυτή είναι σκόπιμα λευκή.

Acknowledgments

I would like to take this opportunity to express first my gratitude to Professor Xenakis Christos, who is my thesis advisor and for devoting time to reading and reviewing my work.

Finally, I would also like to thank my family for being always supportive of my education, indeed without them none of this would be possible.

Table of Content

Abstract 4
Περίληψη 5
Figure Index 6
Chapter 1: Introduction 8
Motivation and Background 8
Production Process of Hardware Systems 11
Workflow 11
Vulnerabilities 12
Chapter 2: Malicious Hardware Overview 15
Taxonomy 15
First Approach (8) 15
Second Approach 17
Hardware Backdoor (13) (14) 17
Integrated Circuits (15) 19
Hardware Trojan (17) (18) 21
Injection Phase 25
Attackers 26
Chapter 3: Malicious Hardware Analysis 28
Detection (18) (20) 28
Destructive 29
Non-Destructive Invasive 30
Non-Destructive Non-Invasive 32
Localization 35
Activation 35
Mensuration 38
Calculation 40
Prevention 40
Obfuscation 41

Invasion 46

Chapter 4: Malicious Hardware Use Cases 55

Chipset Level Backdoor 55

DMA Attack 56

Software Attacks Enabled by Malicious Hardware 58

Malicious USB Device 59

Chapter 5: Conclusion 61

Chapter 6: References 62

Abstract

Nowadays with the rapid development and application of new technologies in all areas of our lives the interaction with integrated circuits is daily. This thesis focuses and analyzes all aspects of malicious hardware and hardware-based attacks. We briefly analyze the production process of hardware systems and the vulnerabilities that can be founded in each of the steps. Moving on, we approach the taxonomy of malicious hardware with two different ways, describe the lifecycle of the hardware system which provides several opportunities to include unwanted functionality and use this to categorize the attackers. Moreover, detection, localization and prevention mechanisms are outlined and thoroughly analyzed as countermeasures in order to deal with the risk of malicious hardware and more specifically Hardware Trojans. Finally, we refer shortly to malicious hardware use cases featuring specific attack vectors.

Περίληψη

Στην εποχή μας, με την ραγδαία εξέλιξη και εφαρμογή των νέων τεχνολογιών σε όλους τους τομείς της ζωής μας, η αλληλεπίδραση με τα ολοκληρωμένα κυκλώματα είναι καθημερινή. Η παρούσα διπλωματική εργασία εστιάζει και αναλύει όλες τις πτυχές του κακόβουλου υλικού και των επιθέσεων που βασίζονται στο υλικό. Αναλύουμε εν συντομία τη διαδικασία παραγωγής συστημάτων υλικού και τις ευπάθειες που μπορούν να δημιουργηθούν σε κάθε ένα από τα στάδια. Προχωρώντας, προσεγγίζουμε την κατηγοριοποίηση του κακόβουλου υλικού με δύο διαφορετικούς τρόπους, περιγράφουμε τον κύκλο ζωής του συστήματος υλικού που παρέχει αρκετές ευκαιρίες για την ενσωμάτωση ανεπιθύμητης λειτουργικότητας και το χρησιμοποιούμε για να κατηγοριοποιήσουμε τους κακόβουλους εισβολείς. Επιπλέον, περιγράφονται και αναλύονται διεξοδικά οι μηχανισμοί ανίχνευσης, εντοπισμού και πρόληψης ως αντίμετρα για την αντιμετώπιση του κινδύνου του κακόβουλου υλικού και πιο συγκεκριμένα των Δούρειων Ίππων Υλικού. Τέλος, αναφερόμαστε εν συντομία σε περιπτώσεις χρήσης κακόβουλου υλικού που χαρακτηρίζονται από συγκεκριμένους τρόπους επίθεσης.

Figure Index

Figure 1: Hardware design flow, synthesis, and layout process 12
Figure 2: IC Production supply chains 13
Figure 3: ASIC and FPGA design flow 13
Figure 4: Generic design flow proposed by the Department of Defense Science Board 13
Figure 5: Hardware attacks taxonomy 15
Figure 6: Hardware backdoors taxonomy 19
Figure 7: Examples of Trojans with various trigger mechanisms 23
Figure 8: Examples of analog payload Trojans 24
Figure 9: Trojan taxonomy based on trigger and payload mechanisms 24
Figure 10: Lifecycle of a hardware system 25
Figure 11: Attackers' categories based on the supply chain 27
Figure 12: Hardware Detection Classification 29
Figure 13: Localization mechanisms 35
Figure 14: Concept of activity minimization. In (a) circuit activity is created by 36
Figure 15: Illustration of the concept of Region and Radius in a circuit 38
Figure 16: Architecture of Simulation Model of paper (28) 39
Figure 17: Arrangement of ring oscillators and Trojans on the chip surface of paper (29) 39
Figure 18: Prevention methods taxonomy 40
Figure 19: Prevention by obfuscation using reconfigurable logic based on (21) 43
Figure 20: Prevention by obfuscation using different variants of functionally equivalent software based on (35) 44
Figure 21: Prevention by obfuscation using simple encryption and reordering of data based on (13) 46
Figure 22: Principle of preventing Trojans by software emulation based on paper (1) 47
Figure 23: Two-gate circuit design and truth table 48
Figure 24: Data guard to block malicious access to the memory bus based on paper (23) 50
Figure 25: TRUSTNET Monitor Overview & Microarchitecture based on paper (39) 52

Figure 26: Guardian core to block malicious access to the memory bus based on paper (40) 52
Figure 27: Prevention by transformation using dual encryption based on paper (24) 54
Figure 28: Shadow Mode inside the Processor 58

Chapter 1: Introduction

Motivation and Background

Cyber-attacks are a significant threat in today’s modern world, with technology evolving faster every day and playing such a critical role in people’s daily lives. People need to feel protected when using their workstations or electronic devices, and in order to do that, more threats have to be identified and treated. Malicious hardware is one category of these threats. It can be defined as a vulnerability, flaw, or built-in back doors such as code inside hardware or firmware of computer chips or hardware Trojans in an integrated circuit, enabling an attacker to gain unauthorized access to a system. Once the machine’s hardware is compromised, the intruder can read the data in the system. For example, the user’s personal identifying information such as passwords, bank account information, or any other sensitive data available, tamper with it or use it for malicious purposes. The described hardware compromisation results in a catastrophic loss of security, which can be expensive to the user or even the organization if the device is used for business purposes and leave the user with untrustworthy hardware. We can only assume that either the security was neglected by design, or the appropriate security measures were not taken, or finally there was a fault in manufacturing the hardware. For example, the vendor is likely to have implemented a backdoor to connect back to them for services but did not apply high-security standards regarding access, authentication & authorization management, or network security. So, that will leave the system vulnerable.

Hardware-level vulnerabilities are difficult to overcome and require, in most cases, physical replacement of the compromised hardware components, resulting in a significant loss of money. For example, a hardware recall similar to Intel’s Pentium FDIV bug, which cost 500 million dollars to recall five million chips, has been estimated to cost many billions of dollars today. Furthermore, malicious hardware can provide control to the software running above due to the fact that hardware is the lowest layer in the computer system. This low-level control enables sophisticated and stealthy attacks aimed at evading software-based defenses. Such a scenario is possible when malicious logic is inserted during design time and gets triggered with a special or unlikely event like introducing a sequence of bytes into the hardware by attackers. Nowadays, the design phase of a hardware component is highly complicated, making it vulnerable to the insertion of malicious logic. We can distinguish two scenarios in which the malicious logic can be inserted during the design phase and remain undetected even after the validation and testing of the hardware. In the first scenario, one employee can intentionally insert malicious circuits into a design prior to validation. In the second scenario, a generally trusted third-party intellectual property (IP) can be proved untrustworthy containing hidden malicious circuits. In either scenario, there is a high possibility that these malicious logic circuits can remain undetected during final design verification and validation and be shipped to be integrated into customers' computer systems. Subsequently, it can be used to escalate privileges, turn off access control checks, or execute arbitrary instructions. Any of the above cases result in the attacker taking control of the machine and eventually releasing system-level attacks (1).

All these vulnerabilities and bugs on the hardware allow attackers to exploit them in order to achieve their ulterior motive. The motivations behind these malicious hardware attacks vary. Some of those can be hardware cloning, unlocking hidden features, imitating user authentication for system access, information leakage, or even unlocking devices to gain access to an internal shell or increase control of a system. Some reported accidents of potential hardware attacks have been reported to steal secret information, gain control of devices, or even crush a system.

Israel, in September 2007, launched a successful airstrike on a nuclear reactor in Syria, while Syria's advanced air defense system did not respond throughout the operation. In 2008, there were speculations that a built-in kill switch had disabled Syria's air defense system that could be accessed and activated remotely. Since the malicious hardware used in industrial fields and military are often highly confidential, researchers cannot accurately determine the implementation details of these attacks. However, it still shows the worries of various communities about the destructive power of malicious hardware alterations.

In 2010, the motor vehicle manufacturer Toyota had to place one of the most extensive product recalls in the automobile industry in the U.S. About 6.5 million cars were affected due to reports of unintended acceleration. Imagining that such an error can be placed maliciously by an attacker into software and hardware parts highlights the importance of advanced research in hardware and embedded systems security. (2)

Another prominent example to picture the threat of malicious hardware is the worm Stuxnet that had been detected in 2010. The malicious code was designed specifically to observe and control technical processes in industrial facilities. Due to the high complexity of the worm, it is generally assumed that the designers of Stuxnet had knowledge of unrevealed Microsoft exploits and the architecture of the industrial control system of the facility. Supposing that such worm could be (partially) implemented in hardware, the operation could be even more precise. In general, using hardware Trojans to attack infrastructure and military applications, such as industrial complexes and power plants, can have a huge impact on the populations of a region and yield serious financial damages. These examples illustrate how pervasive computer hardware has become in human life and how much society depends on it—without being aware of it. (3)

In 2016, Yang proposed a tiny malicious Hardware Trojan named A2, in which they implemented a privilege escalation attack by running a set of seemingly harmless commands in the OR1200 processor. Such lightweight analog malicious backdoors are awfully hard to detect.

Finally, the Free Software Foundation revealed, in January 2018, the Intel Management Engine (ME), a built-in subsystem in Intel computers, which can take complete control over the computer and even has access to the main memory. The ME structure can be a major threat to the users' privacy and security. Users though do not have the ability to audit, examine, or disable it, and from their perspective, this could also be considered a malicious hardware alteration.

Production Process of Hardware Systems

An attack at the hardware level differs from an attack at the software level. Opposite to software, hardware is a physical good and can only be altered in limited ways after manufacturing. Hardware also requires significant effort and human resources in order to be duplicated as easily as software. So as to examine hardware level attack scenarios, the following questions should be answered:

1. Where are possible vulnerabilities located?
2. How can vulnerabilities be protected?
3. Who are the possible attackers?

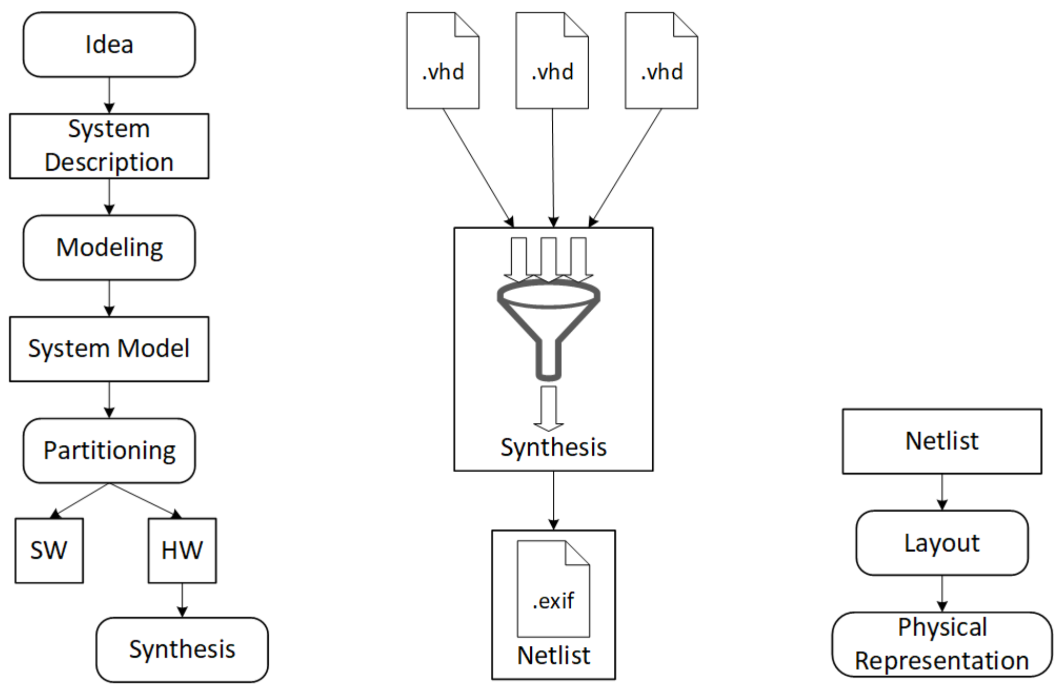
In order to respond to the above questions, we will first give a brief overview of the production and design process of microprocessors to provide a detailed understanding of the risks and vulnerabilities originating in the hardware production process.

Workflow

Growing competitive pressure and ever-shorter product launch times make it necessary to reuse already developed hardware components. Therefore, hardware must be present in a special, reusable format. The solution is to textually describe the behavior of the hardware so that it can be generated through synthesis. Such descriptions are verbalized in so-called hardware description languages (HDL) such as VHDL or Verilog.

Fig. 1(a) below shows some substantial parts of the hardware development process. After the system has been described in natural language, the system model is designed in a modeling language. Based on the model, partitioning is used to decide which parts of the functionality will be implemented in hardware and which in software. Through synthesis, the hardware will be generated from the system model components intended for this purpose. Fig. 1(b) shows how the synthesis process works. The “source code” that is scattered across multiple files is translated into a format that only includes logic gates and connecting signals.

The resulting representation of the hardware netlist is then mapped to the desired target technology (ASIC, FPGA). Here, the respective logic elements are placed on the chip and are connected to each other via electrical connections (place and route). The result is a file that fits the physical representation of the hardware (see Fig. 1(c)). The physical representation is submitted to the chip manufacturer (Tape Out). After manufacturing, the chips are returned or delivered to the market.



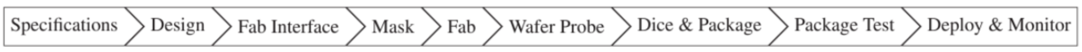
(a) HW development process (b) Synthesis Process (c) Physical layout process

Figure 1: Hardware design flow, synthesis, and layout process

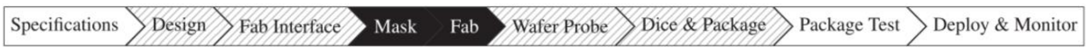
Vulnerabilities

As stated above, the hardware design and production process consist of many steps and corresponding interfaces. Each of these steps in the process along with every transition between the process steps is vulnerable.

Fig. 2(a) below summarizes every step of the process, focusing on the physical manufacturing process (4). Some of the physical manufacturing process steps cannot be trusted; thus, they are categorized based on their trustworthiness. Fig. 2(b) below illustrates this by assigning white to the trusted and black to the non-trustworthy steps. If no assumption about trustworthiness can be made, the step is hatched. Mask and Fabrication steps cannot be trusted because during pre-IC fabrication; a backdoor could be inserted at the time of design, within integrated IP, or even during mask or silicon modification. After IC fabrication, malicious logic could also find its way in through physical or packaging modifications, side-channel exploits (i.e., power, analog, RF), and even maintenance or upgrade updates.



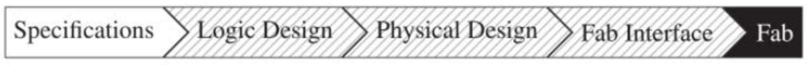
(a) Traditional Supply chain



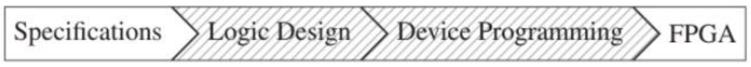
(b) Supply chain with steps based on their trust level

Figure 2: IC Production supply chains

Authors at (4) also specify confidentiality classes for the different development processes for ASICs (Fig. 3(a)) and FPGAs (Fig. 3(b)).



(a) Design flow ASIC



(b) Design flow FPGA

Figure 3: ASIC and FPGA design flow

The Defense Science Board, Department of Defense, U.S. (5), provides a thorough look at the links between the development and production processes and the risks involved (Fig. 4).

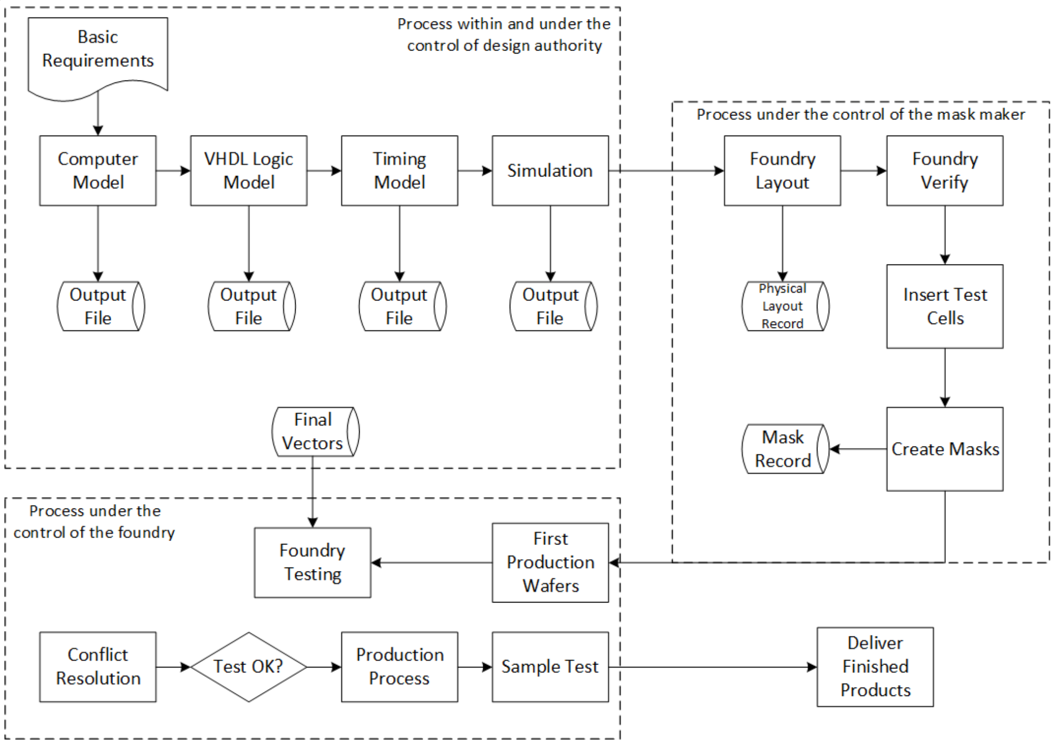


Figure 4: Generic design flow proposed by the Department of Defense Science Board

In this way, malicious functionality could be inserted directly as code by developers into the hardware description. Moreover, it is likely that functionality bought as IP cores (IP—Intellectual Property) implements malicious, unwanted functionality in addition to the expected one. We also cannot eliminate the possibility that errors or additional functionality are introduced by software tools, such as synthesizers or place-and-route tools. (6)

Chapter 2: Malicious Hardware Overview

Taxonomy

ENISA defines vulnerability as: "*The existence of a weakness, design, or implementation error that can lead to an unexpected, undesirable event compromising the security of the computer system, network, application, or protocol involved (ITSEC)*." So, based on that fact, we can define a hardware attack as the act of taking advantage of a hardware vulnerability. (7)

First Approach (8)

The first approach towards hardware attacks taxonomy is outlined in Fig. 5.

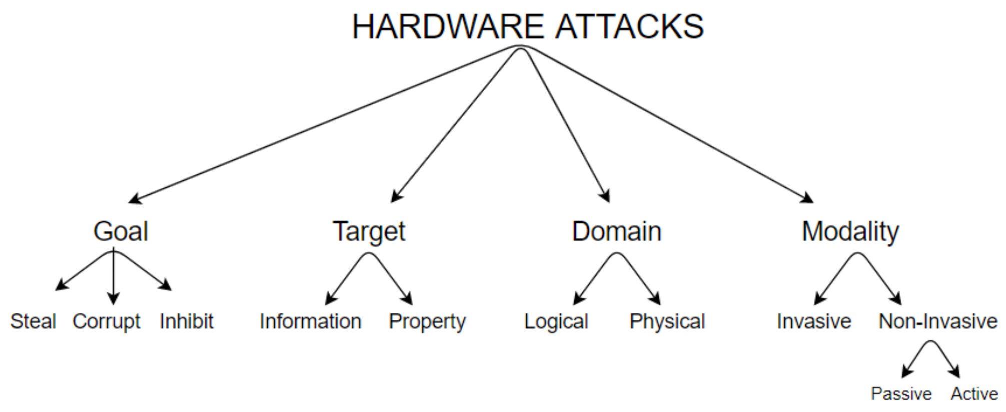


Figure 5: Hardware attacks taxonomy

Firstly, a hardware attack is categorized based on the goal, the desired result that the attacker is trying to succeed, in our case, a malicious action. The attacker's aim is the attacked hardware, towards which efforts are directed. The attacked hardware is characterized as a target and can be either information that the hardware is handling or property of the hardware itself (functional or non-functional).

A hardware attack can be initiated to steal, corrupt, or inhibit a target. Either of those actions can result in a violation of the information security principles, the Confidentiality Integrity Availability (CIA) triad. More specifically, stealing a cryptographic key discloses confidential information or trade secrets which are intellectual property (IP) rights on confidential information that may be sold or licensed, violates the Confidentiality principle since the attacker obtains confidential data or information illegally. Cryptography is used primarily to protect the confidentiality of data. That is why a cryptographic key is needed to ensure that the data exchanged between two parties remain confidential. However, the use of cryptography is not only restricted to confidentiality which is the main focus but also extends to checking the integrity and authentication processes as well. Furthermore, when corrupting a permission file or a memory word, the attacker modifies an asset without authorization, so the Integrity is lost. Finally, inhibiting a service or a defense mechanism allows the attacker to shut out of the system the authorized users in violation of the Availability.

Hardware attacks also have a domain, an area in which they are implemented. The domain is classified into logical and physical. The logical domain attacks are implemented starting from the upper layers with respect to the hardware. These attacks are not targeting directly the hardware but the software running on top of it. Some examples of those attacks are privilege escalation, exploitation of vulnerabilities in processor microarchitectures such as Meltdown (9), Spectre (10), or cache-based attacks (11) (12). On the other hand, physical domain attacks are implemented directly on the hardware device through proper actions such as backdoor programs and modifications in integrated circuits.

Moreover, the hardware attacks can also be categorized according to the modality in which they are carried out. We can further split them into invasive and non-invasive (passive or active) attacks. The invasive attacks are the ones including physical intrusions against the attacked hardware device such as disconnection ("*Usually a circuit breaker, a fused switch, or a fused circuit-breaker-assembly that disconnects the conductors of an electric circuit from the source of supply.*"), desoldering ("*In electronics, desoldering is the removal of solder and components from a circuit board for troubleshooting, repair, replacement, and salvage.*"), or depackaging ("*Depackaging a chip in order to get access to its surface.*") of its internal components. Attacks belonging to this sub-category are micro probing, reverse engineering, or data remanence attacks.

In contrast to invasive attacks, the non-invasive ones are carried out without any physical contact with the device. This type of attack is divided into passive and active non-invasive attacks. Passive non-invasive attacks can be carried out by analyzing and measuring one or more physical dynamic entities at the attacked hardware device. Examples of these sub-category attacks are all distinct types of side-channel attacks (timing, power, electromagnetic, acoustic, and optical attacks). Active non-invasive attacks are carried out by performing a specific action on the device, forcing the system into abnormal states that the goal is easier to reach. Examples of these sub-category attacks are all distinct types of fault attacks (supply, clock, heating, and radiation attacks) and test-infrastructure-based attacks.

Second Approach

A second approach towards hardware attacks taxonomy could classify them depending on the hardware vulnerability exploited. In such a case, hardware attacks can be split into hardware backdoors, maliciously altered or flawed integrated circuits, and hardware trojans. Each of the previously mentioned categories poses a threat to the hardware and subsequently to the whole system. We are going to analyze these categories thoroughly.

Hardware Backdoor (13) (14)

Hardware backdoors are vulnerabilities inserted intentionally inside a hardware device to guarantee to the entity that builds it either the possibility of later access or use of it outside the scope of intended cases. Furthermore, we have to consider that malicious hardware modification can pose a threat to the system, providing attackers a foothold into sensitive or critical information.

Such malicious modifications in order to create backdoors can come into the design in several ways. First of all, they can either come from a core design component, for example, a few lines of Hardware Design Language (HDL), or a third-party intellectual property (IP). Nowadays, hardware designs use a variety of third-party IP components that are integrated into the designs after passing validation tests but without code review for any malicious modification. Nevertheless, even in the case of a possible complete code review, it could be highly improbable to discover a thoroughly hidden backdoor since modern designs come with many bugs. Additionally, the malicious modifications might as well come during the hardware design phases. At first, the hardware is designed to meet operational requirements and coded into HDL. Then, hardware undergoes many and strict validations as hardware bugs are more expensive to fix after deployment. When done, the design is processed using computer-aided design tools (CAD). So, thousands of engineers can have access to hardware during its design and creation phases and alter it maliciously.

A key factor of hardware backdoors is that they can remain inactive during testing and can be triggered in order to wake up at a later time. Then during the validation phase, we encounter difficulty detecting them because of the exponentially many different ways that the hardware backdoors can be expressed. We will describe two trigger types, ticking timebombs and cheat codes, that can wake up a hardware backdoor to attack the system.

Ticking timebombs attacks can be programmed into hardware devices with malicious HDL code in order to be triggered automatically a fixed amount of time after the unit powers on. An attribute of this trigger type is that it can go undetected by evading validation techniques and then trigger without any warning. This poses a significant threat to many high-security areas. The timebomb can undermine the system security or function as a ‘kill-switch’ in hardware despite the use of a secure, tamper-free environment and running of trusted code. It is crucial to notice that the adversary does not need to have privileged access to the machine in order to launch this attack.

Cheat code attacks, on the other hand, require the attacker to have privileged access to the machine (e.g., is a user and a designer) in order to execute code on the malicious hardware and provide the cheat code key. So, cheat codes can be defined as backdoors that are triggered by data values. The adversary in this type of attack needs to insert a special input or sequence of inputs that functions as a key in order to trigger the malicious hardware to attack. This input, also characterized as the identification of the adversary at the hardware backdoor logic, must be unique in order to avoid any detection during validation techniques. The cheat codes can be communicated in two ways, either as a single-shot or as a sequence cheat code. Single-shot cheat codes are large pieces of data sharing the entire code through a single data value. In contrast, sequence cheat codes are small pieces of data over multiple cycles or inputs sharing the code in pieces. The cheat codes can introduce themselves as addresses in hex; for example, 0xdecafbad can be a secret trigger to wake up a hardware backdoor.

Actions resulting from the triggering of a backdoor are often referred to as payload. Payloads can be categorized either as emitter or corrupter. Emitter payloads send out extra information or perform extra actions beyond what is specified and are easier for an attacker to implement. In comparison, corrupter payloads change existing messages to alter data that plays a specific role and are likely to cause other aspects of the system to fail in unexpected ways without careful engineering.

The above descriptions regarding hardware backdoors taxonomy are depicted below in Fig. 6.

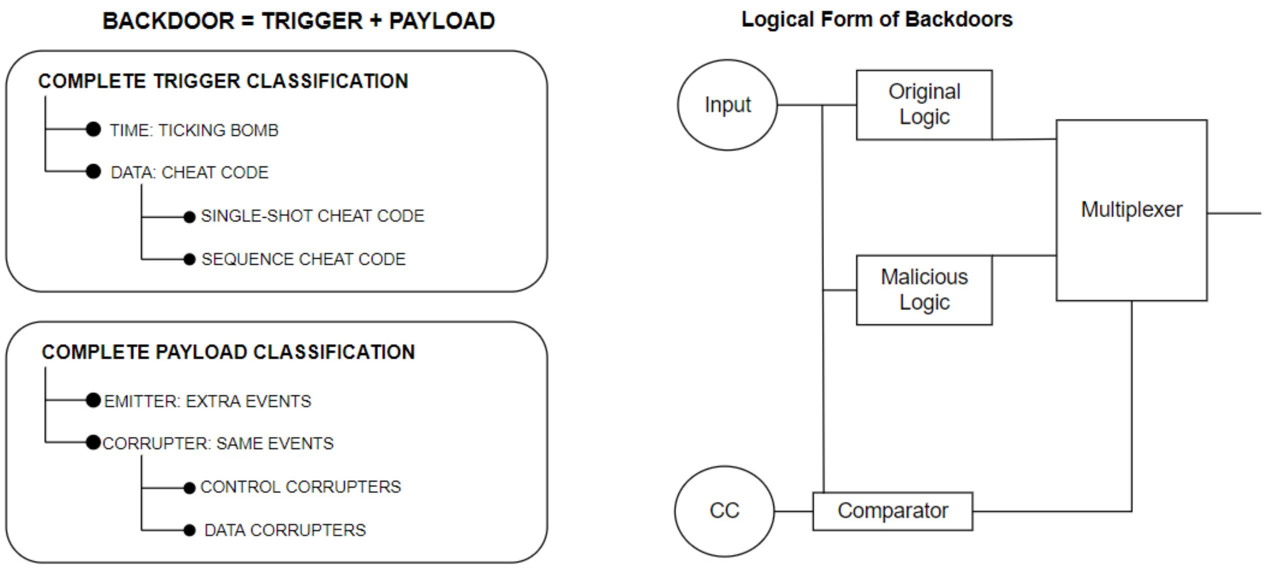


Figure 6: Hardware backdoors taxonomy

Integrated Circuits (15)

An integrated circuit (also referred to as an IC, a chip, or a microchip) is a set of electronic circuits on one small flat piece (or “chip”) of semiconductor material that is usually silicon. It is defined as a circuit in which all or some of the circuit elements are inseparably associated and electrically interconnected so that it is considered to be inseparable for construction and marketing. Circuits fulfilling this definition can be constructed using a variety of different technologies, including thin-film transistors, thick-film technologies, or hybrid integrated circuits. However, in general, an integrated circuit refers to the single-piece circuit construction primarily known as a monolithic integrated circuit, usually built on a single piece of silicon. (16)

Circuits are the result of the integration of large numbers of tiny MOS transistors into a small chip. Those circuits are smaller, faster and less expensive orders of magnitude than the ones constructed of discrete electronic components. The fast adoption of standardized ICs in place of designs using discrete transistors have been ensured due to the IC’s mass production capability, reliability, and building-block approach to integrated circuit design. Virtually all electronic equipment are using ICs and have reformed the world of electronics. Digital home appliances, mobile phones and computers that are now inseparable parts of the modern societies, are made possible by the small size and low cost of ICs. (16)

ICs are susceptible to weakness in design and implementation defects or flaws introduced during manufacturing, and as a result, adversaries can exploit these vulnerabilities. Furthermore, ICs are also subject to malicious alteration even after the deployment and can frequently be counterfeited and sold as genuine. These vulnerabilities are hard to detect. The expense and time needed to inspect the IC after being manufactured for malicious alteration have a great cost due to the expertise required. Typically, an IC involves almost 100-400 steps, so it is vulnerable to being maliciously altered by several actors.

The main threats introduced into the security properties of an IC are counterfeiting to be sold as genuine, reverse engineering to discover sensitive data and tampering to make malicious alterations or sabotage IC operation. Counterfeiting poses a threat to authenticity and dependability, and it refers to replicas that are approaching the genuine product in such a way that can be mistaken by users, resellers, or testers. Most of the counterfeits are often legitimate IC's that have failed testing have been discarded, then salvaged and resurfaced with a newer version number. These IC's, which are typically documented with misleading mechanical characteristics or performances, are more subject to failure or compromise.

On the other hand, Reverse Engineering poses a threat to intellectual property and data confidentiality because it aims to obtain information about the IC's operation. These attacks are physical and can be either invasive (like depackaging) or non-invasive (like side-channel attacks). They are highly sophisticated attacks and require expertise, resources, and patience that they remain rare. Lastly, tampering poses a threat to integrity and trustworthiness and can be performed to the design of an IC. Its purpose is to alter the functionality of an IC maliciously.

To summarize, we have to point out that because the IC is the lowest layer in a computer system, when specific malicious alterations are performed, it can give the adversaries the ability to bypass, subvert or gain control over all software running above it. All the above can result in stealthy and sophisticated attacks evading software-based defenses. For example, an adversary in order to activate embedded malicious circuits might introduce a sequence of predetermined bytes into the IC. Such action will enable leaks of highly sensitive data or crypto keys or halt the processor at critical or random processing times.

Hardware Trojan (17) (18)

Hardware Trojan can be defined as a malicious modification of the circuitry of an integrated circuit that results in undesired behavior when it is deployed. So, hardware trojans fall under the category of maliciously altered integrated circuits, which can be inserted at any phase of the IC development. Additionally, Hardware Trojans are designed by intelligent adversaries to be stealthy, unlike IC's manufacturing defects which have been extensively researched for years and can manifest with delay fault or stuck-at fault. So, Hardware Trojan is a more complex problem than the appearance of manufacturing defects.

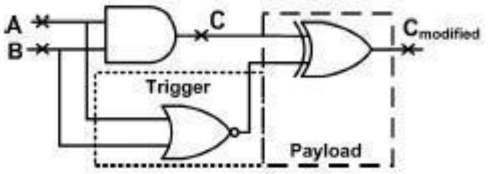
Hardware Trojans can include two core parts: trigger and payload. A trojan trigger can be defined as an activation mechanism that monitors several signals or sequences of occurrences in the circuit. At the same time, a trojan payload is the functionality or the part of the circuit affected by the activation of the Trojan. So, when the trigger meets an anticipated condition or event, the payload gets activated to perform malicious actions. However, the Trojan payload remains dormant until triggered, making it harder to detect because the IC operates as a normal circuit.

Trojan trigger mechanisms can be classified into two types, digital or analog. Digital trigger mechanisms can fall into combinational (which "*becomes active when a specific condition arises in the internal signals and/or circuit flip-flops or a portion of it.*" (19)) and sequential (which "*becomes active upon the appearance of a specific occurring sequence(s) of rare logic values at internal circuit signals*" (19)). A combinational trigger trojan uses conditions such as A=0 and B=0, as shown in Fig. 7(a). That condition triggers the payload, and the output C is being sent out modified ($C_{modified}$). Usually, an attacker will choose an extremely rare condition in order to have very little chance of Trojan triggering during manufacturing testing.

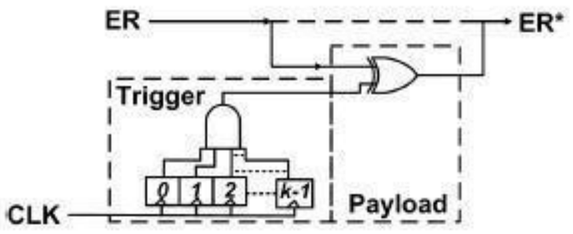
On the other hand, the sequential trigger trojan (also called timebombs) are activated by a period of uninterrupted functioning or the event of a sequence. We can categorize them into synchronous, asynchronous, hybrid, or rare sequences that trigger trojans. Synchronous versions are stand-alone counters which are being triggered when reaching a particular count. In Fig. 7(b), we can see a synchronous k-bit counter activated when the count reaches 2k-1, thus modifying the node ER to an incorrect value at node ER^*. Asynchronous, on the contrary, are not based on a clock but in a rising transition at the output of an AND gate with two inputs (p, q), as shown in Fig. 7(c). Hybrids are a combination of synchronous and asynchronous at the same design, which determines the condition of the trigger (Fig. 7(d)). More complex state machines are used to produce the condition of the trigger based on a rare sequence of events. Note that sequential trigger trojans are more challenging to detect because of the sequence of rare conditions needed to be tested, which can be uncontrollably large.

Analog trigger mechanisms can also be split into on-chip sensors and activity. The on-chip sensor trigger mechanism is used to trigger a malfunction. In Fig. 7(e), we can see an analog trigger with an on-chip sensor which, after a great number of cycles, causes the output to be modified. Lastly, Fig. 7(f) presents a different analog trigger mechanism that causes malfunction after high circuit activity and temperature rise.

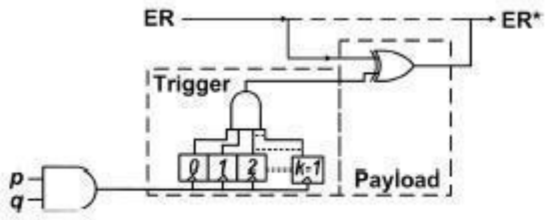
The objective of such attacks can be numerous, taking into consideration the attacker's perspective. Such objectives could be to malign the image of a company in order to gain a competitive edge in the market; to disrupt a major national infrastructure by causing malfunction in electronics used in mission-critical systems; to leak secret information from inside a chip to illegally access a secure system, or to tear down the security and safety of a system. Moreover, recent investigations have shown that an intelligent adversary can mount a hard-to-detect Trojan attack mostly by using just a few transistors or logic gates in a large multimillion transistor system-on-chip (SoC) design. In addition to what is mentioned above, Trojan attacks can also be mounted by selectively changing specific process steps, e.g., the doping profile, to affect a circuit's operational reliability.



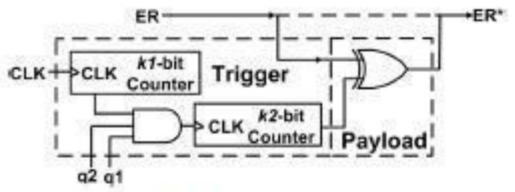
(a) Combinationally triggered Trojan



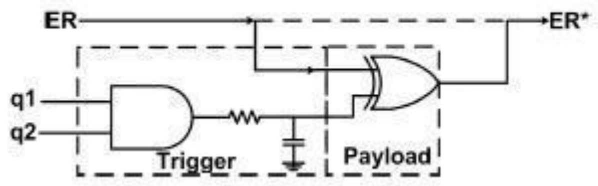
(b) Synchronous counter ("time-bomb") Trojan



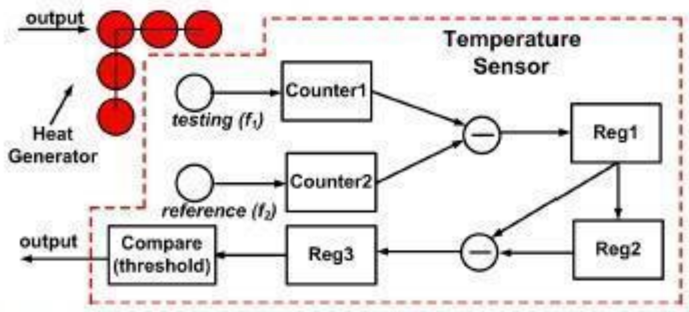
(c) Asynchronous counter Trojan



(d) Hybrid counter Trojan



(e) Analog Trojan triggered based on logic value



(f) Analog Trojan triggered based on circuit activity

Figure 7: Examples of Trojans with various trigger mechanisms

Hardware Trojans can also be categorized based on their payload except from the triggers, which can be split into digital and analog payload types. Analog payload Trojans can affect noise margin, performance, and power which all are circuit parameters. Examples of analog payload trojans are excess activity on the circuit, like the one shown in Fig. 7 (f), bridging faults with the assistance of an inserted resistor (Fig. 8(a)), and delay of the path by increasing the capacitive load (Fig. 8(b)). On the other hand, a digital payload Trojan can affect either the logic values at chosen internal nodes or alter the contents of memory locations. Lastly, based on already explored works, we can introduce two different types of Trojan payload mechanisms: the "information leakage" attack and the "Denial of Service" (DoS) attack.

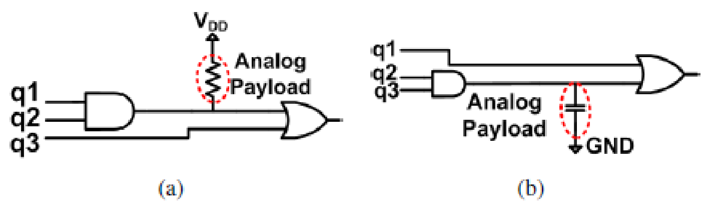


Figure 8: Examples of analog payload Trojans

The above descriptions about the hardware trojans taxonomy are summarized in Fig. 9.

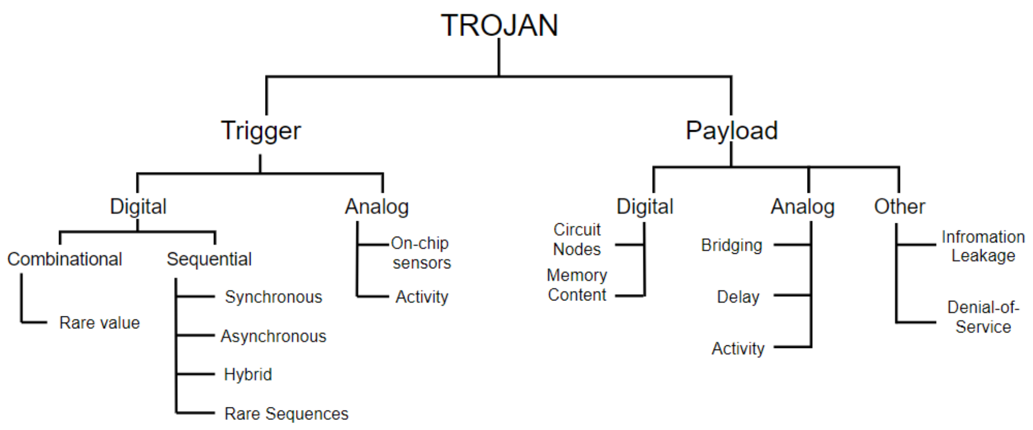


Figure 9: Trojan taxonomy based on trigger and payload mechanisms

Injection Phase

The lifecycle of a hardware system provides several opportunities to include hidden, unwanted functionality. The stages of the lifecycle of a digital IC could be divided into four: Design, Manufacture, Test, and Deployment, as shown in Fig. 10.

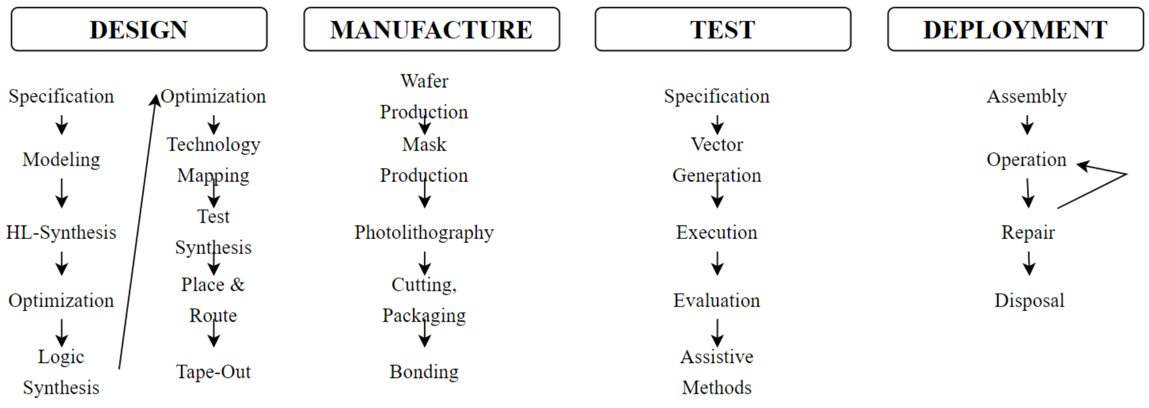


Figure 10: Lifecycle of a hardware system

Design

“Design phase” is the first phase in the lifecycle of a hardware system, and here are specified, for the first time, the properties of the system that is going to be developed. Next, modeling is performed in a system description language, leading to a high-level synthesis being carried out. Then, optimization measurements are taken in order to meet different criteria, and the system’s behavior is mapped to a structural representation using logic synthesis. Proceeding to the next step, the design is mapped to the targeted technology, and the circuit components are placed and connected to the chip area. Then following additional optimization, a possible test synthesis takes place, which inserts dedicated logic and helps the testing of the manufactured system. Lastly, the tape-out step is performed, i.e., the transfer of the production documents to the manufacturer.

Manufacture

Since the design phase is completed, we can commence with the “Manufacturing phase.” The first step in the case of a mask manufacturer, is to produce the masks for semiconductor production. Then, the chipmaker uses the masks in a photolithographic process to expose the wafers produced by the wafer manufacturer. Typically, a multitude of identical ICs is produced on a wafer. Next, the wafer is cut, the individual chips are inserted into packages, and the external connections of the ICs are connected to the connection pins of the package (bonding).

Test

“Testing phase” is the third in the lifecycle of a hardware system and verifies the produced IC’s functionality by using functional testing carried out by test engineers. As soon as the tests are specified, test vectors are generated and applied to the primary input of the integrated circuit. Then, with the aid of the dedicated test logic, the outputs of the integrated circuit are monitored. Furthermore, an analysis of those results helps to make a statement about the proper functioning of the IC. Lastly, methods such as side-channel analysis, statistical, or specialized vector generation, called assistive, allow testing for specific aspects like Trojan detection.

Deployment

Finally, is the “Deployment phase” of the IC after the testing is completed and the correct functionality of the IC is certified. “Deployment phase” starts with the assembly in which the IC is inserted into its application. After that, the operation of the IC follows, which can be interrupted by failures and repair times. Once the deployment phase is completed, the disposal of the IC marks the end of its life cycle.

Attackers

So, now it is clear that all the phases of the life cycle that we analyzed above are vulnerable to attacks. These attacks differ significantly depending on the phase in which they occur, as well as the attackers involved. Attackers can be classified by various properties in the following basic categories:

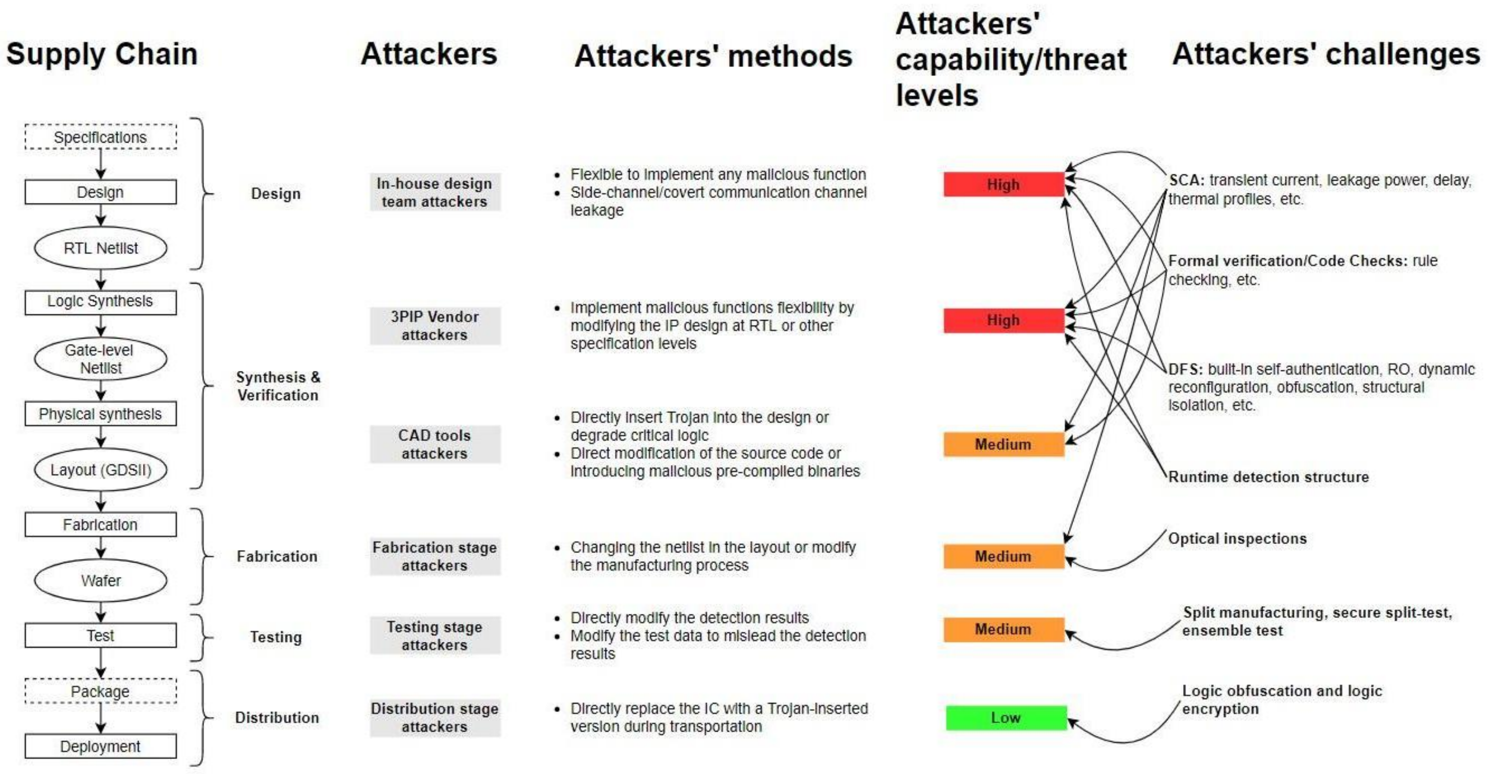
- the knowledge of the attacker, such as which information is available
- the expertise of the attacker, such as which skills are available
- the resources that are available to the attacker, both human and financial
- the function of the attacker such as designer, manufacturer, mask manufacturer, developer of software tools
- the sphere of the attacker, such as their influence, where they can become active, etc.

Moreover, Fig. 11 presents the attackers' categories based on the phase of the supply chain in which they occur. They are classified as:

- In-house design team attackers
- 3PIP Vendor attackers
- CAD tools attackers
- Fabrication stage attackers
- Testing stage attackers
- Distribution stage attackers

These attackers' categories are followed by:

- the methods each group of attackers is using,
- the capability/thread level of each group of attackers measured in a 3-level scale (High, Medium, Low), and
- the challenges each group of attackers is facing.



Supply Chain
Attackers
Attackers' methods
Attackers' capability/threat levels
Attackers' challenges
Specifications
Design
RTL Netlist
Logic Synthesis
Gate-level Netlist
Physical synthesis
Layout (GDSII)
Fabrication
Wafer
Test
Package
Deployment
Design
Synthesis & Verification
Fabrication
Testing
Distribution
In-house design team attackers
3PIP Vendor attackers
CAD tools attackers
Fabrication stage attackers
Testing stage attackers
Distribution stage attackers
Flexible to implement any malicious function
Side-channel/covert communication channel leakage
Implement malicious functions flexibility by modifying the IP design at RTL or other specification levels
Directly insert Trojan into the design or degrade critical logic
Direct modification of the source code or introducing malicious pre-compiled binaries
Changing the netlist in the layout or modify the manufacturing process
Directly modify the detection results
Modify the test data to mislead the detection results
Directly replace the IC with a Trojan-inserted version during transportation
High
High
Medium
Medium
Medium
Low
SCA: transient current, leakage power, delay, thermal profiles, etc.
Formal verification/Code Checks: rule checking, etc.
DFS: built-in self-authentication, RO, dynamic reconfiguration, obfuscation, structural isolation, etc.
Runtime detection structure
Optical inspections
Split manufacturing, secure split-test, ensemble test
Logic obfuscation and logic encryption

Figure 11: Attackers' categories based on the supply chain

Chapter 3: Malicious Hardware Analysis

The previous chapter focused on introducing malicious hardware and the different categories that can alter the hardware maliciously, resulting in catastrophic consequences. So, in this chapter, we present detection, localization, and prevention mechanisms of the previously mentioned taxonomy, diving deeper into malicious hardware analysis.

After many years of research that is still ongoing, there are many ways to deal with the risk of malicious hardware and, more specifically, Hardware Trojans. This section focuses mostly on hardware Trojans' detection, localization, and prevention mechanisms.

Detection determines whether a hardware system contains an unspecified functionality.

Localization determines the Trojan's topological (logical and/or physical) position if a detection mechanism diagnoses the examined system as compromised.

Prevention inhibits the inclusion of Trojans into hardware

Detection (18) (20)

This section describes the state-of-the-art Hardware Trojan detection techniques and classifies them based on how they are detecting malicious logic. Fig. 12 shows this exact classification. The majority of these techniques address Trojan detection in manufactured ICs. Significantly few researches have addressed it at a higher-level design since the detection technique most of the time is not easily scalable for larger designs.

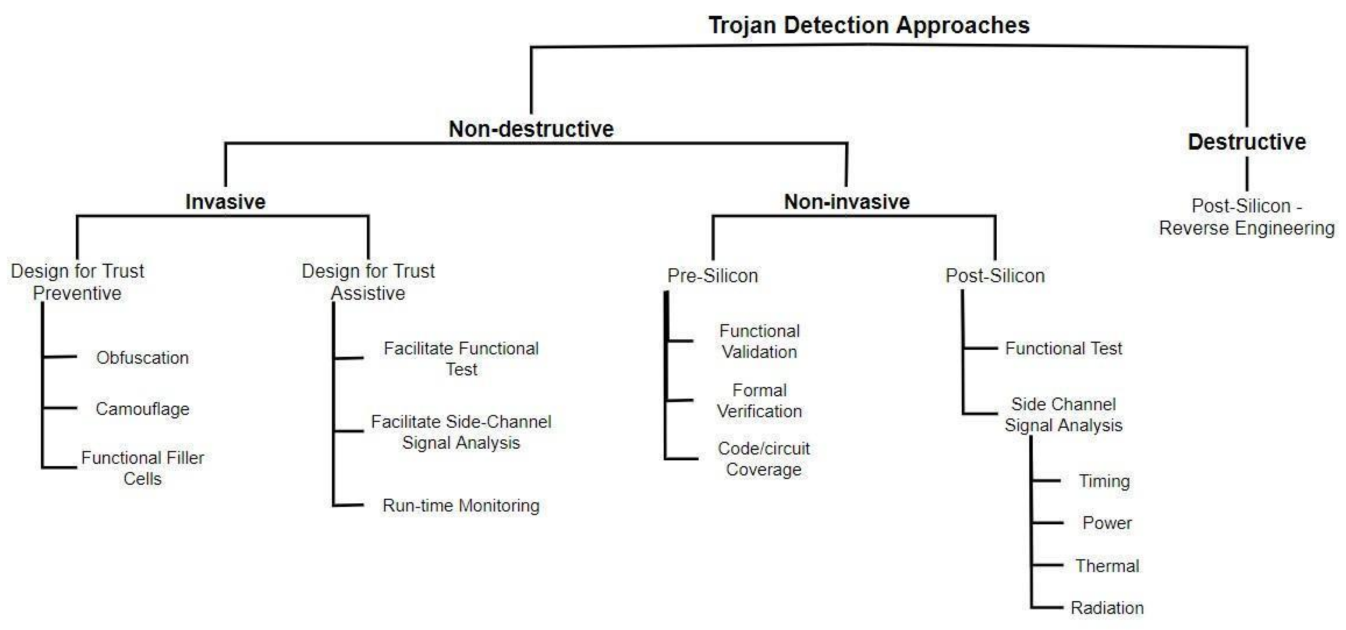


Figure 12: Hardware Detection Classification

The trojan detection techniques can be classified into two main types: destructive and non-destructive. Destructive reverse engineering techniques are being used to a sample of the manufactured ICs in order to test them, using Chemical Mechanical Polishing (CMP) followed by Scanning Electron Microscope (SEM) image reconstruction and analysis, and then obtain a Trojan free chip also known as Golden Chip (GC). However, such techniques are extremely expensive, time-consuming and the results cannot be generalized to all the manufactured IC's. Furthermore, testing only a small portion of the chips is ineffective since the adversary may affect only a small percentage of them. Non-Destructive techniques can be classified into two main sub-categories: Invasive and Non-Invasive. The invasive techniques alter the design in order to embed features for Trojan detection, while non-invasive techniques leave the design unmodified.

Destructive

As far as destructive techniques are concerned, one of the methods used is to apply sophisticated failure analysis techniques. Some of the techniques mentioned above are: scanning optical microscopy (SOM), scanning electron microscopy (SEM), pico-second imaging analysis (PICA), voltage contrast imaging (VCI), light-induced voltage alteration (LIVA), charge-induced voltage alteration (CIVA), etc. Although the above methods can be proven effective for authentication purposes, they are extremely time-consuming and expensive. Furthermore, many require the chip to be prepared by backside thinning and de-processing operations.

So, we can conclude that the above approach is not suitable for applications in which every chip needs to be authenticated. Another disadvantage is that many of these techniques are becoming ineffective for technologies in the nanometer domain. Moreover, the adversary can insert randomly Trojan into chips, so spending a significant amount of time on each chip for authentication will be prohibitively expensive. That leads us to conclude that new and efficient methods are required to detect Trojans with higher confidence levels and minimum authentication time.

Non-Destructive Invasive

A potentially more effective way to detect hardware Trojan is through non-destructive invasive techniques by dealing with the Trojan problem in the design phase through design-for-trust (DfT). The invasive techniques are further categorized into assistive and preventive. Assistive techniques facilitate the detection of inserted Trojans by using post-manufacturing tests, whereas preventive techniques prevent the insertion of Trojans during the design or fabrication of an IC.

Assistive techniques, which is the first class of design-for-trust (DfT) approaches, are classified into three categories according to their objectives:

- *Functional Testing*: Focuses on triggering a Trojan from inputs and observing the Trojan effect from outputs which is considered difficult due to the stealthy nature of Trojans. Moreover, the possibility of activating a Trojan is significantly prevented by the large number of low controllable and low observable nets in the design. However, some approaches exist that can facilitate the activation of a Trojan for functional-test-based detection techniques. One approach suggests increasing controllability and observability of nodes by inserting test points into the circuit. Another approach proposes multiplexing two outputs of a DFF (Delay Flip Flop), Q and $\overline{Q}$, through a 2-to-1 multiplexer and selecting either of them. This approach extends the state space of the design and increases the possibility of exciting/propagating the Trojan effects to circuit outputs, making them detectable. Both approaches are also beneficial to side-channel-based methods that need partial activation of Trojan circuitry.

- *Side-Channel Signal Analysis*: Several design methods have been developed to facilitate the side-channel-based detection methods. Such design methods and techniques can detect functional Trojans without activating them by measuring their secondary action characteristics. For example, functional Trojans are never completely inactive because of their need to continuously monitor for the activation conditions. Let's consider a Trojan that activates based on a specific state of a data bus in the chip. The implementation, in this case, needs some type of comparator to be installed to monitor the wires of the data bus. So as the data bus changes, the logic comparators, e.g., AND gates, switches and therefore consumes power. Thus, side-channel signal analysis can possibly detect the power anomaly introduced by the operation of the comparator. Other side-channels signals include electromagnetic field variations, temperature variations, voltage variations, etc., occurring at various locations across the chip. New methods can be developed that use such signals to detect and isolate hardware Trojans.

  Furthermore, the detection of tightly coupled functional Trojans can be succeeded even without applying a digital stimulus due to the highly sensitive nature of side-channel analysis techniques. The presence of the Trojan logic gates adds additional capacitance to the power grid, which sequentially changes the power grid's impulse response and can be tested by injecting an analog stimulus onto the grid at one place and measuring the response at another.

  The effectiveness of side-channel-based measurement and analysis techniques can be improved by adopting design-for-trust (DfT) techniques. This way, the circuitry can be added to support the measurement and analysis processes with the condition to incorporate a validation strategy for the on-chip support circuits due to the potential of the adversary to sabotage the sensors.
- *Runtime Monitoring:* Runtime monitoring of critical computations can significantly increase the level of trust concerning hardware Trojan attacks as triggering Trojans during pre-silicon, and post-silicon tests is very difficult. Runtime monitoring approaches can use existing or supplemental on-chip structures to monitor chips' behavior or operating conditions, such as transient power and temperature. Therefore, the chip can be disabled upon detecting any abnormalities or bypassing it to allow reliable operation, even though there will be some performance overhead.

Research has shown that an on-chip analog neural network can be trained to distinguish trusted from untrusted circuit functionality based on measurements obtained via on-chop acquisition sensors.

Preventive techniques, which is the second class of design-for-trust (DfT) approaches, are also classified into three categories according to their objectives:

- *Logic Obfuscation:* Aims at hiding the actual functionality and implementation of a design by inserting built-in locking mechanisms into the original design. Consequently, the locking circuits become visible, and the proper function appears only when the correct key is applied. This approach increases complexity in identifying the actual functionality without knowing the suitable input vectors. Thus, the attackers are having a hard time inserting a targeted Trojan. In sequential logic obfuscation, additional states are introduced in a finite state machine to conceal its functional states. In contrast, in combinational logic obfuscation, XOR/XNOR gates can be introduced at specific locations in a design.
- *Camouflaging:* Focuses on obstructing attackers from extracting a correct gate-level netlist of a circuit from the layout through imaging different layers, so the original design is protected from the insertion of targeted Trojans. Camouflaging can be defined as a layout-level obfuscation technique to create indistinguishable layouts for different gates by adding dummy contacts and faking connections between the layers within a camouflaged logic gate.
- *Functional Filler Cell:* Aims at filling all-white spaces during layout design based on the built-in self-authentications approach. When these cells are inserted, they are connected automatically to form a combination circuit that could be tested. A failure during later testing denotes that a Trojan has replaced a functional filler. That approach is used because layout design tools cannot fill 100% of the area with regular standard cells in a design.

Non-Destructive Non-Invasive

A different approach in order to detect Trojan is by comparing the behavior of the test IC with a golden functional model or the golden IC instance. Those techniques can be further classified into two main types: run-time and test-time. The run-time techniques utilize an online monitoring system that tries to detect suspicious activity during in-field operations. On the other hand, the test-time techniques aim to detect Trojan-infected chips before deployment. Diving deeper into the above two approaches, we can better understand the methods used for each.

- *Run-time:* Many researchers are proposing different approaches regarding run-time techniques. One method includes the addition of reconfigurable Design for Enabling Security (DEFENSE) logic in a given SoC to enable live functionality monitoring. Once a deviation from normal functionality is detected, appropriate countermeasures are triggered instantly due to the fact that the checks can be performed concurrently with the normal circuit operation. However, it is not yet known how effective the above approach is and how the performance of the hardware is affected. (21)

  Furthermore, researchers also present a novel SoC bus architecture that can identify malicious bus behaviors associated with Trojan hardware, defend the system and system bus from them, and report the malicious behaviors to the system CPU without losing bus performance. The authors of the particular work report an additional gate-count of about eight hundred (800) logic gates in a four million gate SoC and negligible delay overhead. (22)

  Finally, the authors at (23) and (24) propose a combined hardware-software approach to perform run-time execution monitoring. A simple, verifiable “hardware-guard” module external to the CPU is considered in this approach. It mainly targets DoS and privilege escalation attacks, using periodic checks by the operating system (OS) enhanced with live check functionality. This research reports a 2.2% average performance overhead using SPECint2006 benchmark programs but does not report the hardware design overhead.
- *Test-time:* Testing-based approaches are classified into two main classes regarding Trojan detection: logic testing-based techniques and measurement of side-channel parameters-based techniques such as power, delay, temperature, and radiation.

  Logic testing-based approaches face a serious challenge: the enormously large Trojan space, which makes computationally infeasible the generation of an extensive set of test vectors to detect all possible Trojans. Hence, a statistical approach seems more suitable for test vector generation in this case. Proposed approaches by researchers include but are not limited to:

- ❖ Randomization-based techniques to probabilistically compare the functionality of the implemented circuit with the design of the circuit
- ❖ Statistical vector generation technique that targets the creation of an ideal set of test vectors that can activate each rare node in a circuit to its rare value multiple times

Side-channel analysis-based approaches for Trojan detection focus on observing the impact of an inserted Trojan on a physical variable such as circuit current transient, power consumption, or path delay. The apparent advantage of this approach lies in the fact that even if the Trojan circuit does not cause observable malfunction during testing, the presence of the extra circuitry can be reflected in some side-channel parameters. Proposed approaches by researchers include but are not limited to:

- ❖ IC fingerprinting technique, where each IC instance is associated with a signature called a "fingerprint" obtained by measuring one or more side-channel parameters.
- ❖ Gate-level characterization technique, where the detection problem was formulated as Linear Programming Problem. For this technique, both path delay and leakage current were considered.

To conclude, we have summarized the advantages and disadvantages of each approach (logic-based testing and side-channel analysis testing) in the table below. This table shows that these two approaches have complementary scope in terms of Trojan detection capability. So, approaches that combine the best of both worlds can be considered as the most promising.

	Logic Testing Approach	Side-channel Approach
Advantages	● Effective for small Trojans ● Robust under process noise	● Effective for large Trojans ● Test generation is easy
Disadvantages	● Test generation is complex ● Large Trojan detection challenging	● Vulnerable to process noise ● Small Trojan detection challenging

Localization

After making use of one or more (maybe even a combination) of the detection methods analyzed above, and malicious circuits are detected in the hardware, the next step is their localization. Localization is the regional detection of a malicious circuit and has a variety of mechanisms. The current localization mechanisms are activation, mensuration, or calculation, as shown in Fig. 13.

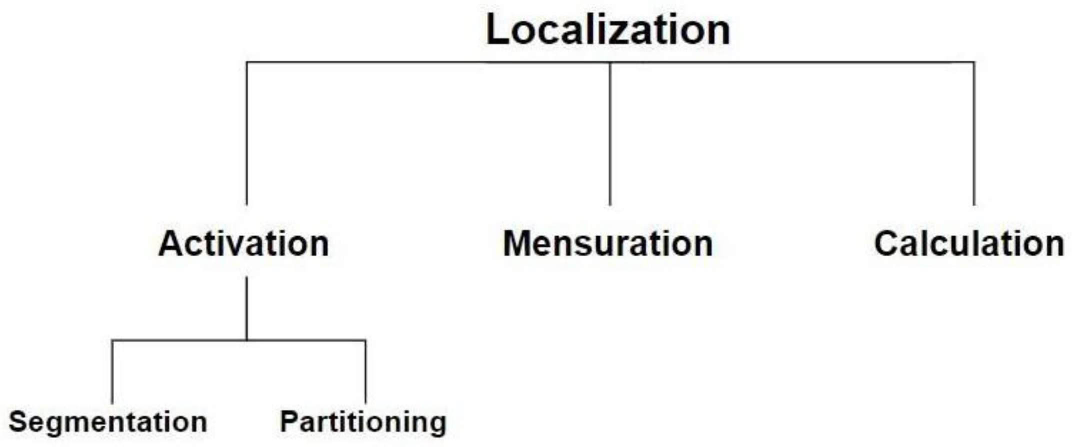


Figure 13: Localization mechanisms

Activation

The activation method to localize hardware Trojans is achieved by activating a subset of gates during the functional test. The purpose is to maximize the activity in certain areas of the circuit while minimizing it in the remaining ones. This method can succeed in two ways, either by segmentation or partitioning. Segmentation is the fragmentation of the circuit in the state space, whereas partitioning is the division of the circuit according to its structural composition.

- *Segmentation:* Segmentation is defined as the functional partitioning of a circuit into subcircuits. This functional partitioning could be achieved by various input vectors that activate specific gates during a functional test. Subsequently, if this specific sequence of test vectors manages to activate a Hardware Trojan, then it is likely to reside in the set of gates that we "segment" and was addressed by that sequence.

  Researchers at (25) follow an approach that finds a subset of gates (a segment) by their controllability by primary inputs. The purpose is to generate segments whose gates will have good scaling factors in a subsequence Gate-Level-

Characterization (GLC). First, some primary inputs are varied while the rest of the inputs remain constant. Then, for the gates that are controlled by the primary inputs, a GLC is performed—so based on the accuracy of the GLC and the characteristics of the segments, a regression model is created to predict the accuracy of the values of the GLC for other unknown segments. Finally, Trojans are detected through a side-channel analysis using GLC, and when a Trojan is detected, the segment in which it was found gives information about its logical position. GLC produces appropriate results in simulations which are, in practice, hard to achieve, as it requires precise control over temperature and accurate measurements.

Another research at (26) uses a different technique which keeps a test vector constant for several clock cycles. This method is intended to keep the activity of the rest of the circuit low so that any activity should only originate from inner state changes (see Fig. 14(a), 14(b)). To identify Trojans, the difference in power consumption compared to a non-tempered golden model is determined through side-channel analysis. A deviation of more than 5% (to account for process variations) might indicate additional circuitry in the vicinity of the logic gates that were stimulated by the test vector.

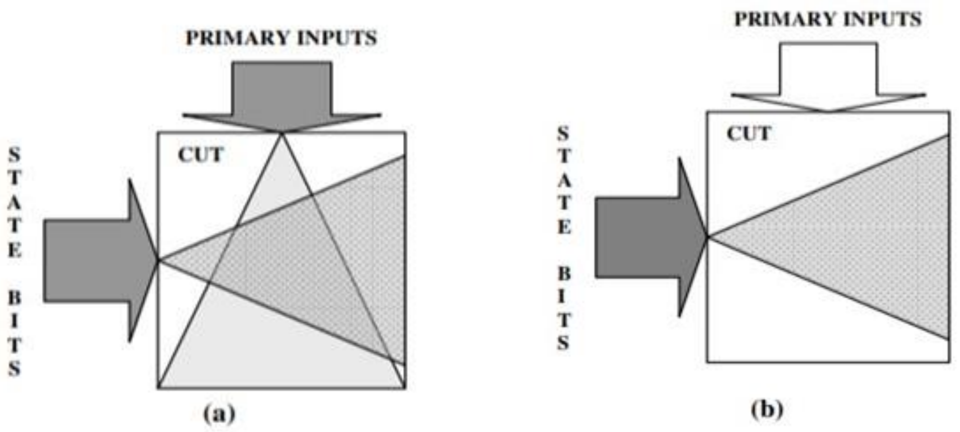


Figure 14: Concept of activity minimization. In (a) circuit activity is created by both flip-flops and PIs whereas in (b) only by flip-flops

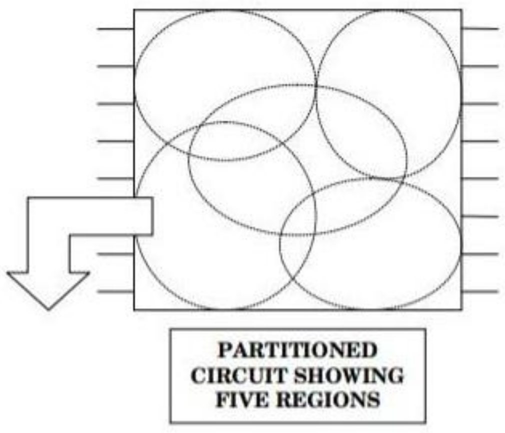
- *Partitioning:* Partitioning is the structural division of the circuit into several subcircuits. The first approach to analyze is based on the Hamming distance of state variables. The Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different.

Simply put, it measures the minimum number of errors that could have transformed one string into the other or the minimum number of substitutions required to change one string into the other.

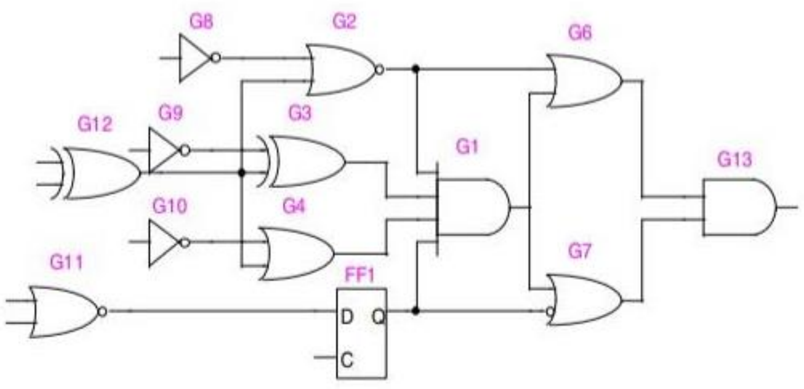
Research at (27) seeks to generate test patterns that activate a particular part of the circuit (partition) while minimizing the activity of other partitions. As a result, the fan-out cone activity of the flip-flops in the partition is low. If a Trojan is identified during a side-channel analysis, it is located in the partition that was active during the test.

As an alternative, the same authors propose a partition method based on the maximization of the toggle count instead of the Hamming distance (19).This is because the Hamming distance does not necessarily permit the drawing of conclusions on the power consumption of a tested circuit. In Fig.15, we can see a sample partitioning of a circuit. More specifically, the partitions of a circuit (Fig. 15(a)) result from a fixed radius. A radius of 0 means that only one specific gate is included in the partition. A radius of 1 contains a specific gate and the gates upstream and downstream from it that are included in the first level. A radius of 2 extends to the second level, etc.

For example, in Fig. 15(b), the following gates belong to a radius of 1: G1, G2, G3, G4, FF1, G6, and G7. Trojans are assumed to be connected to circuit components that fulfill a specific functionality. Flip-flops and their gates execute this functionality. Therefore, a flip-flop threshold is defined that determines how many flip-flops are present in a partition. In other words: the radius of a partition is extended until the appropriate number of flip-flops is contained in the partition. Afterward, test patterns are generated to maximize activity within the partitions while the activity in the rest of the circuit is minimized. If a Trojan is subsequently detected through side-channel analysis, it is located in the partition whose test pattern recognized the Trojan.



(a)



(b)

Figure 15: Illustration of the concept of Region and Radius in a circuit

Mensuration

Parts of a circuit that have been tampered with can also be localized using the measuring setup that is used for detection. Multiple methods can be used to capture the side channels that are subsequently evaluated by side-channel analysis.

Researchers at (28) capture IDDT across multiple power ports scattered across the entire chip surface (Fig. 16). The measured current might give clues about the presence of a Trojan. If a significant deviation from IDDT is observed near a power port, a Trojan is suspected in its proximity.

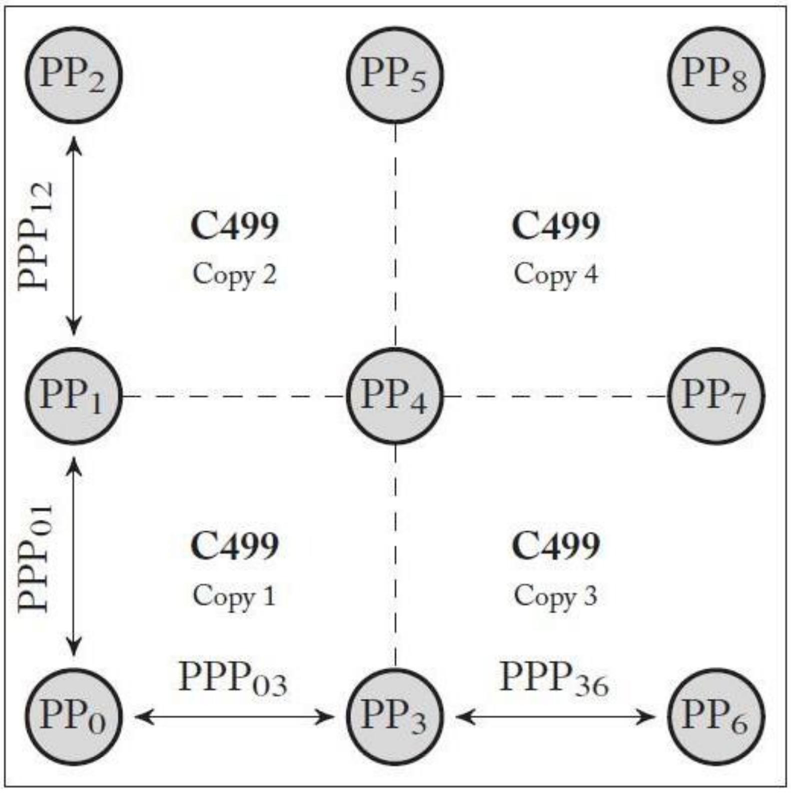


Figure 16: Architecture of Simulation Model of paper (28)

Furthermore, other research (29) uses a network of ring oscillators distributed over the entire chip area to determine measurement variables. Fig. 17 shows the arrangement of the ring oscillators as well as the locations of several implanted Trojans. If Trojans are built into the chip, they alter the frequency of neighboring ring oscillators. This fact can be used to determine the spatial position of a Trojan.

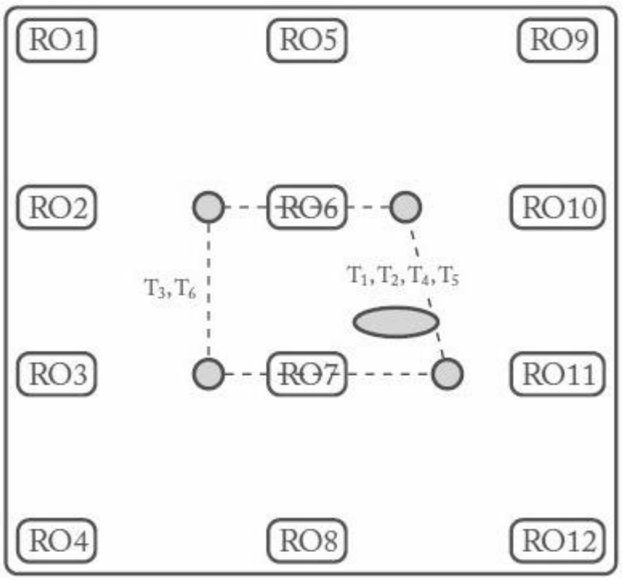


Figure 17: Arrangement of ring oscillators and Trojans on the chip surface of paper (29)

Calculation

A Trojan can also be localized by using mathematical methods.

Such a method is GLC (Gate-Level Characterization), which characterizes circuits at gate level. Consequently, it means that scaling factors are determined for each individual gate of a circuit in order to calibrate for process variations. If scaling factors differ strongly from their expected values, the presence of a Trojan can be assumed. Because a Trojan influences the scaling factors of the gates to which it is connected, this can be a good clue for its localization. (30)

Prevention

In contrast to detection and localization, prevention is the action of stopping Trojans and their actions from happening. Trojan prevention methods have also been explored to ensure hardware security. Prevention methods can be classified into two main categories: Obfuscation and Invasion (Fig. 18).

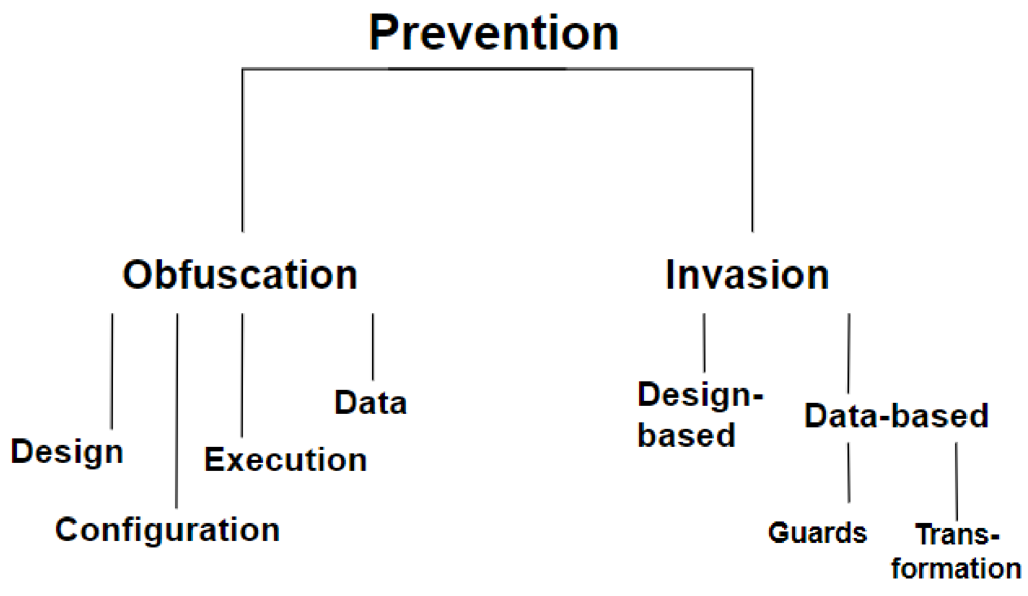


Figure 18: Prevention methods taxonomy

Obfuscation

Obfuscation is a technique by which the structure or the description of the electronic hardware is modified to intentionally disguise its functionality, which makes it significantly more difficult to reverse-engineer. In other words, hardware obfuscation modifies parts of a hardware system so that the resulting architecture and functions become un-obvious of an adversary. The parts of a hardware system that can be modified are:

- Design

  A hardware system is obfuscated at the design level in order to make it more difficult for the adversary to reverse engineer the function of the IC.

  At (31), the authors propose, in order to obfuscate design at gate level, an approach that changes the original functionality of the design by modifying the netlist. An additional state machine is inserted with two execution modes: an obfuscated and a non-obfuscated mode. At the beginning of operations, the state machine is in the obfuscated mode and remains there until a specific pattern is applied to the primary inputs. While in obfuscated mode, invalid data is returned on the primary outputs. When the change from obfuscated to non-obfuscated mode occurs, a transition to a state of the original state machine takes place. This method can prevent the insertion of Trojans because it makes reverse engineering, and therefore the insertion of malicious circuits by an attacker, significantly more difficult.

  Furthermore, another method is introduced at (32) that inserts additional inputs and outputs and an additional state machine. Authentication keys are generated by the additional circuit components, which will spread through the entire hardware system. This is done in a special execution mode which is activated at test time. The authors note that it is more difficult for an attacker to reverse engineer the original functionality because of the added obfuscation through the additional circuit components.

  The authors present an approach for obfuscation that splits an existing design into two state spaces at (33). One state-space implements the original functionality, and the other implements the obfuscated functionality. The starting point of all operations is the obfuscated state space. The transition from obfuscated to normal state space can only be triggered by a very rare condition. Therefore, an attacker who reverse engineers a design cannot find a suitable place to insert the malicious circuit safely. Either a Trojan is added to the obfuscated state space—which renders the Trojan harmless—or it is linked to a supposedly difficult to reach state, increasing its visibility in functional tests. The starting point is a state in the obfuscated space, the initialization state space. A unique sequence of states must be traversed to reach the normal state space, which is only possible if specific values are provided as primary inputs. This sequence of values is called the initialization key. If an invalid key is provided during the initialization phase, a transition into a state is enforced that is part of an isolated state space, which cannot be left. In this way, potential Trojans are rendered harmless.

Moreover, at (34) is presented a method to not only ease detection of hardware Trojans but to obfuscate a design against Trojan implementations partially. The main goal of this approach is to increase the number of reachable states initially and then partition the flip-flops into different groups to enhance the state-space variation. In their approach, the state-space variation operates as an obfuscation technique.

- Configuration

  Another way to achieve obfuscation is by configuration. In order to achieve it, one part of the circuit should be realized as reconfigurable logic, i.e., as Field-Programmable Gate Array (FPGA). Essential functions of the hardware system are implemented into this reconfigurable part of the chip. The functionality is added to the reconfigurable logic after the manufacturer provides the chip and inserts it into the application. The above functionality will likely be added during the deployment phase while assembling the system.

  Fig. 19 below shows the approach that (21) has implemented. As shown in Fig. 19, the system intended to avoid attacks altogether or prevent successful attacks from executing. Signals will be tapped by signal processor networks (SPN) which are monitored by security monitors (SM). A security and coordination processor (SECOPRO) configures SPN (which signals should be checked), as well as SM (which tests are done). If the SECOPRO detects unauthorized behavior, it can influence the signals of the hardware system by using signal control. All configurations are stored securely in an encrypted flash memory that makes the reverse engineering of the logic by an attacker virtually impossible. The actual configuration is only taken during operation. The manufacturer does not know its shape and cannot make any changes.

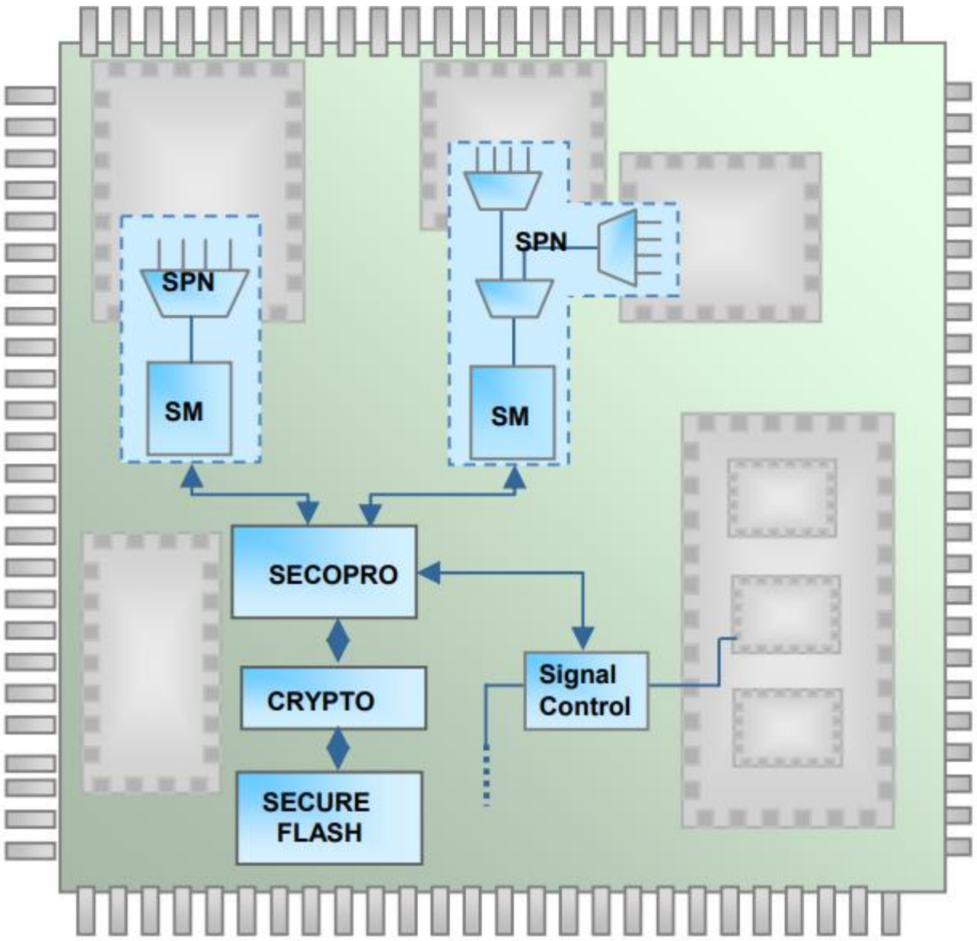


Figure 19: Prevention by obfuscation using reconfigurable logic based on (21)

- Execution

  The execution of software on non-trustworthy hardware is another way to achieve obfuscation.

  Researchers at (35) introduce a method that uses a multi-core system on which different cores are running instances of functionally equivalent software (different algorithms or different compilations of the same algorithm). Their approach is based on the assumption that a Trojan is activated on the occurrence of a rare event. It is unlikely that two different versions of functionally equivalent software satisfy the same rare condition and activate the same Trojan. The described concept is illustrated in Fig. 20 below.

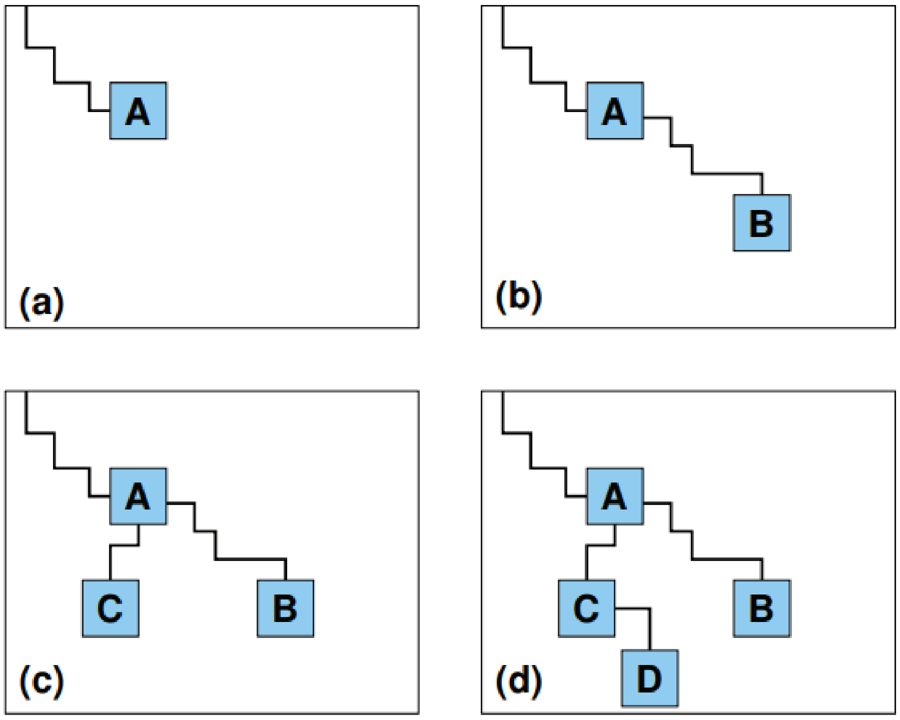


Figure 20: Prevention by obfuscation using different variants of functionally equivalent software based on (35)

Initially, two different versions A and B, are executed on two cores of the hardware system (Fig. 20(a) and 20(b)). If the results V_A and V_B of both executions are the same ($V_A = V_B$), it can be assumed with great certainty that no Trojan has been triggered. If the two results differ ($V_A \neq V_B$), it is assumed that a Trojan has been triggered on one core. In order to find in which core the Trojan is activated, another variant of the task, C, is started (Fig. 20(c)). If the result from the execution of V_C is the same as one of the other results, then the Trojan was activated in the core, where the execution delivers a different result. If all three results are different ($V_A \neq V_B \neq V_C$), another variant D is started (Fig. 20(d)), and so on. Aiming to avoid executing tasks on compromised cores, the level of trust is reduced for each core on which a Trojan is detected. Reducing the trust of a compromised core allows a scheduler to avoid allocating one. The same authors introduce such a scheduler in (36).

- Data

  Lastly, during the operation of a hardware system, in order to prevent a Trojan's payload, data can also be obfuscated. The basic idea is that the trigger condition for a Trojan cannot occur.

  Authors at (13) define two types of triggers that can activate a Trojan:

  a. single-shot cheat codes and
  b. sequence cheat codes

  In addition, they define two types of functional units of a hardware system:

  a. computational units and
  b. non-computational units.

  Data is obfuscated at inputs to untrusted functional units and de-obfuscated at their outputs (Fig. 21(a)). Single-shot cheat codes can be obfuscated by simple encryption (such as bitwise XOR or the addition of a random value, see Fig. 21(a)). However, this is only possible for non-computational functional units. For computational units, homomorphic functions are used.

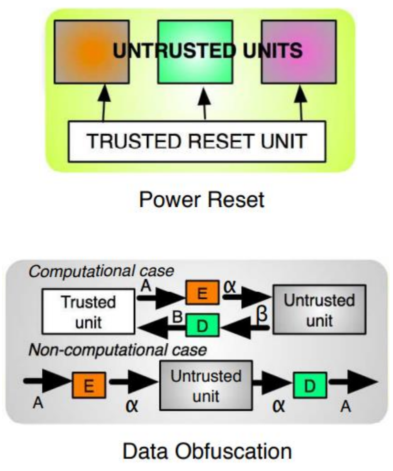
  A function f is homomorphic with respect to another function g if:

$$f(x)g(y) = g\big(f(x,y)\big)$$

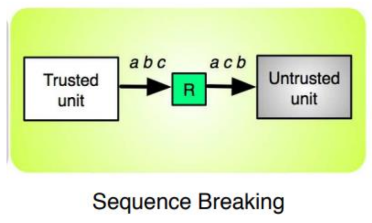
  A simple example is the function in the below equitation:

$$x^2y^2 = (xy)^2$$

  Suppose a functional unit implements the squaring function (see Equation 3.4). To obfuscate the original value x, it is multiplied by a random value y. The functional unit calculates $(xy)^2$. For de-obfuscation, the result of the functional unit must be divided by the square of the random value y^2, which gives the original result of the functional unit, x^2. In this way, a valid trigger condition should be prevented from occurring for a Trojan within a functional unit. To obfuscate sequences of values, data streams are rearranged in a way that prevents the occurrence of a trigger condition caused by the sequence (Fig. 21 (b)). In cases where it is not possible to rearrange the data stream, dummy values are inserted to break the sequence.



(a) Obfuscation of single-shot cheat codes



(b) Obfuscation of sequence cheat codes

Figure 21: Prevention by obfuscation using simple encryption and reordering of data based on (13)

Invasion

Invasion is a technic to prevent Trojans by inserting additional functionality into a hardware design. Invasion can be based on the design of a hardware system or the data to be processed. If an invasive approach is design-based, a hardware design is altered so that Trojans can be defended during operation. If the data to be processed is monitored during operation, the invasive approach is data-based. A data-based invasive approach can be implemented in two ways with:

- Guards, when the added circuitry monitors the authenticity of data,
- Transformation, when the added circuitry modifies data of an application to render them useless for a trigger condition.

Below we analyze further the aforementioned invasive approaches for the prevention of Trojan:

- Design-based

  A design-based invasive approach to prevent Trojans modifies the original design.

  Authors at (37) propose the introduction of a separate security layer for 3D hardware systems. 3D hardware systems are hardware systems that consist of several stacked and interconnected layers. The security layer is used to perform security tasks such as cryptography. The possibility of defending against hardware Trojans is mentioned briefly but details are left for future research.

  Researchers at (1) replace suspicious components of a circuit by software emulation. Suspicious components are determined during the functional test at design time. A functional unit counts as suspicious if it is not activated during the entire testing process and could therefore act as a trigger for a Trojan. This method is called Unused Circuit Identification (UCI). So, if the input and the output of a functional unit have the same value during the entire testing process, the component counts as inactive and can be replaced by a short circuit. To still be able to maintain the functionality of the hardware system, suspicious units are replaced by detection hardware and the original function is emulated by BlueChip software. BlueChip software identifies and removes suspicious circuits and inserts runtime hardware checks. These hardware checks raise software exceptions to provide the BlueChip software an opportunity to progress the computation by emulating instructions, despite the fact that BlueChip may have removed legitimate circuits. Fig. 22 below illustrates the principle.

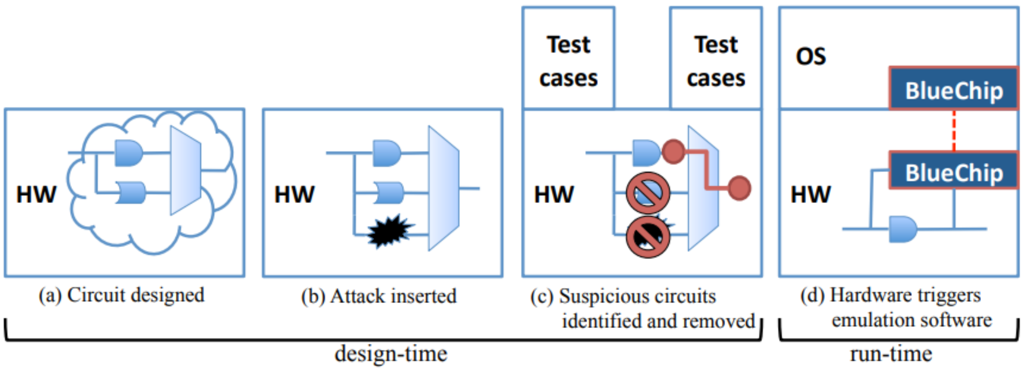
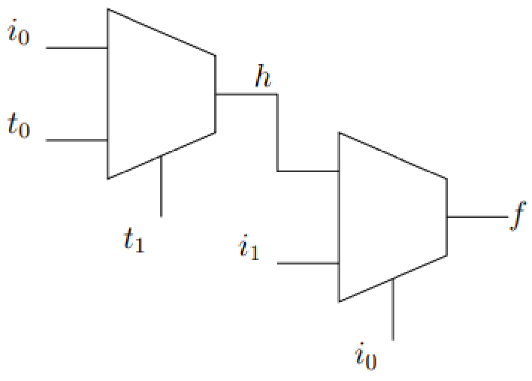


Figure 22: Principle of preventing Trojans by software emulation based on paper (1)

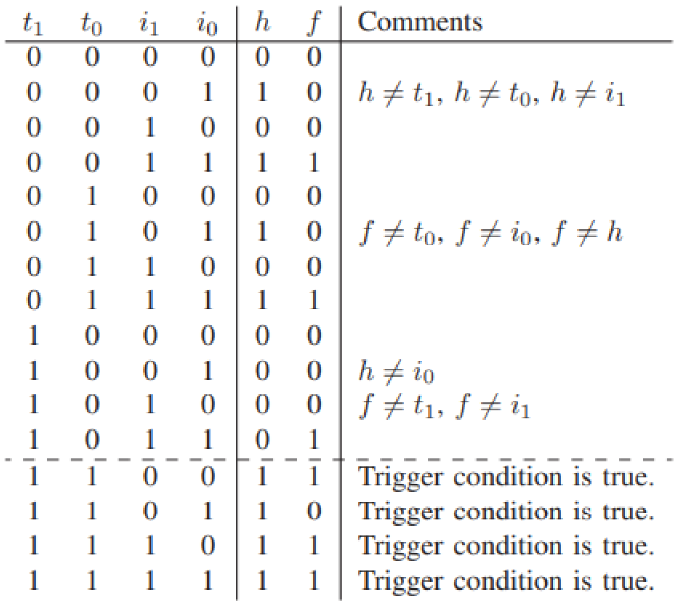
Lastly, authors at (4) proved that circuits exist that are classified as non-suspicious by UCI but can still be used as Trojans. The research looks for circuits that meet the conditions of the UCI but are malicious nonetheless. After introducing constraints (maximum number of gates, gate availability, maximum number of input-outputs, etc.), every possible combination of these gates is examined to determine whether this circuit is:

1. admissible,
2. obviously malicious,
3. stealthy.

If a circuit meets these conditions, it can execute malicious functions and is still not recognized by UCI. Fig. 23(a), for example, shows such an example circuit representing an AND gate with inputs i_0, i_1 (non-trigger inputs) and t_0, t_1 (trigger inputs). The trigger condition is $t_0 \wedge t_1$ and the output function f in a non-triggered state is $f_{NT} = i_0 \wedge i_1$. So, when the circuit is in the triggered state ($t_1 = t_0 = 1$), it fulfills the function f as shown the corresponding truth table (Fig. 23(b)). This circuit evades detection by UCI because of the way the circuit was constructed. There is no intermediate function equivalent to $i_0 \wedge i_1$ that UCI can short-circuit the output with. This idea of creating a non-trigger function which is not equivalent to any internal function is the key to defeating UCI.



(a) Two-gate, stealthy, admissible, and malicious circuit



t_1	t_0	i_1	i_0	h	f	Comments
0	0	0	0	0	0	
0	0	0	1	1	0	$h \neq t_1$, $h \neq t_0$, $h \neq i_1$
0	0	1	0	0	0	
0	0	1	1	1	1	
0	1	0	0	0	0	
0	1	0	1	1	0	$f \neq t_0$, $f \neq i_0$, $f \neq h$
0	1	1	0	0	0	
0	1	1	1	1	1	
1	0	0	0	0	0	
1	0	0	1	0	0	$h \neq i_0$
1	0	1	0	0	0	$f \neq t_1$, $f \neq i_1$
1	0	1	1	0	1	
1	1	0	0	1	1	Trigger condition is true.
1	1	0	1	1	0	Trigger condition is true.
1	1	1	0	1	1	Trigger condition is true.
1	1	1	1	1	1	Trigger condition is true.

(b) Truth table of two-gate circuit

Figure 23: Two-gate circuit design and truth table

Experiments for a maximum of three gates, each with two inputs and one output, as well as a gate library of {AND,OR,NOT,NAND,2-input-MUX} for two trigger inputs and one or two non-trigger inputs yield 27 circuits with different Boolean functions from 256 possible ones. The authors further reported a decrease in detection rate the more test cases were checked by UCI. This phenomenon is explained by an increase in false negatives and a decrease in false positives. However, the more test cases are examined, the higher the risk that a Trojan is detected during a functional test. Therefore, attackers have to find the perfect balance between bypassing UCI and staying hidden from functional tests. To still be able to test hardware for harmful components at design time, the authors propose to define a class of malicious circuits. One can search explicitly for such circuits in the course of a functional test. However, test methods that use test cases as their only specification for correct behavior (such as UCI) are not suited for this purpose, because the specification might be incomplete.

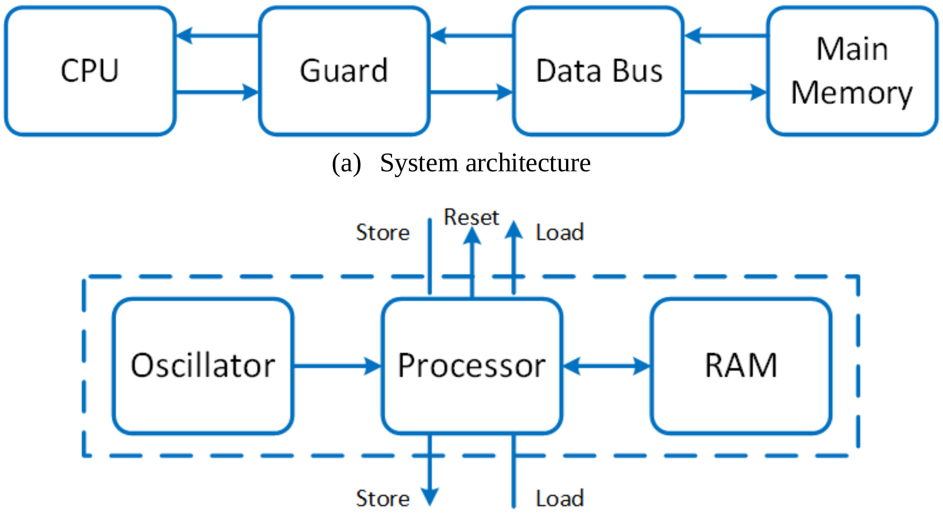
- Data-based – Guards

  A data-based invasive approach can be implemented as a guard. The job of a guard is to check and verify processed data for authenticity. If it recognizes that data is invalid or that data is being accessed without permission, it can initiate defensive measures.

  Authors at (22) and (38) propose Trojan-resistant bus architecture with a functionality that is based to a large extend on data guards. The proposed architecture offers protection against malicious bus masters and malicious bus slaves. Malicious bus masters access memory without authorization. If this is detected, the master is denied access to the bus by the secure address decoder. Malicious masters can also block the bus by locking it and issuing a lock signal so that other masters can no longer access the bus. This is prevented by a maximum time span for which a master can request exclusive bus access. Once this time span is over, the secure bus arbiter revokes the access of the bus master to the bus. Malicious bus slaves can block the bus by continuously sending a wait signal. This problem is addressed in the same manner as a continuous lock signal: if a bus slave claims the bus for too long, its access is revoked by a secure bus matrix. If a Trojan is detected, an interrupt signal is sent to initiate further defensive measures.

Moreover, authors at (23) introduce a data guard to prevent Denial-of-Service (DoS)-attacks. The data guard has a simple design so that its authenticity can be verified with simple measures. In addition, it is located off-chip. The proposed approach should protect a computer system from

DoS-attacks. To counter DoS-attacks, the guard checks if the higher-level operating system is still running. For this purpose, the guard monitors all accesses to the memory (Fig. 24(a)). The operating system sends a liveness check to the guard every 1µs. The liveness check is implemented as pseudorandom non-cached memory access. The guard detects the liveness check and sets an internal watchdog timer to a pseudorandom value. If the watchdog times out, it assumes that the operating system is no longer running—a DoS-attack is detected. The CPU is reset to allow further operation. Fig. 24(b) shows implementation details of the guard. An oscillator is used as internal timer, RAM is used to store pseudorandom values. A reset signal is used to reset the CPU or to inform a higher-level instance of attack. A processor takes control and monitors the memory bus (load and store signals).



(a) System architecture

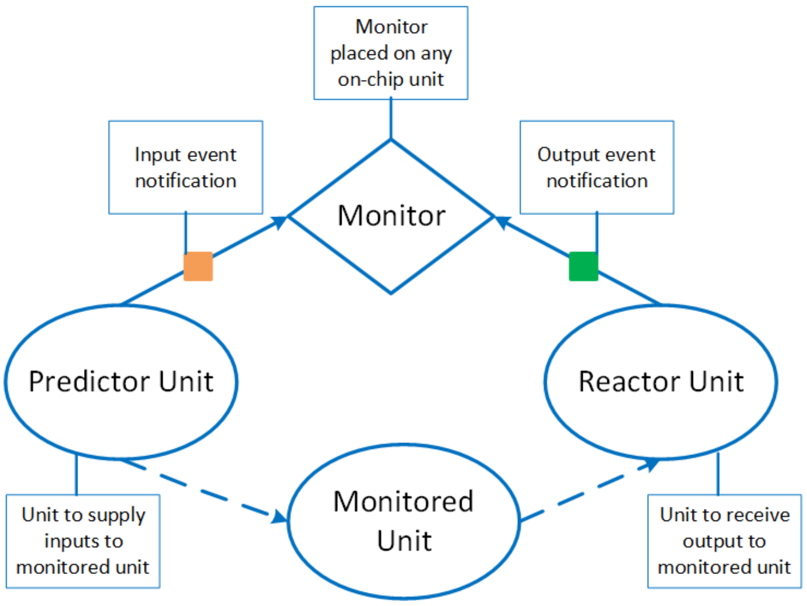
(b) Implementation details of the guard

Figure 24: Data guard to block malicious access to the memory bus based on paper (23)

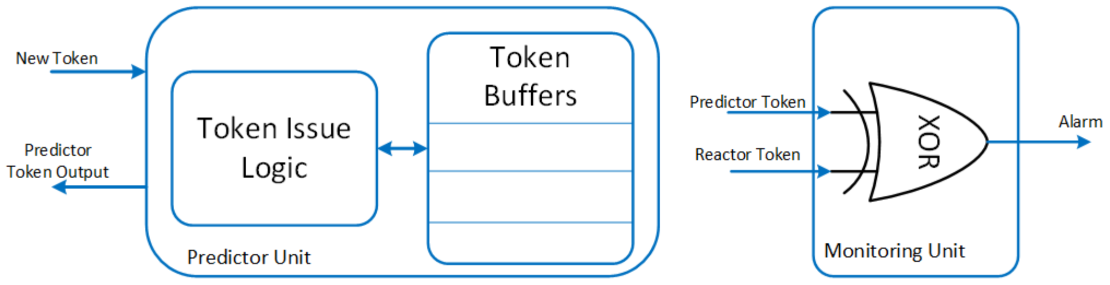
Furthermore, researchers at (39) present a predictor/monitor/reactor-model, which checks the functional units of a hardware system for authenticity (Fig. 25(a)). They present an approach to combat Trojans especially in microprocessors at runtime. However, in their paper the design process is not trusted: they address the possibility that malicious functionality is injected during the design phase by malicious designers. The following constraints are defined:

1. the number of malicious designers is low,
2. the activity of malicious designers remains unnoticed,
3. the attackers need few resources to inject a backdoor,
4. the backdoor is activated by a trigger and
5. the Read-Only Memories (ROMs) written during the design phase contain correct data(microcode)

Two types of backdoors serve as a Trojan model: emitter (send data) and corruptor (modify data) backdoors. The latter are very difficult to detect, since it can be hard to distinguish their operations from normal, legitimate operations. The measure proposed for preventing the hardware backdoors is an on-chip monitoring system that consists of four parts: predictor, reactor, target, and monitor (Fig. 25(a)). Fig. 25(b) shows details of the internal structure of the predictor and the monitor units. The monitor takes on the role of guard by comparing predicted values with actual results. If the values do not match, a Trojan attack is assumed. An alarm is issued to initiate countermeasures.


Monitor placed on any on-chip unit
Input event notification
Monitor
Output event notification
Predictor Unit
Reactor Unit
Monitored Unit
Unit to supply inputs to monitored unit
Unit to receive output to monitored unit

(a) Predictor, reactor, target object, monitor



(b) Implementation details

Figure 25: TRUSTNET Monitor Overview & Microarchitecture based on paper (39)

In another research (40) authors base their work on the assumption that data always have to pass through the main memory before it can leak. Thus, if malicious access to the memory can be prevented, the leakage of data can be prevented. This is achieved by replication of memory operations which are performed by a software application. The source code of the application is altered to create two simultaneous data streams: the original one and a parallel one, for which the addresses are generated using a secret key (see the left part of Fig. 26). Both data streams are compared to each other, making it possible to detect if the original data stream is manipulated by a Trojan. The data streams are compared by an external guard, which is placed between the CPU and the main memory (see the right part of Fig. 26). The approach is a combined hardware-/software solution. The software is manipulated by the end-user to create a second data stream with its own keys. The guard core is also programmable, so that the end-user can determine its implementation herself.

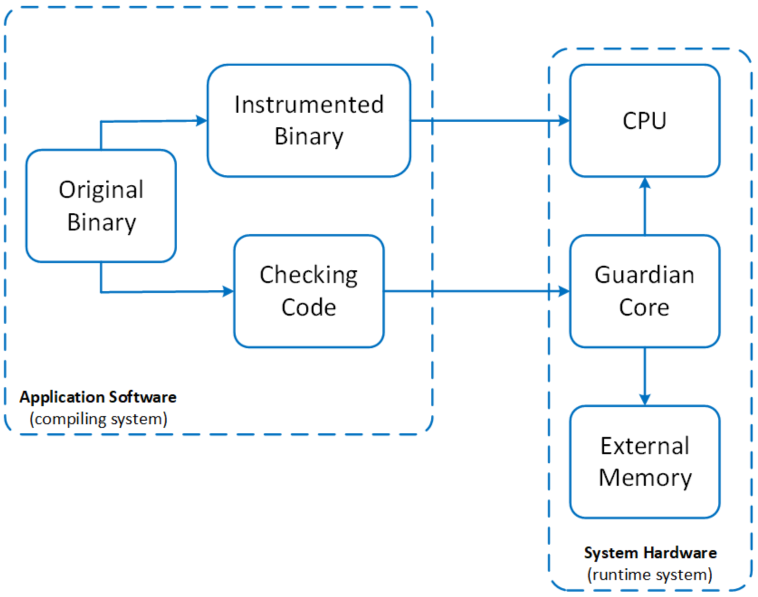


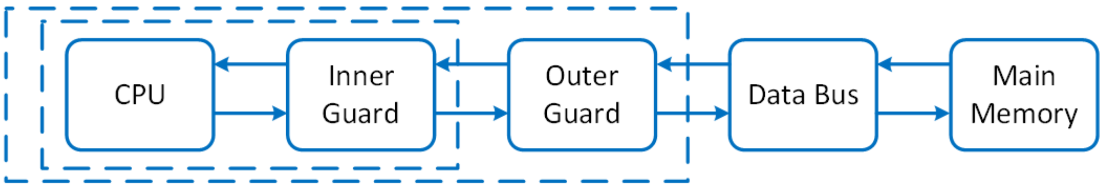
Figure 26: Guardian core to block malicious access to the memory bus based on paper (40)

- Data-based – Transformation

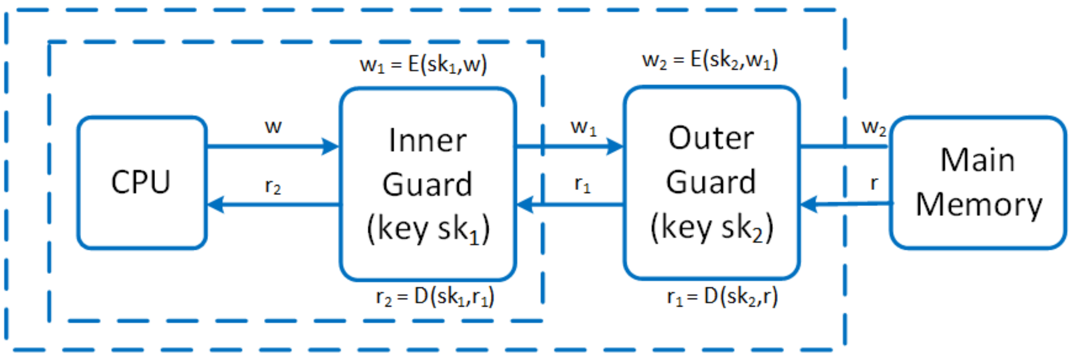
  Another possibility for a data-based invasive approach is to transform the processed data in a way that renders them unusable for a Trojan. The goal is to prevent the trigger from being activated.

  Authors at (13) use two types of transformation: encryption and permutation. To implement the encryption, additional hardware modules need to be included in the original design. Since the mechanisms can be kept very simple, simple blocks can be inserted at the inputs of computational units for encryption. For decryption, they can be placed at their outputs. In order to permute data within a data stream, the memory control is altered in a simple manner.

  Furthermore, authors at (24) present an invasive approach which uses dual encryption. Fig. 27(a) shows the architecture of the approach. An external and an internal guard are implemented, both located between the CPU and the main memory (Fig. 27(a)) to defend against attacks to leak data from the main memory. The guards each use a different key sk_1 and sk_2 (Fig. 27(b)). The transformation is performed within the guards. If data *w* should be written to the main memory, it is encrypted by the internal guard ($w_1 = E(sk_1,w)$). The external guard encrypts the output of the internal guard ($w_2 = E(sk_2,w_1)$). The dually encrypted data is then passed on to the main memory. Read access takes place in the opposite order: first data *r* is decrypted by the external guard ($r_1 = D(sk_2,r)$) and then by the internal guard ($r_2 = D(sk_1,r_1)$). The memory itself contains only encrypted data. The defense against attacks that leak data is achieved as follows: if the CPU or the internal guard is compromised, the attack can be defeated by the external guard. Since the outer guard can be made reconfigurable, it is very difficult to compromise.



(a) Architecture



(b) Dual-encryption details

Figure 27: Prevention by transformation using dual encryption based on paper (24)

Chapter 4: Malicious Hardware Use Cases

In this chapter we will focus on describing different malicious hardware use cases that can subvert the system through hardware attacks and remain stealthy. We will present four examples of malicious hardware use cases that use different attack vectors and most of them, if not all, manage to remain undetected from protection software.

Chipset Level Backdoor

A chipset is a set of electronic components that manages the flow of data between components on the motherboard or components on an expansion card. Authors at (41) introduce and describe explicitly a proof-of-concept chipset level rootkit/network backdoor based on Intel's chipset 8255x Ethernet Controller. The aforementioned backdoor has the ability, due to its low-level position in the computer system, to virtually bypass all commodity firewall and host-based intrusion detection software. Consequently, it can send out and receive malicious network packets without being identified thus becoming a serious threat in high profile attacks like corporate espionage or cyber terrorist attacks.

The rootkit/network backdoor described at (41) paper can remain invisible due to the fact that it resides, below both the NDIS (Network Driver Interface Specification) and TDI (Transport Driver Interface) Operating System interfaces, at the physical hardware layer of the network card. It is worth mentioning that both components (NDIS & TDI) are the most common target for authors focused on malware and security software since they are the deepest layers in the Operating System for sending and receiving network packets. The authors at (41) propose a rootkit backdoor that is developed and installed as a Windows kernel driver for the Intel's 8255x Ethernet Controller.

The Intel 8255x chipset consists of 2 main components the Command Unit (CU) and the Receive Unit (RU). In order to achieve data exfiltration, transmission of data out of the compromised host, a data packet with specific data structure should be constructed and send to the CU. Finally, after sending to the CU a start command the packet will be transmitted. We have to point out that the aforementioned chipset backdoor bypasses the network stack and builds the entire malicious packet to be sent, therefore making it invisible to security software and the Operating System. However, on a normal operation the packet is built in the Windows network stack and then send to NIC (Network Interface Card) driver.

For data infiltration to be achieved with the chipset backdoor, all incoming packets on the networks are handled by the RU. So, when a packet arrives it raises an interrupt to inform the CPU. On a normal operation the interrupt will be handled by the windows NDIS and the NIC driver. However, in our case in order to avoid detection the driver can redirect the interruption to a different address, where it can analyze the packet received.

Overall, according to paper (41) when a malware is deep in the OS and close to the physical hardware it becomes stealthier and more difficult to detect for various reasons:

- Security vendors find it difficult to operate at this level,
- There is no network protocol stack support from the OS at this level,
- A generic, robust product cannot be produced due to the hardware specific nature of the code

The defenses proposed by the authors also at (41) are:

- Software side defense by detecting outbound traffic and inbound traffic.
- Hardware side defense by inspecting and blocking outgoing traffic with a hardware firewall.
- Network-based intrusion detection systems (NIDS), to detect the backdoor, or any other rootkit secret traffic.

DMA Attack

Modern operating systems implement the concept of virtual memory by having each process to run in a separate address space. So, applications are prevented from accessing any memory locations, memory isolation, not explicitly authorized by the virtual memory controller, called memory management unit (MMU). MMU translates virtual memory to physical memory addresses. However, devices connected to bus do not implement the concept of virtual memory and use instead shared address space and access the physical memory by using Direct Access Memory (DMA). DMA enables I/O controllers to transfer data directly to or from the main memory bypassing the main CPU and OS. So, this could become a great threat since attackers could take advantage and tamper critical areas of memory such as the OS kernel. DMA attacks are considered a powerful class of attacks for adversaries that have physical access to a system or compromised firmware locally or remotely on peripheral hardware such as network cards.

Over the last decade, chipset and hardware vendors have introduced new technologies aimed at reinforcing kernel protection against DMA attacks, increasing security and enabling isolation at the device address space. They have introduced Input/Output Memory Management Unit (I/O MMU) technologies which behaves like a firewall for I/O controllers and filters out unauthorized I/O controllers' accesses to the main memory. However, even I/O MMU contains vulnerabilities which enable malicious code to have access to protected resources. Vulnerabilities found in the Intel VT-d, which is Intel's implementation of an I/O MMU, are thoroughly analyzed and described by authors at (42).

Furthermore, additional research (43) shows that many devices with built-in hardware protections continue to be vulnerable. Researchers tested HP and Dell laptops and discovered two different DMA vulnerabilities. More specifically they tested:

1. *Dell XPS 13 7390 2-in-1,* which was susceptible to pre-boot DMA attacks by performing DMA code injection directly over Thunderbolt during the boot process. This is called closed-chassis DMA attack due to fact that lets the attacker connect to the exposed port of the device without otherwise having to modify the device.
2. *HP ProBook 640 G4,* which even though it was protected with HP Sure Start Gen4 and VT-d, was still susceptible to an open-chassis pre-boot DMA attack. The researchers at (43) open the case of the device and replaced the wireless card in the system with a Xilinx SP605 FPGA development platform. Then they were able to successfully attack the system with a DMA attack and gain control over the device.

In general, the aforementioned attacks provide real-world examples of the weaknesses and threats at the hardware and firmware level and how quickly these can subvert hardware protections, defenses at the operating system and even software layers.

Software Attacks Enabled by Malicious Hardware

Earlier attempts were made to leak data by altering the design of an IC with hard-coded Hardware Trojans which are in fact hard to detect but the malicious circuit is useful only for the purpose for which is made. So, authors at (44) present two different hardware designs to support powerful, general purpose attacks and implements them in a real system. The attacks presented in the paper that are supported by the malicious hardware designs include:

- a privileged escalation mechanism,
- a service that steals passwords from users on the system, and
- a login backdoor that gives an attacker full and high-level access to the machine

The paper addresses the designing and implementation of Illinois Malicious Processors (IMPs) which is a malicious CPU consisted of a Leon3 processor (open-source SPARC design) with two added mechanisms:

- memory access
- invisible malicious code execution called shadow mode (Fig. 28)

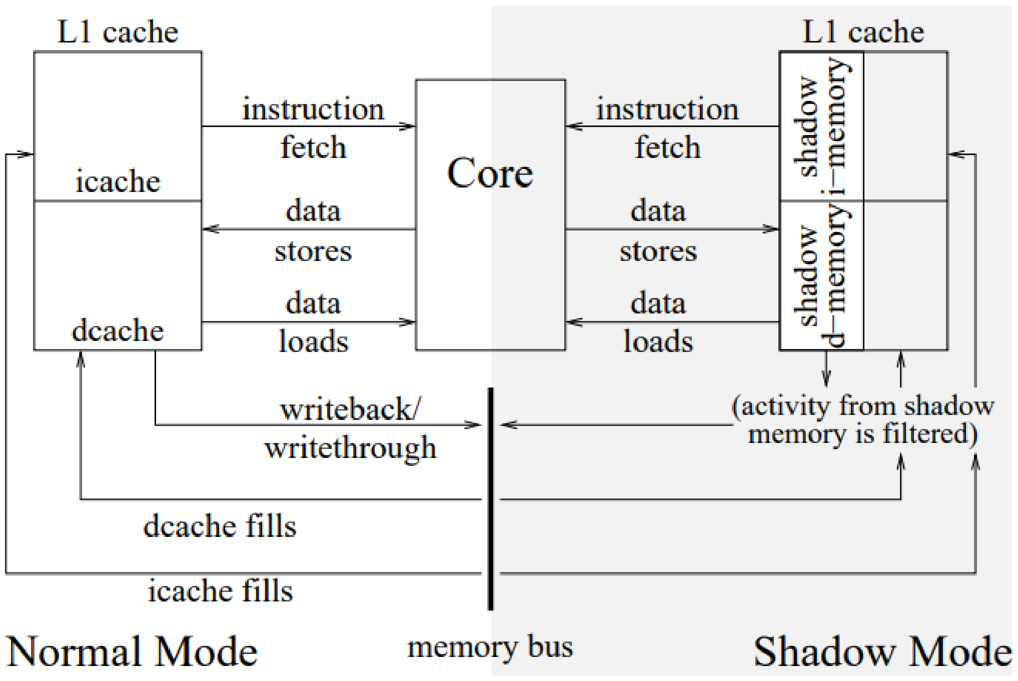


Figure 28: Shadow Mode inside the Processor

A specific sequence of bytes on the data bus triggers the memory access mechanism to get activated. Once its activated, the memory access mechanism disables the MMU privilege levels for memory accesses, therefore granting unprivileged software access to all memory, including privileged memory regions like the operating system kernel. The memory access mechanism is used to implement the privilege escalation program attack that bypasses the usual security checks.

Regarding the shadow mode mechanism authors at (44) made it similar to Intel System Management Mode (SMM) due to the fact that shadow mode instructions have full processor privilege and are not visible to software. Shadow mode get activated by a predetermined bootstrap trigger, which is a set of conditions to tell the IMP to load some code (firmware) from nearby data and execute it in shadow mode. The shadow mode mechanism is used to implement two attacks: a login backdoor and a password sniffer service.

Malicious USB Device

Nowadays USB (Universal Serial Bus) devices are widespread, easy to use and were created to improve the connection of pug-and-play devices to PCs. The USB Flash Drive is one of the most used USB Devices in the world and it consists of a flash memory with an integrated USB interface. As a result of its widespread use, it became a huge attack vector for malicious code. One category of attack exploits USB devices and does not rely on malicious software stored inside a USB Flash Drive. This attack category is called HID spoofing.

USB human interface device (USB HID) is a USB specification for computer peripherals. It describes a device class (a type of computer hardware) for human interface devices such as keyboards, mice, game controllers and alphanumeric display devices. It is a system that requires two-way communication between the system and the user. (45) So, HID spoofing keys use specialized hardware to trick a computer to expect a USB HID key, for a example a keyboard. The malicious USB disguised as fake keyboard, as soon as the device is plugged into the computer, injects keystrokes. These keystrokes are a set of commands that gives a hacker remote access to the victims' computer thus compromising it. Papers and articles describe the process of reprogramming the USB interface to act as a malicious USB HID Keyboard which can amongst other malicious actions run and execute a batch script or spawn a background

TCP reverse shell that will connect back to a server chosen by the attacker. The first demo of a malicious HID USB key was done by Adrian Crenshaw at Defcon 18 in 2010. Adrian Crenshaw used Teensy 2.0 a complete USB-based microcontroller development system, programmable over Mini USB in C or Arduino dev package with USB HID support. (46)

Some real-world USB based attacks are the Stuxnet attack discovered back in 2010 and more recently in 2017, an attack on a Nuclear Power plant by a user who wanted to watch the movie La La Land. Overall, these attacks are not happening at mass scale but mostly are used by APT (Advanced Persistent Threat) groups and nation state level attackers.

Chapter 5: Conclusion

Malicious hardware is an emerging threat as the presence of designated hardware in different applications including household, financial and military systems continue to rise. Due to the high complexity of the hardware development life cycle and decentralized production chains, hardware becomes increasingly untrusted. In this thesis, we presented a comprehensive analysis of malicious hardware. After giving a short introduction to the problem, we presented two approaches for malicious hardware taxonomy. We classified malicious hardware according to their components and behavioral characteristics and also categorized them depending on the hardware vulnerability exploited. Furthermore, we described shortly the lifecycle phases in which malicious logic can be inserted and based on that referred to the various attacker categories. We presented methodologies for the detection of hardware Trojans, such as formal verification, logic testing and side-channel analysis. We also introduced mechanisms to locate efficiently hardware Trojans within a system and presented obfuscation and invasion as reliable ways of preventing hardware from being infected by hardware Trojans. Finally, we analyzed briefly malicious hardware use cases featuring specific attack vectors.

Chapter 6: References

1. **M. Hicks, M. Finnicum, S. T. King, M. M. K. Martin, J. M. Smith.** "Overcoming an Untrusted Computing Base: Detecting and Removing Malicious Hardware Automatically". *IEEE Symposium on Security and Privacy.* [Online] 2010. https://ieeexplore.ieee.org/abstract/document/5504712?casa_token=HxGrA_xYOWoAAAAA:1SFZPZbVcmlg-vgOHV3UQCXy83o130s_UkM0XUXNesBbQo8VqGbPUGfFcbhQepJ_5zDgKrZiy-Y.

2. **contributors, Wikipedia.** 2009–2011 Toyota vehicle recalls. *Wikipedia, The Free Encyclopedia.* [Online] https://en.wikipedia.org/wiki/2009%E2%80%932011_Toyota_vehicle_recalls#Recall_timeline.

3. —. Stuxnet. *Wikipedia, The Free Encyclopedia.* [Online] https://en.wikipedia.org/wiki/Stuxnet.

4. **C. Sturton, M. Hicks, D. Wagner and S. T. King.** "Defeating UCI: Building Stealthy and Malicious Hardware.". *IEEE.* [Online] 2011. https://ieeexplore.ieee.org/document/5958022.

5. **Force, Defense Science Board Task.** "High Performancee Microchip Supply". *Department of Defense.* [Online] 2005. https://dsb.cto.mil/reports/2000s/ADA435563.pdf.

6. **R. S. Chakraborty, I. Saha, A. Palchaudhuri and G. K. Naik.** "Hardware Trojan Insertion by Direct Modification of FPGA Configuration Bitstream". *EEE Design & Test.* [Online] 2013. https://ieeexplore.ieee.org/document/6461919.

7. **Wikipedia contributors.** "Vulnerability (computing)". *Wikipedia, The Free Encyclopedia.* [Online] 2020. https://en.wikipedia.org/wiki/Vulnerability_(computing).

8. **P. Prinetto, G. Roascio.** "Hardware Security, Vulnerabilities, and Attacks: A Comprehensive Taxonomy". *ceur-ws.* [Online] 2020. http://ceur-ws.org/Vol-2597/paper-16.pdf.

9. **M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, M. Hamburg.** "Meltdown: Reading

kernel memory from user space.". *USENIX.* [Online] 2018. https://www.usenix.org/conference/usenixsecurity18/presentation/lipp.

10. **P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, Y. Yarom.** "Spectre attacks: Exploiting speculative execution.". *IEEE.* [Online] 2019. https://ieeexplore.ieee.org/abstract/document/8835233?casa_token=M3vqYEixWlIAAAAA:s3foP7lkwTQvO4BGtAqvbpfaW8QCQCf8Kzxtzxl3rFVIQUWO9z2UTcG_epX_iy-T7Ssh-aWAXy0.

11. **Y. Yarom, K. Falkner.** "Flush+ reload: a high resolution, low noise, l3 cache side-channel attack". *USENIX.* [Online] 2014. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom.

12. **M. Schwarz, S. Weiser, D. Gruss, C. Maurice, S. Mangard.** "Malware guard extension: Using sgx to conceal cache attacks.". *Springer.* [Online] 2017. https://link.springer.com/article/10.1186/s42400-019-0042-y.

13. **S. Sethumadhavan, A. Waksman.** "Silencing Hardware Backdoors". *IEEE Symposium on Security and Privacy.* [Online] 2011. https://ieeexplore.ieee.org/document/5958021.

14. **S. Sethumadhavan, A. Waksman, M. Suozzo, Y. Huang, J. Eum.** “Trustworthy hardware from untrusted components”. *ACM Digital Librady.* [Online] 2015. https://dl.acm.org/doi/10.1145/2699412.

15. **K. M. Goertzel, B. A. Hamilton.** "Integrated Circuits Security Threats and Hardware Assurance Countermeasures". *researchgate.* [Online] 2013. https://www.researchgate.net/publication/290762635_Integrated_circuit_security_threats_and_hardware_assurance_countermeasures.

16. **Wikipiedia contributors.** "Integrated Circuits". *Wikipedia, The Free Encyclopedia.* [Online] 2020. https://en.wikipedia.org/wiki/Integrated_circuit.

17. **Rajat Subhra Chakraborty, Seetharam Narasimhan and Swarup Bhunia.** "Hardware Trojan: Threats and Emerging Solutions". *IEEE International High Level Design Validation and Test Workshop.* [Online] 2009. https://ieeexplore.ieee.org/document/5340158.

18. **K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia and M. Tehranipoor.** "Hardware Trojans: Lessons Learned after One Decade of Research". *researchgate.* [Online] 2016.

https://www.researchgate.net/publication/303711304_Hardware_Trojans_Lessons_Learned_after_One_Decade_of_Research.

19. **M. Banga, M. S. Hsiao.** "A region based approach for the identification of hardware Trojans". *IEEE.* [Online] 2008. https://ieeexplore.ieee.org/document/4559047?denied=.

20. **X. Wang, M. Tehranipoor and J. Plusquellic.** “Detecting malicious inclusions in secure hardware: Challenges and solutions”. *IEEE.* [Online] 2008. https://ieeexplore.ieee.org/abstract/document/4559039.

21. **M. Abramovic, and P. Bradley.** "Integrated Circuit Security - NewThreats and Solutions". *ACM Ditigal Library.* [Online] 2009. https://dl.acm.org/doi/10.1145/1558607.1558671.

22. **L.W. Kim, J.D. Villasenor and C.K. Koc.** "A Trojan-resistant System-on-chip Bus Architecture". *IEEE.* [Online] 2009. https://ieeexplore.ieee.org/document/5379966.

23. **G. Bloom, B. Narahari and R. Simha.** "OS Support for Detecting Trojan Circuit Attacks". *IEEE.* [Online] 2009. https://ieeexplore.ieee.org/document/5224959.

24. **Bloom, G.** "Providing Secure Execution Environments with a Last Line of Defense against Trojan Circuit Attacks". *Science Direct.* [Online] 2009. https://www.sciencedirect.com/science/article/pii/S0167404809000145.

25. **S. Wei, S. Meguerdichian and M. Potkonjak.** "Gate-level characterization: Foundations and hardware security applications,". *IEEE.* [Online] 2010. https://ieeexplore.ieee.org/document/5522644.

26. **M. Banga, and M. Hsiao.** "A Novel Sustained Vector Technique for the Detection of Hardware Trojans.". *IEEE.* [Online] 2009. https://ieeexplore.ieee.org/document/4749695.

27. **M. Chandrasekar, M. Banga, L. Fang, and M. Hsiao.** "Guided test generation for isolation and detection of embedded trojans in ics.". *ACM Digital Library.* [Online] 2008. https://dl.acm.org/doi/abs/10.1145/1366110.1366196.

28. **R. M. Rad, X. Wang, M. Tehranipoor and J. Plusquellic.** "Power supply signal calibration techniques for improving detection resolution to hardware Trojans.". *IEEE.* [Online] 2008. https://ieeexplore.ieee.org/document/4681643.

29. **Tehranipoor, X. Zhang and M.** "RON: An on-chip ring oscillator network for hardware Trojan detection.". *IEEE.* [Online] 2011. https://ieeexplore.ieee.org/abstract/document/5763260.

30. **Mirhoseini, F. Koushanfar and A.** "A Unified Framework for Multimodal Submodular Integrated Circuits Trojan Detection.". *IEEE.* [Online] 2011. https://ieeexplore.ieee.org/document/5657256.

31. **Bhunia, R. S. Chakraborty and S.** "Hardware protection and authentication through netlist level obfuscation.". *IEEE.* [Online] 2008. https://ieeexplore.ieee.org/document/4681649.

32. **R. S. Chakraborty, S. Paul and S. Bhunia.** "On-demand transparency for improving hardware Trojan detectability.". *IEEE.* [Online] 2008. https://ieeexplore.ieee.org/document/4559048.

33. **Chakraborty, S. Bhunia and R. S.** "Security against hardware Trojan through a novel application of design obfuscation.". *IEEE.* [Online] 2009. https://ieeexplore.ieee.org/document/5361306.

34. **M. S. Hsiao, and M. Banga.** "ODETTE: A non-scan design-for-test methodology for Trojan detection in ICs.". *IEEE.* [Online] 2011. https://ieeexplore.ieee.org/document/5954989.

35. **D. McIntyre, F. Wolff, C. Papachristou, S. Bhunia and D. Weyer.** "Dynamic evaluation of hardware trust.". *IEEE.* [Online] 2009. https://ieeexplore.ieee.org/document/5224990.

36. **D. McIntyre, F. Wolff, C. Papachristou and S. Bhunia.** "Trustworthy computing in a multi-core system using distributed scheduling." . *IEEE.* [Online] 2010. https://ieeexplore.ieee.org/document/5560200.

37. **Huffmire, T.** "Extended abstract: Trustworthy system security through 3-D integrated hardware.". *IEEE.* [Online] 2008. https://ieeexplore.ieee.org/document/4559061.

38. **Villasenor, L. Kim and J. D.** "A System-On-Chip Bus Architecture for Thwarting Integrated Circuit Trojan Horses." . *IEEE.* [Online] 2011. https://ieeexplore.ieee.org/document/5556060.

39. **Sethumadhavan, A. Waksman and S.** "Tamper Evident Microprocessors.". *IEEE.* [Online] 2010. https://ieeexplore.ieee.org/document/5504715.

40. **A. Das, G. Memik, J. Zambreno and A. Choudhary.** "Detecting/preventing information leakage on the memory bus due to malicious hardware.". *IEEE.* [Online] 2010. https://ieeexplore.ieee.org/document/5456930.

41. **Sparks, S.** "A chipset level network backdoor.". *ACM Digital Library.* [Online] 2009. https://dl.acm.org/doi/10.1145/1533057.1533076.

42. **Sang, F.** "Exploiting an I/OMMU vulnerabiltiy.". *IEEE.* [Online] 2010. https://ieeexplore.ieee.org/abstract/document/5665798.

43. **researchers, Eclypsium.** "Direct Memory Access Attacks.". *Eclypsium.* [Online] 2020. https://eclypsium.com/2020/01/30/direct-memory-access-attacks/.

44. **King, S.** "Designing and Implementing Malicous Hardware.". *ACM Digital Library.* [Online] 2008. https://dl.acm.org/doi/10.5555/1387709.1387714.

45. **contributors, Wikipedia.** "USB Human Interface Device class.". *Wikipedia, The Free Encyclopedia.* [Online] https://en.wikipedia.org/wiki/USB_human_interface_device_class.

46. **Crenshaw, A.** "Prgrammable HDI USB Keyboard/Mouse Dongle for Pen-Testing.". *DEFCON.* [Online] 2018. https://defcon.org/images/defcon-18/dc-18-presentations/Crenshaw/DEFCON-18-Crenshaw-PHID-USB-Device.pdf.

47. **J., Tehranipoor M. and Lee.** "Protecting ips against scan-based side-channel attacks" In Introduction to Hardware Security and Trust. *Tehranipoor.* [Online] 2012. https://tehranipoor.ece.ufl.edu/intro-to-hardware-security-and-trust/.