A Comprehensive Survey on Hardware-Software co-Protection against Invasive, Non-Invasive and Interactive Security Threats

Md Habibur Rahman, Graduate Student Member, IEEE
Email: rahman.mdhabibur@ufl.edu
University of Florida
Gainesville, FL, United States

***Abstract*—In the face of escalating security threats in modern computing systems, there is an urgent need for comprehensive defense mechanisms that can effectively mitigate invasive, non-invasive and interactive security vulnerabilities in hardware and software domains. Individually, hardware and software weaknesses and probable remedies have been practiced but protecting a combined system has not yet been discussed in detail. This survey paper provides a comprehensive overview of the emerging field of Hardware-Software co-Protection against Invasive and Non-Invasive Security Threats. We systematically review state-of-the-art research and developments in hardware and software security techniques, focusing on their integration to create synergistic defense mechanisms. The survey covers a wide range of security threats, including physical attacks, side-channel attacks, and malware exploits, and explores the diverse strategies employed to counter them. Our survey meticulously examines the landscape of security vulnerabilities, encompassing both physical and software-based attack vectors, and explores the intricate interplay between hardware and software defenses in mitigating these threats.Furthermore, we discuss the challenges and opportunities associated with Hardware-Software co-Protection and identify future research directions to advance the field. Through this survey, we aim to provide researchers, practitioners, and policymakers with valuable insights into the latest advancements and best practices for defending against complex security threats in modern computing environments.**

***Index Terms*—Hardware-Software co-Protection, Non-Invasive Attacks, Invasive Attacks, Hardware-Software Interaction, IC Security, Hardware Security**

I. Introduction

In an era characterized by ubiquitous computing and interconnected systems, ensuring the security and integrity of computing environments [1], [2], [3] has emerged as a paramount concern. The pervasive nature of modern computing infrastructures exposes them to an increasingly diverse array of security threats [4], [5], ranging from physical attacks [6], [7] on hardware components (e.g. Invasive and non-Invasive attacks [8]) to sophisticated software exploits [9], [10] targeting system vulnerabilities. In response to these multifaceted threats, researchers and practitioners have turned to a synergistic approach known as Hardware-Software co-Protection [11], [12] to fortify computing systems against both invasive and non-invasive security vulnerabilities. The abstraction layers of hardware and software are displayed in Figure 1

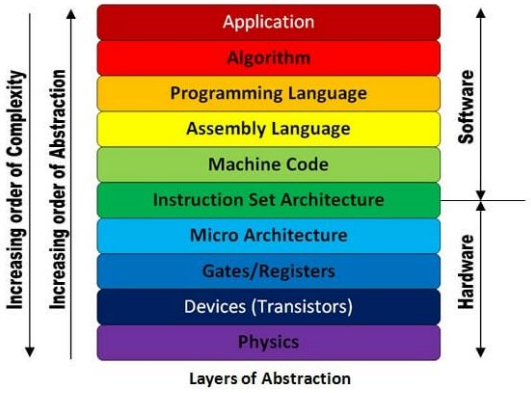


Fig. 1. HW-SW Layers of a computing system. [13]

The concept of Hardware-Software co-Protection embodies the integration of hardware and software-based security mechanisms, leveraging the strengths of each domain to create robust defenses against a broad spectrum of threats. Hardware-based defenses provide a solid foundation by implementing secure hardware architectures, encryption primitives [14], [15], and tamper-resistant technologies to safeguard critical system components against physical attacks. This security primitives in the hardware level can be incorporated in the software domain to facilitate risk-free execution of software programs [16], [17] . Meanwhile, software-based defenses employ techniques such as secure boot mechanisms, memory protection, and runtime monitoring to mitigate software vulnerabilities and thwart malware exploits. Software level techniques for security measurement [18] can be taken to construction of hardware to make the hardware secure.

This survey paper presents a detailed exploration of the Hardware-Software co-Protection paradigm against Invasive and Non-Invasive Security Threats. By systematically examining the landscape of security vulnerabilities and the interplay between hardware and software defenses, this survey aims to provide a thorough understanding of the latest advancements and best practices in this topic. Through an in-depth analysis

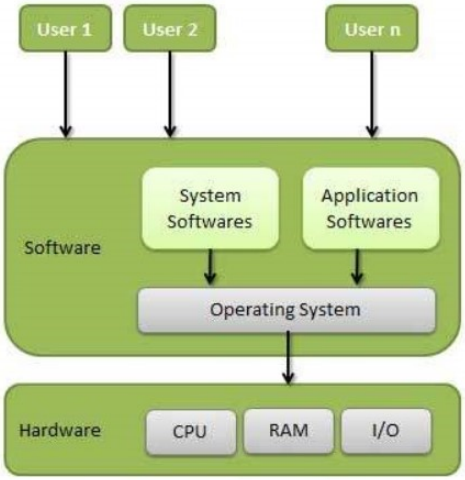


Fig. 2. How Softwares and Hardwares are intertwined in a Real Computing System.

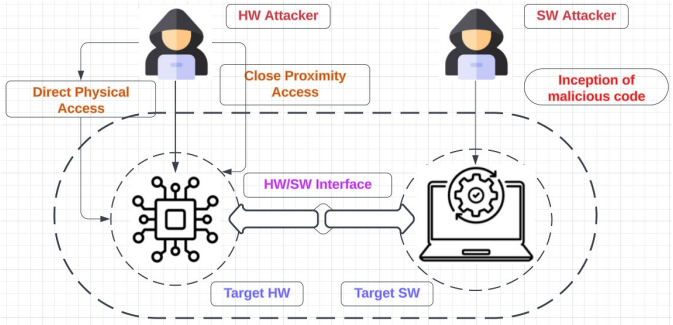


Fig. 3. HW-SW Attack Scenario and Possible co-Protection against them

of invasive threats targeting hardware components, including fault injection attacks [19], [20], [21] and reverse engineering techniques, coupled with an examination of software-based threats such as malware infections and side-channel attacks, this survey elucidates the diverse challenges facing modern computing systems. A framework for HW/SW co-ordination and probable holistic protection against attacks from either domain are shown in Figure 4.

The subsequent sections of this paper will discuss the following topics.

- *Overview of Security Threats in HW and SW Domain:* We will provide an in-depth examination of the current landscape of hardware security threats facing computing systems. This will include an analysis of both invasive threats, such as physical tampering and hardware Trojans, as well as non-invasive threats like software-based attacks including malware, side-channel attacks [22], and cryptographic vulnerabilities . By understanding the diverse range of threats, the paper sets the stage for the necessity of comprehensive protection mechanisms.
- *Threat model in SW and HW Domains Combined:* When a software runs into a hardware(e.g. RTL or gate level design), each of the domains are not aware of others' respective design properties or other quantities. We will explore which vulnerabilities or threats may arise because of lack of awareness of SW execution from hardware or hardware execution from software.
  For instance, it may happen that running a complex design in software domain is causing excessive power dissipation in a a particular part of the design. But the hardware is unaware and unprotected against this. Similarly, software is running without any awareness from the hardware execution. These kind of security vulnerabilities will be explored.
- *Insights from real-World instances and existing methodologies:* It will draw insights from real-world research papers to illustrate the effectiveness of hardware-software co-protection approaches. By referencing findings and case studies from reputable research publications, the paper will provide concrete examples of how these approaches have been implemented and their impact on enhancing security posture. This section will include summaries of key findings, experimental results, and comparative analyses from selected research papers.
- *Analysis of Co-Protection Methodologies:* This paper will delve into various methodologies and techniques that integrate both hardware and software defenses to mitigate security threats effectively. This analysis will cover a wide range of approaches including hardware sensor response based compiler execution, hardware root of trust, protection of software programs using hardware power traces and hardware model of invasive and non-invasive attacks, cryptographic accelerators, and software-based intrusion detection systems. The paper will discuss the strengths and limitations of each approach, highlighting the need for comprehensive solutions.
- *Categorization of Strategies:* We will categorize hardware-software co-protection strategies based on their underlying principles and deployment scenarios. This categorization may include classifications such as prevention techniques, detection mechanisms, and mitigation strategies. Additionally, the paper will categorize solutions based on the targeted threat models, such as protecting against physical attacks, network-based attacks, or insider threats.
- *Running and Future Working Directions and Challenges:* Lastly, the paper will discuss future directions and challenges in the development and implementation of hardware-software co-protection mechanisms. This may include emerging trends such as the integration of machine learning and AI-based approaches, challenges in securing emerging technologies like IoT and cloud computing, and the need for standardization and interoperability of security solutions.

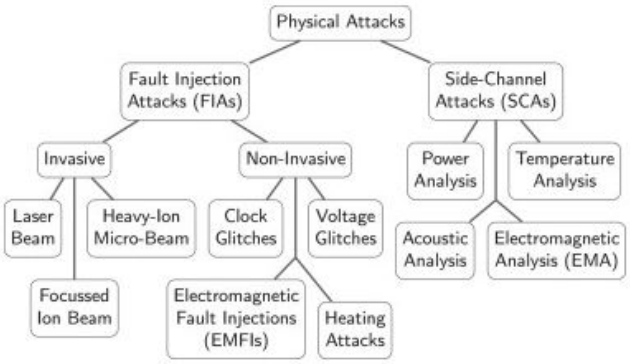


Fig. 4. HW Physical Attack Taxonomy [23]

II. Hardware non-Invasive and Invasive Security Vulnerabilities:

A. Hardware Non-Invasive Security Threats

Non-invasive hardware security vulnerabilities refer to weaknesses in electronic systems that can be exploited without physically altering the hardware [24]. These vulnerabilities often exploit unintended behaviors or characteristics of the hardware components themselves, such as electromagnetic emissions, power consumption, or timing variations. Principal types of non-Invasive attacks consist of fault injection attacks and side channel attacks.

1) Fault Injection Attacks:: These attacks involve inducing faults or errors in a device's operation to compromise its security. Through the exploitation of weaknesses within hardware design or the manufacturing process, attackers can inject faults such as voltage spikes, clock glitches, or electromagnetic interference. These flaws may result in unforeseen system behaviors, crashes, or breaches in security, posing a potential threat to the confidentiality, integrity, and availability of the system.

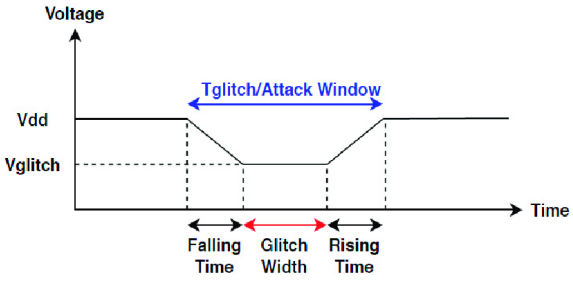


Fig. 5. Representation of negative voltage glitch [25]

- *Voltage glitch attack:* One major fault injection attack is Voltage glitching meaning manipulating the device's power supply to induce faults and trigger unexpected behaviors [26], [27]. By manipulating the voltage levels supplied to the device during its operation(please refer to Figure 5), attackers can induce glitches or faults in the system, causing it to malfunction or execute unintended commands. These attacks are often meticulously timed to occur during critical moments, such as cryptographic operations or authentication processes, enabling attackers to bypass security measures or extract sensitive information.

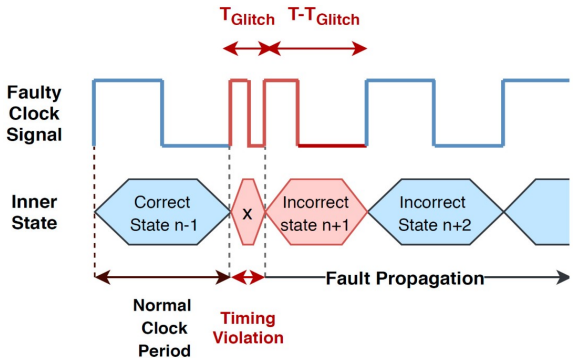


Fig. 6. Representation of clock glitch and fault-injection [28]

- *Clock glitch attack:* Clock glitching means introducing glitches into the device's clock signal (please refer to Figure 6) to disrupt its normal operation [29], [30] and potentially exploit vulnerabilities. By manipulating the clock signals that govern the device's operations, attackers can introduce glitches or disturbances in the timing sequence, causing the device to behave unpredictably or execute unintended instructions(please refer to Figure 6).
- *Optical fault Injection:* Using laser or light pulses to induce faults in the device's components [31] leading to security compromises. In this type of attack, a focused laser beam is precisely targeted at specific components within the device, such as integrated circuits or memory cells. By introducing localized heat or inducing electromagnetic interference, the laser can disrupt the normal operation of the targeted components [32], causing faults or errors in their behavior. These faults can be exploited by attackers to manipulate the device's operation, extract confidential data, or bypass security measures.

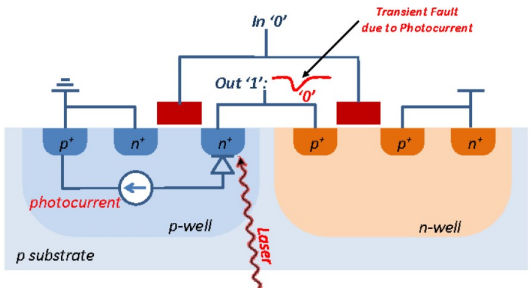


Fig. 7. Generation of photocurrent through the back-side due to LASER attack. [33]

2) Side Channel Attacks:: These attacks exploit information leaked by a device during its normal operation [34], such as power consumption, electromagnetic radiation, or timing variations. Common types include:

- *Power Analysis Attacks:* A Power Analysis Side-Channel Attack involves exploiting fluctuations in the power usage

of electronic devices during their functioning to deduce confidential information, like cryptographic keys or processed data [35], [36], [37]. By monitoring and analyzing these power consumption patterns, attackers can infer details about the device's internal operations, such as executed instructions or variable values. This type of attack is especially concerning for embedded systems and cryptographic devices due to its potential for remote execution without physical access.

- *Electromagnetic (EM) Side-Channel Attacks:* An Electromagnetic (EM) side-channel attack is a sophisticated method used to exploit unintentional emissions of electromagnetic radiation or electrical signals from electronic devices during their operation. By analyzing these emissions, attackers can glean sensitive information [38], [39], [40] such as cryptographic keys or data being processed by the device, without directly accessing the device itself. EM side-channel attacks pose a significant threat to the security of embedded systems, particularly those handling confidential or sensitive information, and require specialized equipment and expertise to execute.
- *Timing Attacks:* Timing variations in a device's operations can be exploited to deduce information about cryptographic computations or sensitive processes [41], [42].

B. Hardware Invasive Security Threats

Invasive hardware attacks involve physically tampering with electronic devices to exploit vulnerabilities or compromise their security [43]. Unlike non-invasive attacks that rely on analyzing the device's behavior or emissions without altering its physical structure, invasive attacks directly manipulate the hardware components. These attacks typically require physical access to the device, allowing attackers to directly interact with its internal circuitry.

Hardware Trojan attacks [44], [45], [46] involve the clandestine insertion of malicious circuitry, known as Trojans, into integrated circuits (ICs) during the design or manufacturing stages. These Trojans are designed to remain dormant under normal operating conditions but can be triggered remotely or under specific circumstances to perform malicious activities, such as leaking sensitive information [47], causing system malfunctions, or providing unauthorized access. Hardware Trojans pose significant challenges to the integrity and security of electronic systems [48], as they can evade traditional software-based security measures and remain undetected during functional testing. Detecting and mitigating hardware Trojans [49] require specialized techniques, including physical inspection, side-channel analysis, and formal verification, to ensure the trustworthiness of hardware components and prevent potential exploitation.

Attackers physically modify the device's circuitry at the chip level to introduce backdoors, modify functionality, or bypass security mechanisms. This can involve techniques such as laser cutting, wire bonding, or focused ion beam (FIB) milling.

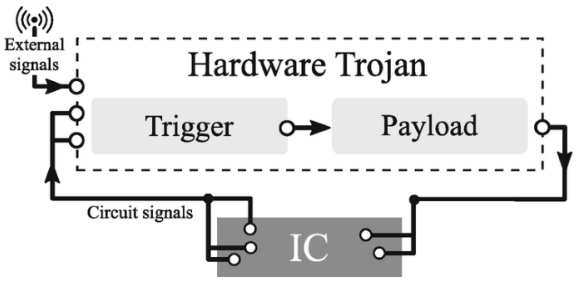


Fig. 8. Hardware Trojan [50]

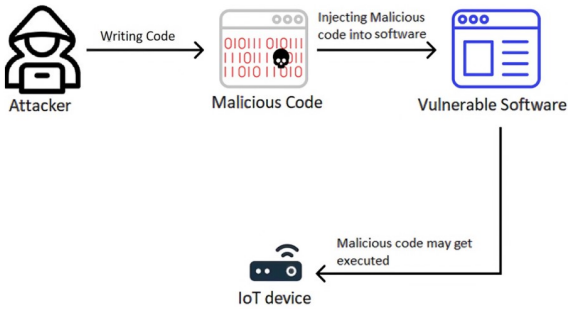


Fig. 9. Bad code injection process [51]

III. Software Execution Vulnerabilities

Software program security threats encompass a broad spectrum of risks and vulnerabilities that can compromise the confidentiality, integrity, and availability of software systems [52]. These threats can originate from various sources, including malicious actors, software bugs, design flaws, and insecure coding practices. Understanding and mitigating these threats are essential for ensuring the trustworthiness and reliability of software applications. Below are some common types of software program security threats:

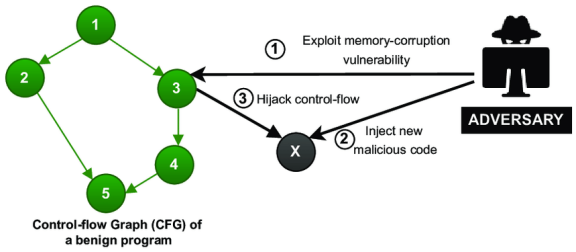


Fig. 10. Example of code-injection attack. The CFG represents the accurate execution flows of a harmless software, where the graph nodes(1-5) indicate a software instruction. A code-injection attacker performs the actions 1-3. [53]

- *Malware:* Malicious software, such as viruses, worms, Trojans, and ransomware, pose significant threats to software security [54]. Malware can infiltrate systems through various vectors, including email attachments, malicious websites, and infected software downloads. Once installed, malware can steal sensitive information, disrupt system operations, and provide unauthorized access to

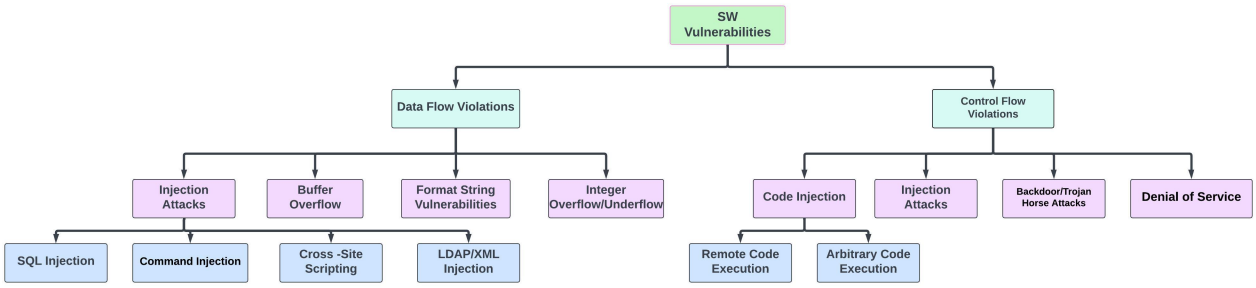


Fig. 11. SW Vulnerabilities and Attacks for data and control flow violations.

attackers [55]. A malicious code injection process can be seen in Figure 9.

- *Code Injection Attacks:* Injection attacks like SQL injection and cross-site scripting (XSS), can probe for vulnerabilities in software input validation mechanisms. Attackers inject malicious code or commands into input fields [56], allowing them to manipulate the behavior of the software and access sensitive data stored in databases or execute unauthorized actions on behalf of legitimate users. By inserting bad code, adversaries can also attempt to steal data and/or control flow of the software design [53], which can be seen in Figure 10.
- *Authentication and Authorization Flaws:* Weak or inadequate authentication and authorization mechanisms can lead to unauthorized access to sensitive data and functionalities. Common vulnerabilities include weak passwords, insufficient password policies, insecure session management, and privilege escalation exploits. Attackers exploit these flaws to bypass validation controls and acquire illegal access to critical system resources.

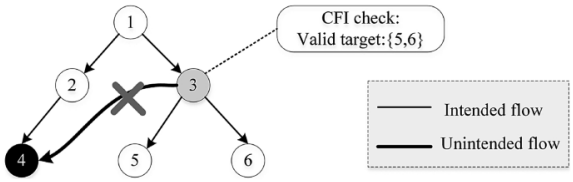


Fig. 12. Control Flow Integrity [57]

- *Information Leakage:* Information leakage vulnerabilities occur when sensitive data is inadvertently exposed to unauthorized parties [58], [59]. This can happen through various channels, including error messages, log files, configuration files, and network transmissions. Attackers can exploit information leakage vulnerabilities to gather intelligence about system configurations, user behaviors, and application logic, facilitating further attacks [60].
- *Data Exfiltration Attacks:* Data exfiltration attacks involve unauthorized extraction or leakage of sensitive data from a software program. Attackers may exploit vulnerabilities in data handling processes, such as insecure data storage, weak encryption, or inadequate access controls, to steal sensitive information. By compromising data flow integrity, attackers can extract confidential data, trade secrets, or personally identifiable information (PII) from the target system.

Fig. 13. Data Leakage Resources.

- *Data Tampering:* Data tampering attacks involve unauthorized modification or manipulation of data stored or transmitted by software systems. Attackers can tamper with data to alter its integrity, accuracy, or authenticity [61], leading to erroneous decisions, financial fraud, or privacy breaches. Common targets of data tampering attacks include databases, configuration files, digital documents, and network communications [62].

IV. Threat Model: SW-HW Affecting Each Other

When there is a specific security attack in hardware domain, it often occurs that software execution is unaware of hardware vulnerabilities and vice versa. Software and hardware being unaware of each others' execution gives rise to several vulnerabilities to each domain.

How hardware attacks affect software execution:

- *Data breaches:* Hardware vulnerabilities, such as side-channel attacks or insecure memory access, can compromise the confidentiality of sensitive data processed by software applications. Attackers may exploit weaknesses in hardware components to gain unauthorized access to data stored in memory [64] or transmitted across the sys-

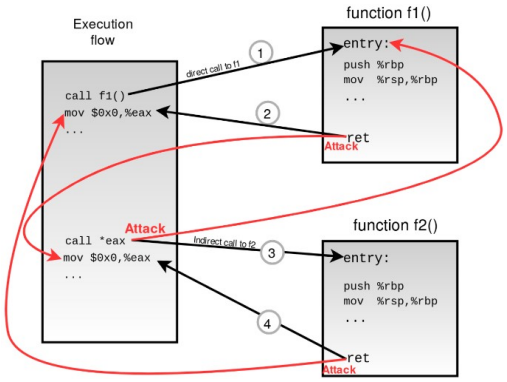


Fig. 14. Bypass example of coarse-grained control flow integrity. The attacker can divert the control flow from several calls and return points. Red arrows show the valid destinations where attackers can redirect the flow. [63]

tem, jeopardizing the privacy of users and organizations [65].

Hardware vulnerabilities can significantly contribute to the emergence of software data breaches by providing attackers with exploitable entry points into systems. These vulnerabilities often stem from weaknesses in the design, implementation, or configuration of hardware components, such as processors, memory modules, or peripheral devices. Attackers can exploit these vulnerabilities to acquire uninvited access to security-critical data or manipulate the behavior of software systems.

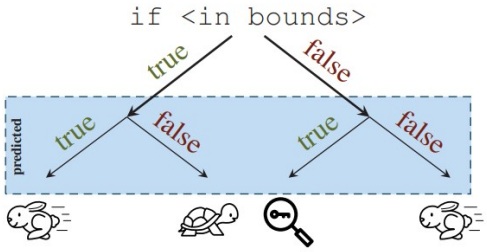


Fig. 15. Spectre attack scenario. Until the bounds check yields a definitive outcome, the branch predictor proceeds with the anticipated branch target, enhancing overall execution speed when predictions are accurate. However, in cases where the bounds check is mistakenly predicted as true, there's potential for secret information leakage under specific circumstances. [66].

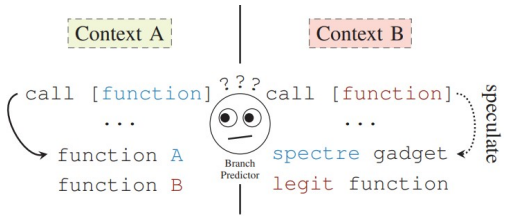


Fig. 16. Spectre attack scenario. Branch predictor makes wrong prediction according to training data from adversaries [66].

An instance of this attack is when attackers leverage hardware vulnerabilities to execute malicious code or exploit software bugs that would otherwise be inaccessible. For example, hardware vulnerabilities like speculative execution flaws (e.g., Spectre [66], Meltdown [67] and deterministic rowhammer [68]) can be exploited to bypass software-based security measures and access privileged information stored in memory. Similarly, vulnerabilities in hardware-based encryption or authentication mechanisms can undermine the security of software applications that rely on these mechanisms for data protection.

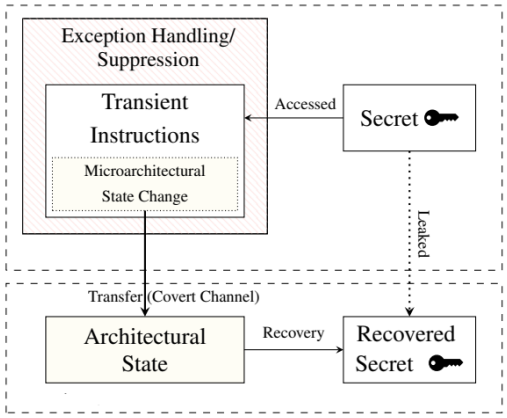


Fig. 17. The Meltdown exploit leverages exception handling or suppression to execute a sequence of temporary instructions. These temporary instructions acquire a secret value that persists and alter the microarchitectural state of the processor accordingly. This establishes one end of a microarchitectural covert channel. The recipient end reads the microarchitectural state, converting it to architectural, and retrieves the secret value. [67].

- *Integrity compromise:* Hardware-level attacks, such as firmware tampering or hardware Trojans, can undermine the integrity of software execution by injecting malicious code or altering critical system functions. This can lead to the execution of unauthorized commands, modification of software binaries, or manipulation of system behavior, posing significant risks to the reliability and trustworthiness of software applications.

  Hardware vulnerabilities can serve as the foundation for software integrity violations, undermining the trustworthiness of software systems and leading to potential security breaches. These vulnerabilities may arise from flaws in the design, implementation, or configuration of hardware components, such as processors, memory modules, or input/output devices. Attackers can exploit these vulnerabilities to manipulate the execution environment of software applications, compromise the integrity of data, or subvert critical security mechanisms.

  For instance, attackers leveraging hardware vulnerabilities to inject malicious code into software systems or modify existing code to alter program behavior. For example, vulnerabilities in hardware-based memory protection mechanisms can enable attackers to overwrite critical system data or execute arbitrary code in privileged contexts, leading to unauthorized access or control over software resources [69], [66]. Similarly, flaws in hardware-based encryption or authentication mechanisms can be exploited to bypass software-based security con-

trols and tamper with data integrity, compromising the trustworthiness of software operations [70], [71].

- *Availability Issues:* Hardware-based security threats can disrupt the availability of software services by exploiting vulnerabilities in underlying hardware infrastructure. Denial-of-Service (DoS) attacks [72] targeting hardware components, such as network interface controllers or memory modules, can degrade system performance, cause system crashes, or render software applications inaccessible, resulting in service disruptions and financial losses.

  A frequent occurrence involves attackers leveraging hardware vulnerabilities to exhaust system resources like CPU, memory, or network bandwidth, inundating the system with an abundance of traffic or nefarious requests. For instance, weaknesses in network hardware or protocols may be utilized to produce extensive volumes of network activity [73], overwhelming network connections and inducing congestion or packet loss. Likewise, deficiencies in memory management units (MMUs) [74] or memory controllers may prompt memory depletion or fragmentation, thereby precipitating system instability or failures.

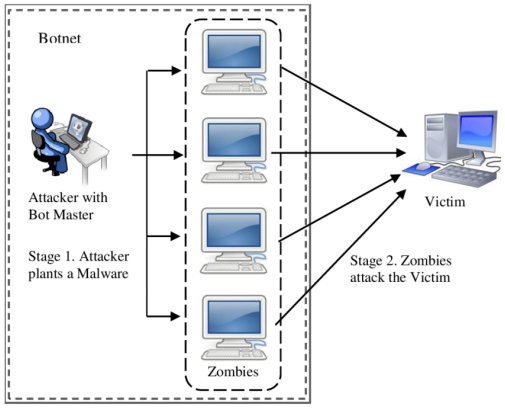


Fig. 18. A simple distributed denial of service attack scenario. [75].

Furthermore, compromised hardware components can serve as attack vectors for launching distributed denial-of-service (DDoS) attacks [72], [76], [77], where multiple compromised devices coordinate to flood target systems with malicious traffic. For instance, attackers may exploit vulnerabilities in Internet of Things (IoT) devices or embedded systems to create botnets capable of launching massive DDoS attacks against internet-facing services, disrupting their availability to legitimate users (Figure 18.

- *Software Exploitation:* Hardware vulnerabilities can serve as entry points for exploiting software vulnerabilities, enabling attackers to escalate privileges [66], [67], execute arbitrary code, or bypass software-based security mechanisms [78]. By exploiting weaknesses in hardware architectures, attackers can launch sophisticated attacks, such as buffer overflows, code injection [79], [80], or privilege escalation [66], compromising the security and stability of software systems.

  Some hardware vulnerabilities, such as buffer overflows or memory corruption flaws in processors, can directly impact software execution. Attackers can craft malicious inputs or code sequences that exploit these vulnerabilities to gain unauthorized access [81], execute arbitrary code, or manipulate system behavior.

How software attacks affect hardware execution:

Software attacks can significantly impact hardware execution by exploiting vulnerabilities in software components to manipulate or compromise the behavior of underlying hardware. These attacks can manifest in various forms, including:

- *Malware Exploitation:* Malicious software such as viruses, worms, or Trojans can extract security weaknesses in operating systems or applications to acquire unauthorized access to hardware resources. Once compromised, the malware can manipulate hardware functionality, disrupt system operations, or steal sensitive data.
- *Code Injection:* Techniques like buffer overflows or injection attacks enable attackers to inject malicious code into running processes. If successful, this injected code can execute arbitrary commands, manipulate hardware registers, or even reconfigure hardware settings, leading to system instability or unauthorized access.

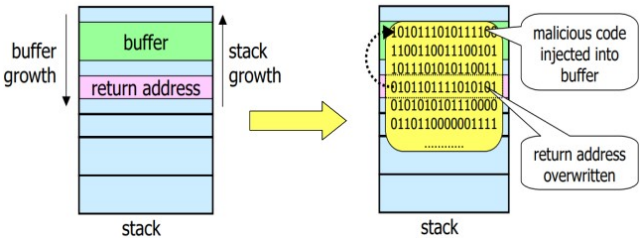


Fig. 19. Code injection attack scenario using stack. [82].

- *Privilege Escalation:* Software vulnerabilities those allow unauthorized users to escalate their privileges can enable attackers to gain elevated access to hardware resources. With escalated privileges, attackers can manipulate hardware configurations, access restricted data, or install malicious firmware, compromising the integrity of the hardware platform [83].

  Windows/Linux kernel Privilege escalation: This vulnerability allowed attackers to escalate privileges on Windows or Linux systems [84] by exploiting a flaw in the kernel. By running a specially crafted application, an attacker could execute arbitrary code with elevated privileges, potentially gaining unauthorized access to hardware resources and compromising system integrity [85].
- *Denial-of-Service (DoS) Attacks:* DoS attacks targeting software vulnerabilities can overwhelm hardware resources with excessive requests or malicious traffic. Software vulnerabilities can lead to unauthorized hardware Denial of Service (DoS) attacks through various mechanisms. Here's how:

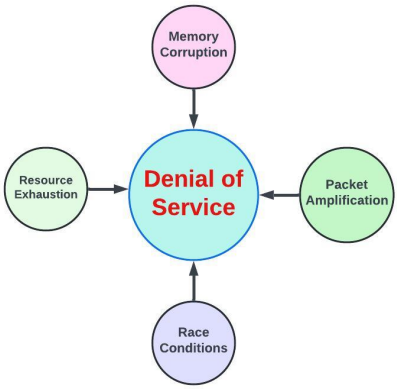


Fig. 20. Sources of denial of service attack in hardware due to software vulnerabilities.

- *Resource Exhaustion:* Vulnerabilities in software systems can be exploited to consume excessive system resources, such as CPU, memory, or network bandwidth. Attackers can leverage these vulnerabilities to launch DoS attacks by flooding the system with requests or executing resource-intensive operations [86], leading to the exhaustion of hardware resources and causing legitimate users to be denied access to the system [87]. Significance of detection [88] and prevention of resource exhaustion attack is paramount.
- *Memory Corruption:* Software vulnerabilities, such as buffer overflows [89], [90] or memory corruption flaws, can result in memory leaks or memory corruption issues. Attackers can exploit these vulnerabilities to consume system memory excessively [91], leading to memory depletion and system instability. This can cause hardware components to become unresponsive or malfunction, resulting in a Denial of Service for legitimate users.
- *Packet Amplification:* Vulnerabilities in network protocols or network-facing software can be exploited to amplify network traffic, leading to network congestion and service disruption. Attackers can manipulate network packets to increase their size or frequency, leveraging vulnerable software components to amplify the impact of their attacks on hardware resources, such as network switches, routers, or firewalls.
- *Interrupt Storms:* Software vulnerabilities in device drivers or kernel components can lead to the generation of excessive hardware interrupts or interrupts storms. Attackers can exploit these vulnerabilities to trigger a large number of hardware interrupts, overwhelming the system's interrupt handling capabilities and causing hardware devices to become unresponsive or enter into a degraded state, resulting in a Denial of Service for legitimate users.
- *Race Conditions:* Software vulnerabilities those result in race conditions or concurrency issues can be exploited to disrupt the normal operation of hardware components. Attackers can manipulate the timing or sequence of software operations to create race conditions [92], [66], leading to unpredictable behavior in hardware devices or systems. This can result in hardware resources being locked or unavailable for legitimate users, causing a Denial of Service [76].

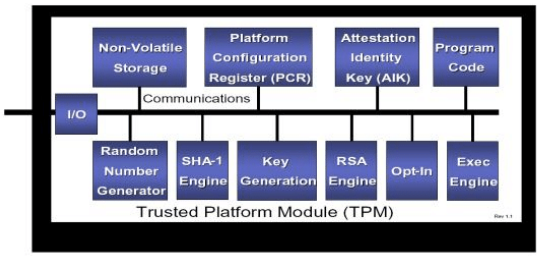
- *Firmware Exploitation:* Vulnerabilities in firmware, such as BIOS or device drivers, can be exploited to compromise hardware functionality. Attackers can modify firmware settings, implant rootkits, or disable security features, undermining the integrity and security of the hardware platform [93].
- *Side-Channel Attacks:*
Software side-channel attacks, such as timing attacks or cache-based attacks while primarily targeting vulnerabilities within software, can also precipitate significant issues in hardware components. These attacks exploit the unintended leakage of information from software execution, often exploiting the underlying architecture and implementation of hardware. For instance, speculative execution [66], a performance optimization technique in modern processors, can inadvertently expose sensitive data through timing or cache-based side channels. This exposure can lead to a myriad of problems in hardware, including compromised confidentiality, integrity, and availability.

V. Existing Techniques for Hardware Software Co-Protection:

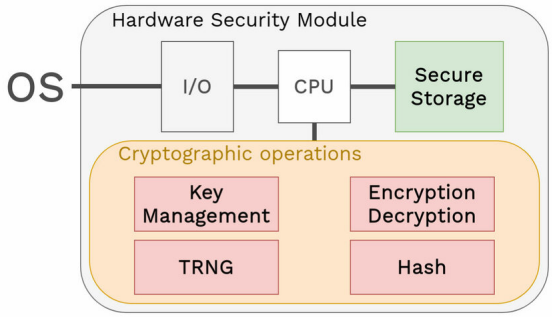
A. Hardware-based Security Mechanisms:

Hardware-based security features such as Trusted Platform Modules (TPM) [94], [95], Secure Enclaves (e.g., Intel SGX) [96], and Hardware Security Modules (HSM) [97], [98] provide a secure foundation for software execution. These components offer secure storage, cryptographic operations, and isolation mechanisms to protect sensitive data and code.

A Trusted Platform Module (TPM) operates as a secure cryptoprocessor that provides a hardware-based approach to managing and protecting cryptographic keys and other sensitive data. At its core, the TPM is designed to carry out cryptographic operations and securely store keys that protect information [94]. When a system with a TPM starts up, the module conducts a series of integrity checks to ensure that the system has not been tampered with. This process, known as the Trusted Boot (tBoot), involves validating each component of the startup process before it is loaded, ensuring that only trusted software is executed. The TPM can generate cryptographic keys that remain within the device; these keys can be used for various security functions but cannot be extracted by software. Additionally, the TPM can encrypt and decrypt data using these keys, providing secure storage that is resistant to external software attacks [95].



(a) A typical Trusted Platform Module Architecture. [99]



(b) Hardware Security Module Architecture. [100]

The TPM also supports remote attestation, creating a virtually tamper-proof environment. This feature allows the TPM to provide a cryptographic report of the hardware and software configuration of the host system to a remote verifier, ensuring that the system is secure and has not been altered. Furthermore, the TPM can seal and bind data, encrypting it in such a way that it can only be accessed on the same TPM with the same hardware configuration. This ability makes TPMs invaluable for scenarios requiring high levels of data security, such as in enterprise environments where ensuring the confidentiality and integrity of sensitive data is critical. Overall, the operation of a TPM enhances the security of a computing system by integrating hardware-based security measures that protect against unauthorized access and tampering.

However, exiting TPM systems possess several drawbacks including performance overhead, cost implications, complexity in management, compatibility issues etc.

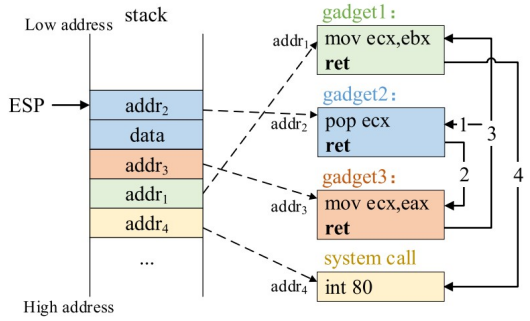


Fig. 21. An example Return on Programming(ROP) attack. [101]

program by altering the destination addresses of indirect jump or call instructions, thereby seizing control over the program's flow. To safeguard against such tampering, a linear encryption technique, such as XOR encryption, encrypts the instructions located at these target addresses, fortifying the integrity of the program's control flow.

When an indirect jump or call instruction is triggered, the instructions at the destination addresses undergo decryption using a decryption key generated through XOR encryption of key2 (as shown in Figure 22) and the address of the call site acquired from the PC register. As long as the program adheres to the paths outlined in the original Control Flow Graph (CFG), the decryption process will proceed accurately, enabling the program to operate smoothly. However, deviating from these paths may lead to a system error and failure of the Jump-Oriented Programming (JOP) technique.

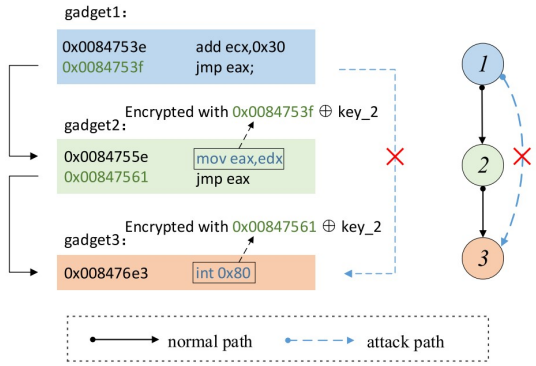


Fig. 22. An example of protecting the control flow of a program encrypting all first instructions at target addresses in the CFG. [101]

B. *Software-hardened Hardware:*

Software techniques such as Data Flow Integrity(DFI) [102], [103] , Data Execution Prevention (DEP) [104], and Control Flow Integrity (CFI) [105], [106], [107] can be implemented to harden hardware against various attacks. These measures make it more difficult for attackers to exploit vulnerabilities in hardware components. But often they include code-redundancy and data-redundancy.

An example control flow protection method is explained in Figure 22. Adversaries manipulate the control flow of a

C. *Hardware-assisted Sandboxing:*

Hardware virtualization technologies such as Intel VT-x [108] and AMD-V enable the creation of isolated execution environments, or sandboxes, where untrusted software can run safely. By leveraging hardware support for virtualization, these sandboxes provide strong isolation between applications and the underlying system.

Hardware-assisted sandboxing leverages specialized features within the hardware architecture to enhance the security and isolation of software applications. By utilizing hardware

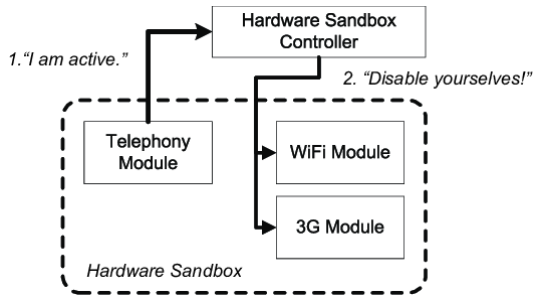


Fig. 23. An instance showcasing the application of hardware sandboxing is the prevention of real-time wiretapping over the internet. This is achieved by disabling all hardware modules necessary for Internet connectivity while a phone call is in progress. [109]

support, such as virtualization extensions in modern CPUs, sandboxing can create isolated environments, known as sandboxes, where untrusted or potentially malicious code can run safely without compromising the integrity of the host system. These hardware features enable the efficient implementation of sandboxing mechanisms, such as memory isolation, privileged access controls, and secure execution environments. As a result, hardware-assisted sandboxing offers robust protection against various security threats, including malware, exploits, and unauthorized access, thereby safeguarding sensitive data and critical system resources.

D. Fine-grained Access Control:

Hardware-enforced access control mechanisms, such as Memory Protection Units (MPUs) [110], [111] and Hardware-based Access Control (HBAC) [112], [113], restrict access to critical resources based on predefined security policies. These mechanisms prevent unauthorized access and limit the impact of software vulnerabilities.

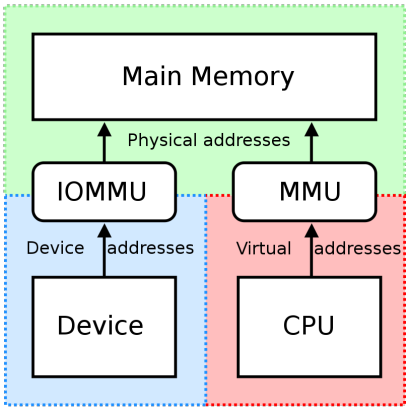


Fig. 24. A typical architecture of memory management unit. Memory management or protection unit is used as a fine grained HW resource distribution approach. [114]

The Memory Protection Unit (MPU) oversees processor transactions, such as instruction fetches and data accesses, and is capable of initiating a fault exception upon detecting an access violation(Figure 25). The primary objective of MPU is to restrict a process from accessing memory regions that have not been specifically allocated to it.

At its core, the MPU operates by defining and enforcing access permissions for various memory regions based on predefined rules and configurations. These rules typically include specifying the allowable types of access (e.g., read, write, execute) and the range of memory addresses accessible to each process or application.

When a processor executes instructions or accesses data in memory, the MPU monitors these transactions and compares them against the configured memory protection settings. If an access violation is detected such as an attempt to read from or write to a memory region that the process is not authorized to access the MPU triggers a fault exception which interrupts the normal flow of execution.

Fine-grained access control of hardware plays a crucial role in reducing software security vulnerabilities by providing granular control over the interactions between software components and hardware resources [114]. By allowing administrators to define precise rules governing access to hardware resources such as memory, input/output ports, and peripherals, fine-grained access control restricts the ability of malicious software to exploit hardware vulnerabilities for unauthorized access or privilege escalation [115], [116] . This approach enhances security by minimizing the attack surface exposed to potential exploits and mitigating the impact of software bugs or vulnerabilities that could otherwise be leveraged to compromise system integrity [117]. Furthermore, fine-grained access control facilitates the implementation of defense-in-depth strategies, where multiple layers of security mechanisms work together to protect against different types of threats, thereby strengthening the overall resilience of the system against cyberattacks.

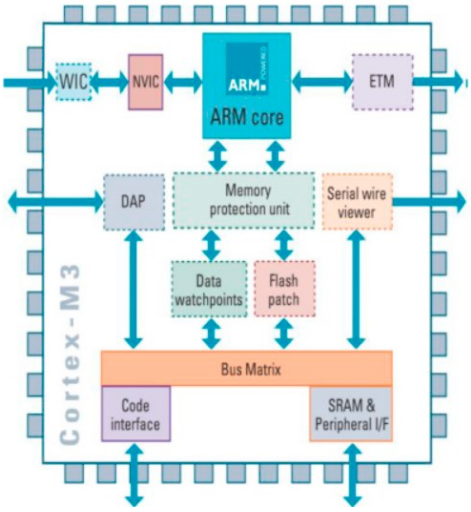


Fig. 25. Memory protection unit available in ARM-cortex M7. [118].

E. Hardware-accelerated Cryptography:

Hardware accelerators for cryptographic operations [119], such as AES-NI and SHA extensions in modern CPUs [120],

improve the performance and efficiency of cryptographic algorithms. By offloading cryptographic tasks to dedicated hardware, these accelerators reduce the attack surface and enhance overall system security.

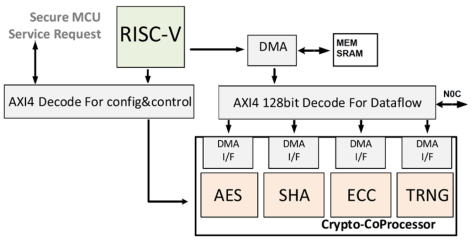


Fig. 26. Hardware architecture of a system with cryptographic accelerator. [121].

Hardware-accelerated cryptography represents a powerful approach to mitigating hardware and software security vulnerabilities by offloading cryptographic operations to specialized hardware components. By leveraging dedicated cryptographic processing units or accelerators integrated into modern hardware architectures, such as CPUs, GPUs, or dedicated cryptographic co-processors, hardware-accelerated cryptography enhances the efficiency and security of cryptographic operations while reducing the burden on software implementations. This not only improves the overall performance of cryptographic algorithms but also minimizes the exposure of sensitive cryptographic keys and operations to potential software-based attacks, such as side-channel attacks or malware exploits.

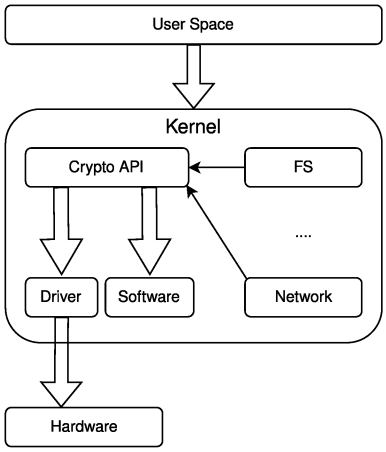


Fig. 27. A crypto API stack. [121].

For instance, when a user space application initiates a cryptographic operation through the Crypto API, the request is relayed to the kernel, which then determines whether to execute it using hardware or software, based on a priority-based hierarchy. While the Linux kernel itself provides a software implementation as an alternative to OpenSSL, hardware implementations typically take precedence due to their higher priority. Applications utilizing this API remain oblivious to the underlying method employed for cryptographic computation, allowing for potential acceleration via either hardware or more efficient software implementations, all without necessitating modifications to their code. The sole requirement is the inclusion of any new implementation into the existing priority list.

F. Hardware-based Intrusion Detection and Prevention:

Hardware-based intrusion [122] detection and prevention systems (IDPS) use specialized hardware components, such as network interface cards (NICs) [123] and programmable logic devices (FPGAs), to monitor and analyze network traffic in real-time. These systems detect and block malicious activities before they can compromise the system.

Hardware-based intrusion detection and prevention systems are a proactive cyber-security approach utilizing dedicated hardware components to detect and mitigate threats at the hardware level. These systems employ specialized hardware modules, like security co-processors or dedicated intrusion detection units, to continuously monitor system activity in real-time. By analyzing network traffic, system calls, memory access patterns, and other critical system events, hardware-based intrusion detection systems can identify anomalous behavior indicative of potential security breaches or unauthorized access attempts. Additionally, these systems can implement hardware-enforced security policies and access controls to prevent malicious activities from compromising system integrity. Operating at the hardware level offers several benefits, including reduced overhead, increased resilience against sophisticated attacks, and improved scalability across heterogeneous computing environments. In summary, hardware-based intrusion detection and prevention systems play a crucial role in strengthening cyber-security defenses, providing an extra layer of protection against evolving cyber-threats.

VI. Analysis of Co-Protection Methodologies: Proposed Approaches

Hardware can be protected through software feedback, and software can be protected using hardware feedback. Hardware-software co-protection methodologies aim to fortify systems against security threats by leveraging a combination of hardware sensors, power spectrum analysis, and information flow modeling. Hardware sensors can be integrated into devices to monitor physical and operational parameters, such as temperature, voltage fluctuations, and electromagnetic emissions, providing real-time insights into potential tampering or anomalies. Power spectrum analysis further enhances security by analyzing the electrical signals produced by hardware components to detect unusual patterns indicative of malicious activity. Meanwhile, information flow modeling involves mapping and controlling the pathways through which data travels within a system, ensuring that sensitive information is protected and that any unauthorized access or data leakage is promptly identified. By combining these approaches, systems can achieve a robust defense mechanism that not only detects and responds to threats in real-time but also proactively mitigates vulnerabilities through comprehensive monitoring and control.

A. *Hardware sensor based SW Protection:*

Hardware sensors are integrated into the physical components of the system, including processors, memory modules, input/output interfaces, and peripheral devices. These sensors continuously monitor physical parameters such as temperature, voltage, current, electromagnetic emissions, and other environmental conditions.

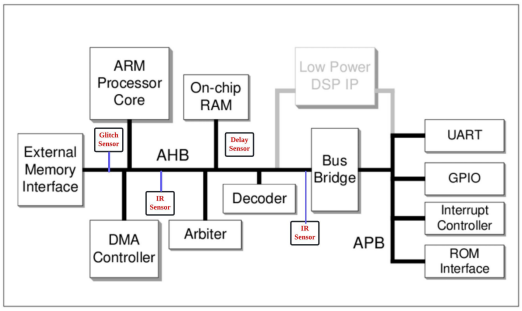


Fig. 28. Arm-SoC with inserted various sensors to capture activities and properties of various modules . [124]

Protecting software execution through hardware on-chip sensor placement involves strategically embedding sensors directly onto the integrated circuits of computer hardware to monitor and detect various aspects of software execution. These sensors can capture real-time data related to temperature, voltage fluctuations, electromagnetic emissions, and other physical characteristics of the hardware environment. By analyzing this data, hardware-based security mechanisms can detect anomalies indicative of unauthorized software execution, such as malicious code injection or runtime attacks. Leveraging hardware sensors offers several advantages, including low-level access to critical system components, reduced susceptibility to software-based attacks, and the ability to operate independently of the software stack, thus enhancing overall system security and resilience.

For detection and protection against several fault-injection attacks, different hardware sensors have been proposed. For instance, Fault-to-time converter(FTC) [125] sensor converts various non-invasive faults to delay and captures the encoded response in ZynQ FPGA. Laser fault injection sensor [126] captures the response of LASER injection into device.

Quantifying security properties of an RTL design and relating them to various injected faults and corresponding violations has been a long discussed issue. Security property driven vulnerability assessment framework against fault injection attacks(SoFI) [127] has been developed to co-relate specific security property violation due to fault injection attacks. How the on-chip sensors respond against faults and how they are related to security property violation has also been discussed [128].

To protect the SW execution using HW sensor response against FIAs, the injection attacks can be modeled in terms of internal quantities such as delay. When SW codes are running into hardware, or in a gate level synthesized design,

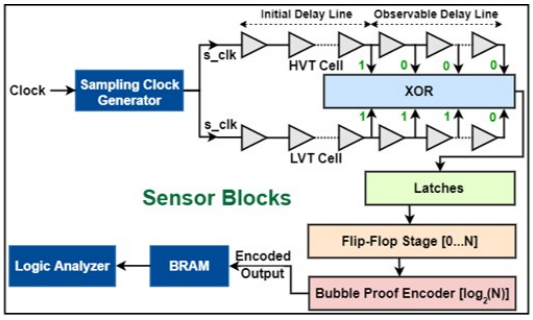


Fig. 29. Fault to time converter sensor. [125]

specific cells of the libraries are being used, each of which are vulnerable to the FIAs defined before. The vulnerability of each cell against each attack can be quantized. When a compiler transforms a high level code(C/C++) into assembly, it can be sensitive to each attack modeled previously. In this way SW execution can be sensitive to HW sensor response and can be protected by redundancy or any other methods. This method is displayed in Figure 34.

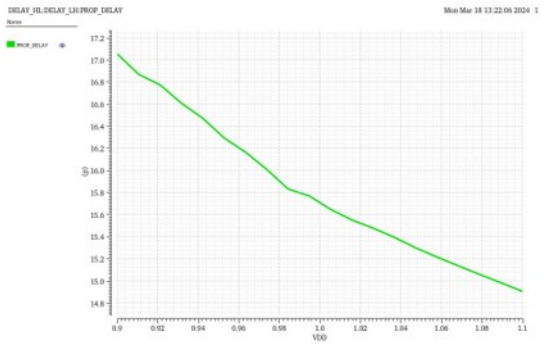


Fig. 30. Propagation delay vs supply voltage variation.

Modeling supply voltage variation involves simulating changes in the voltage supplied to the hardware components, which can occur due to fluctuations in the power supply or deliberate manipulation by attackers. This variation can lead to transient faults, where the voltage drops below the required threshold, causing errors or malfunctions in the system. Several previous research suggested that supply voltage variation can be modeled as a fluctuation in delay(Figure **??**). So, we can add a delay component in the standard cell library that can be activated at a certain time to show the effects of supply voltage variation.

Modeling laser fault injections in standard cells involves simulating the effects of laser-induced faults on the behavior of semiconductor devices within the cells. One approach to achieve this is by adding additional current components to the standard cell models, which represent the changes in device characteristics caused by laser irradiation. Modeling and validation of LFI into hardware involves several steps such as identifying vulnerable locations, defining fault models, integrating current components, simulate LFI attacks etc.

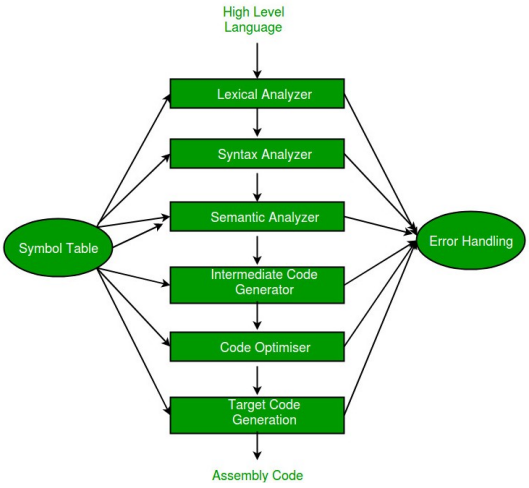


Fig. 31. Steps a compiler performs to translate a high level code to machine language.

The compiler plays a significant role in bridging the gap between software and hardware execution by translating high-level programming code into low-level instructions that can be understood and executed by hardware components. As software developers write code in languages like C, C++, or Python, the compiler analyzes the code, performs optimizations, and generates machine code tailored to the target hardware architecture(Figure 31). This machine code is then executed directly by the hardware, enabling the software's functionality to be realized efficiently. Additionally, modern compilers often incorporate optimization techniques like loop unrolling, instruction re-arrangement, and register relocation to maximize performance and minimize resource utilization.

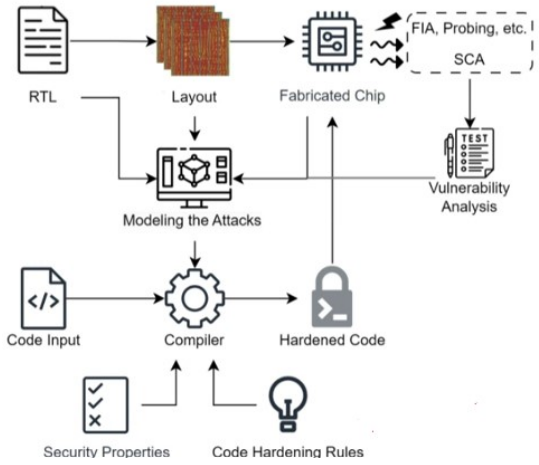


Fig. 32. During compilation, software can be hardened against physical attacks by this framework.

Compilers may play a vital role in mitigating hardware-software vulnerabilities by implementing various security mechanisms and optimizations during code generation. One fundamental approach is through the enforcement of memory safety checks, such as stack canaries, bounds checking, and address space layout randomization (ASLR) [129], which prevent buffer overflows and other memory corruption vulnerabilities. Additionally, compilers can implement control-flow integrity (CFI) mechanisms to detect and prevent code execution hijacking attacks, for instance: return-oriented programming and jump-oriented programming.

We can drive the compiler or generate extra plugins to make it aware of hardware vulnerabilities(Figure 32). The compiler may take the essence of modeling various hardware faults and be aware of hardware execution.

B. Instruction spectrum based software-hardware protection:

When a software runs on a hardware, it creates several property traces on a hardware, such as power, delay variation etc. These traces can be tracked to HW execution strategies and HWs can be made aware of the placement of sensors etc. Modifications of fault injection detection and prevention strategies in the hardware side can be done to prevent any attack that can for example, produce high power traces or any unusual activities during the SW execution.

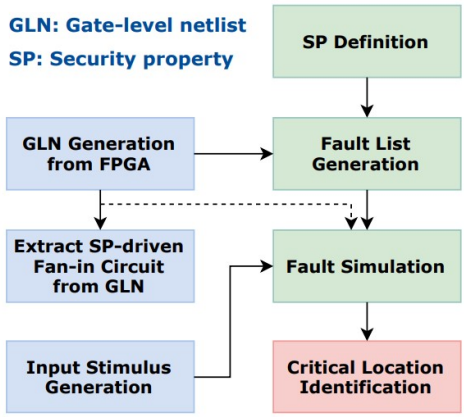


Fig. 33. Critical Location identification on HW using SoFI Framework. [127]

Developing a power model for assembly instructions involves quantifying the power consumption associated with executing each instruction within a processor. By analyzing the power spectrum derived from these models, software execution can be guided to optimize both hardware and software security simultaneously. The power spectrum reflects the power consumption patterns exhibited by different instructions during execution, offering insights into the energy requirements of various software operations.

Leveraging this information, software can be designed or modified to prioritize low-power instructions or sequences, reducing overall power consumption and minimizing the risk of hardware-based attacks, such as side-channel attacks. Additionally, by aligning software execution with the power spectrum, potential vulnerabilities in both hardware and software can be mitigated, as the power consumption characteristics of specific instructions can serve as indicators of potential security risks. This approach not only enhances hardware

security by reducing susceptibility to power-based attacks but also strengthens software security by optimizing execution patterns to minimize exposure to potential vulnerabilities.

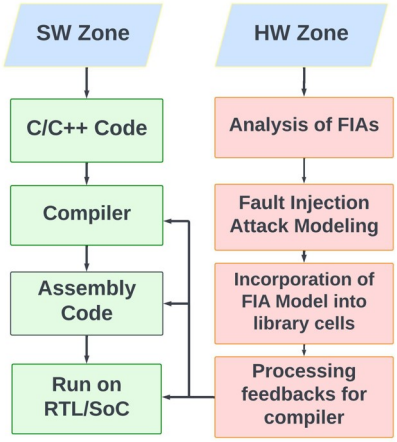


Fig. 34. SW protection by HW sensor feedback against Fault Injection attacks.

Building power model for each type of instruction(such as load,store from memory, arithmetic operations etc) enables us to construct power model for specific applications(Figure 35). If any application, such as matrix multiplication, or convolution operation for a CNN creates stress in the specific part of the hardware by drawing excessive power, we can trace that information in the hardware domain, and send it to the compiler so that software execution is aware of the hardware execution and power trace.

Software modules(such as sensors) running on the system assist in monitoring and analyzing the power consumption patterns. These software components collect power consumption data and perform statistical analysis to identify anomalies or suspicious behavior. They also define and manage security policies based on the observed power spectra and trigger appropriate responses to mitigate security risks.

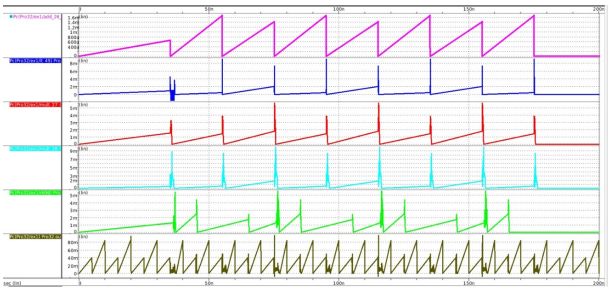


Fig. 35. Power traces of hardware blocks during execution of instructions in a processor and its peripherals.

C. Securing SW execution via monitoring HW Architectural Activity:

Securing software execution through hardware architectural activity is an effective strategy in modern computing systems. By integrating security measures directly into the hardware architecture, such as through hardware-based encryption, secure enclaves, or trusted execution environments, vulnerabilities at the software level can be mitigated more effectively. Hardware-based security provides a robust foundation for protecting sensitive data and critical processes from various threats, including malware, unauthorized access, and tampering.

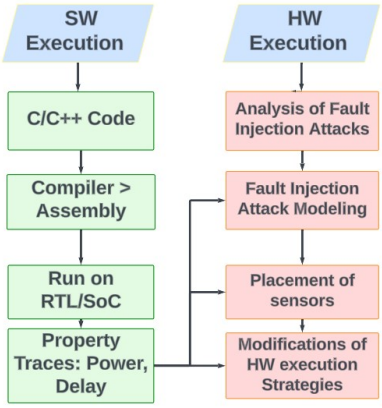


Fig. 36. HW protection by examining property traces during SW execution.

When a SW program runs into HW, such as a processor, different components of the processor gets 'activated' with the execution of instructions. As the execution of instruction sequences correlate with hardware resources such as registers, memory, computing units, corruption in any of the blocks will also hamper SW execution. Corruption in hardware can be in many forms. For instance, a data forwarding unit chooses the data to be put in ALU from execution or memory stage. If the forwarding unit is corrupted, wrong data can be put into the hardware even if we do the correct software execution (Figure 38).

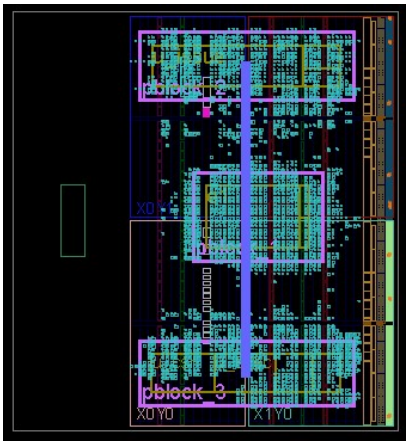


Fig. 37. FTC Sensor placed among various blocks of an SoC and the response is characterized

Hardware architectural activities can be collected via sensors also. In figure 37, the FTC sensor is placed between

different placed parts of a ZynQ SoC that has a core processor and peripherals like UART, SPI, timer etc. The sensor response changes when they are placed closer or further to some blocks those have relatively higher activities. Via placement of sensors and collecting their responses, we can send feedback to the software side that during execution of some particular instructions, some blocks have more vulnerability towards a bit flip or data failure.

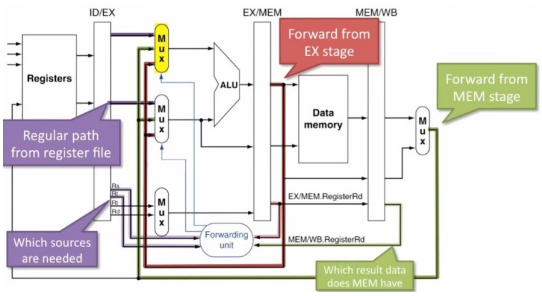


Fig. 38. Simple processor displaying the usage and probable corruption of a data forwarding path.

Securing software execution involves monitoring the activity of distinct hardware blocks within the system to detect and prevent potential security threats. By continuously monitoring the behavior of hardware components such as the CPU, memory modules, and input/output interfaces, anomalous activities indicative of malicious software behavior can be identified in real-time. This monitoring process typically involves analyzing metrics such as resource utilization, data access patterns, and communication protocols to detect unauthorized access attempts, abnormal program executions, or other suspicious activities. By integrating hardware-level monitoring mechanisms into the system architecture, software execution can be safeguarded against a wide range of security threats, including malware infections, code injections, and privilege escalation attacks.

D. Information flow modeling approach for HW/SW Security Verification

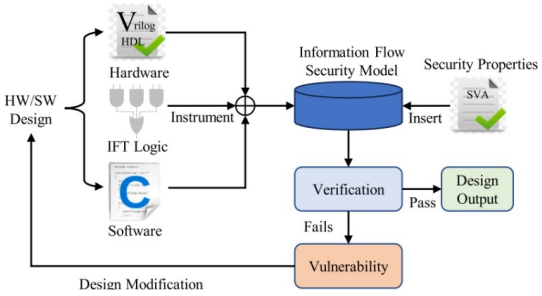


Fig. 39. Proposed design flow of HW/SW co-verification method

Information flow implication method is another approach we consider. Software programs are converted into hardware descriptions using a model and combined with hardware designs to form a logic circuit to track the information flow among software and hardware. So using data security labels, all logical information flows are captured. This flow can be incorporated into CAD verification tools and easily determine the violation of information flow security property, leading to detecting software, hardware, system level security vulnerabilities.

Security properties for HW/SW designs.

#	Security Vulnerabilities for SW/HW Designs	Implementation level	Security Properties	
			Confidentiality: Security assets can not leak to unclassified regions.	Integrity: Untrusted data can not flow trusted zones.
1	malicious/backdoor program	SW	√	√
2	insecure software function	SW		√
3	malicious logic or hardware Trojan	HW	√	√
4	hardware design flaw	HW	√	√
5	system-level hardware Trojan	HW and SW	√	√
6	timing side-channel	HW and SW	√	

Fig. 40. Security properties of HW/SW designs.

VII. Future Work and Conclusion

The exploration of hardware-software co-protection against interactive security threats offers a promising avenue for future research and development in the field of cybersecurity. As interactive security threats continue to evolve and grow in sophistication, it is imperative to develop comprehensive and integrated approaches that combine hardware and software mechanisms to mitigate these threats effectively.

One area for future work is the development of novel hardware architectures and co-design methodologies specifically tailored to address interactive security threats. Sensor based and machine learning based hardware-software property analysis are two prominent directions to protect both hardware and software. Collaborative efforts between hardware and software engineers will be essential to design and implement these advanced security features seamlessly within existing computing systems.

Additionally, future research should focus on the refinement and optimization of hardware-software co-protection mechanisms to achieve a balance between security, performance, and usability. This involves conducting extensive performance evaluations and benchmarking studies to assess the overhead and impact of security mechanisms on system performance, as well as user experience. Furthermore, exploring adaptive and dynamic security strategies that can respond to evolving threats in real-time will be crucial for ensuring the resilience of hardware-software co-protection solutions.

References

[1] R. Luna and S. A. Islam, "Security and reliability of safety-critical rtos," *SN Computer Science*, vol. 2, p. 356, Jun 2021.

[2] B. Lampson, "Computer security in the real world," *Annual Computer Security Applications Conference*, vol. 16, 12 2000.

[3] S. Duggineni, "Impact of controls on data integrity and information systems," pp. 29–35, 07 2023.

[4] H. Tabrizchi and M. Kuchaki Rafsanjani, "A survey on security challenges in cloud computing: issues, threats, and solutions," *The Journal of Supercomputing*, vol. 76, pp. 9493–9532, Dec 2020.

[5] H. Pearce, R. Karri, and B. Tan, "High-level approaches to hardware security: A tutorial," *ACM Trans. Embed. Comput. Syst.*, vol. 22, apr 2023.

[6] C. Dong, Y. Xu, X. Liu, F. Zhang, G. He, and Y. Chen, "Hardware trojans in chips: A survey for detection and prevention," *Sensors*, vol. 20, no. 18, 2020.

[7] M. T. Rahman, Q. Shi, S. Tajik, H. Shen, D. L. Woodard, M. Tehranipoor, and N. Asadizanjani, "Physical inspection attacks: New frontier in hardware security," in *2018 IEEE 3rd International Verification and Security Workshop (IVSW)*, pp. 93–102, 2018.

[8] M. Vidakovic and D. Vinko, "Hardware-based methods for electronic device protection against invasive and non-invasive attacks," *Electronics*, vol. 12, no. 21, 2023.

[9] M. Humayun, M. Niazi, N. Z. Jhanjhi, M. Alshayeb, and S. Mahmood, "Cyber security threats and vulnerabilities: A systematic mapping study," *Arabian Journal for Science and Engineering*, vol. 45, pp. 3171–3189, Apr 2020.

[10] Y. Su, M. Li, C. Tang, and R. Shen, "An overview of software vulnerability detection," 2016.

[11] M. Malenko and M. Baunach, "Hardware/software co-designed security extensions for embedded devices," in *Architecture of Computing Systems – ARCS 2019* (M. Schoeberl, C. Hochberger, S. Uhrig, J. Brehm, and T. Pionteck, eds.), (Cham), pp. 3–14, Springer International Publishing, 2019.

[12] J. Gu, B. Zhu, M. Li, W. Li, Y. Xia, and H. Chen, "A Hardware-Software co-design for efficient Intra-Enclave isolation," in *31st USENIX Security Symposium (USENIX Security 22)*, (Boston, MA), pp. 3129–3145, USENIX Association, Aug. 2022.

[13] E. Hozan, "Understanding the layers of a computer system," 02 2020.

[14] T. Huffmire, T. Levin, T. Nguyen, C. Irvine, B. Brotherton, G. Wang, T. Sherwood, and R. Kastner, "Security primitives for reconfigurable hardware-based systems," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 3, may 2010.

[15] R. Karam, R. Liu, P.-Y. Chen, S. Yu, and S. Bhunia, "Security primitive design with nanoscale devices: A case study with resistive ram," in *2016 International Great Lakes Symposium on VLSI (GLSVLSI)*, pp. 299–304, 2016.

[16] D. Kirovski, M. Drinić, and M. Potkonjak, "Enabling trusted software integrity," *SIGARCH Comput. Archit. News*, vol. 30, pp. 108–120, oct 2002.

[17] A. Maña, J. Lopez, J. J. Ortega, E. Pimentel, and J. M. Troya, "A framework for secure execution of software," *International Journal of Information Security*, vol. 3, pp. 99–112, Nov 2004.

[18] D. Dorfmeister, F. Ferrarotti, B. Fischer, E. Haslinger, R. Ramler, and M. Zimmermann, "An approach for safe and secure software protection supported by symbolic execution," in *Database and Expert Systems Applications - DEXA 2023 Workshops* (G. Kotsis, A. M. Tjoa, I. Khalil, B. Moser, A. Mashkoor, J. Sametinger, and M. Khan, eds.), (Cham), pp. 67–78, Springer Nature Switzerland, 2023.

[19] A. Gangolli, Q. H. Mahmoud, and A. Azim, "A systematic review of fault injection attacks on iot systems," *Electronics*, vol. 11, no. 13, 2022.

[20] S. Delarea and Y. Oren, "Practical, low-cost fault injection attacks on personal smart devices," *Applied Sciences*, vol. 12, no. 1, 2022.

[21] A. Gangolli, Q. H. Mahmoud, and A. Azim, "A systematic review of fault injection attacks on iot systems," *Electronics*, vol. 11, no. 13, 2022.

[22] S. Kaur, B. Singh, and H. Kaur, "Stratification of hardware attacks: Side channel attacks and fault injection techniques," *SN Computer Science*, vol. 2, p. 183, Mar 2021.

[23] C. Shepherd, K. Markantonakis, N. van Heijningen, D. Aboulkassimi, C. Gaine, T. Heckmann, and D. Naccache, "Physical fault injection and side-channel attacks on mobile devices: A comprehensive analysis," *Computers Security*, vol. 111, p. 102471, 2021.

[24] P. Prinetto and G. Roascio, "Hardware security, vulnerabilities, and attacks: A comprehensive taxonomy," in *Italian Conference on Cybersecurity*, 2020.

[25] Z. Kazemi, D. Hely, M. Fazeli, and V. Beroulle, "A review on evaluation and configuration of fault injection attack instruments to design attack resistant mcu-based iot applications," *Electronics*, vol. 9, p. 1153, 07 2020.

[26] K. Gomina, J.-B. Rigaud, P. Gendrier, P. Candelier, and A. Tria, "Power supply glitch attacks: Design and evaluation of detection circuits," in *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pp. 136–141, 2014.

[27] Y. Lu, "Injecting software vulnerabilities with voltage glitching," *ArXiv*, vol. abs/1903.08102, 2019.

[28] Z. Kazemi, D. Hely, M. Fazeli, and V. Beroulle, "A review on evaluation and configuration of fault injection attack instruments to design attack resistant mcu-based iot applications," *Electronics*, vol. 9, no. 7, 2020.

[29] B. Selmke, F. Hauschild, and J. Obermaier, "Peak clock: Fault injection into pll-based systems via clock manipulation," in *Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop*, ASHES'19, (New York, NY, USA), p. 85–94, Association for Computing Machinery, 2019.

[30] Z. Kazemi, A. Papadimitriou, I. Souvatzoglou, E. Ae, M. Ahmed, D. Hely, and V. Beroulle, "On a low cost fault injection framework for security assessment of cyber-physical systems: Clock glitch attacks," pp. 7–12, 07 2019.

[31] F. Cai, G. Bai, H. Liu, and X. Hu, "Optical fault injection attacks for flash memory of smartcards," in *2016 6th International Conference on Electronics Information and Emergency Communication (ICEIEC)*, pp. 46–50, 2016.

[32] D. Petryk, Z. Dyka, R. Sorge, J. Schaeffner, and P. Langendoerfer, "Optical fault injection attacks against radiation-hard registers," 06 2021.

[33] D. Z. Zabib, M. Vizentovski, A. Fish, O. Keren, and Y. Weizman, "Vulnerability of secured iot memory against localized back side laser fault injection," *2017 Seventh International Conference on Emerging Security Technologies (EST)*, pp. 7–11, 2017.

[34] Y. Zhou and D. Feng, "Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing," *IACR Cryptol. ePrint Arch.*, vol. 2005, p. 388, 2005.

[35] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Berlin, Heidelberg: Springer-Verlag, 2007.

[36] H. Gamaarachchi and H. Ganegoda, "Power analysis based side channel attack," 01 2018.

[37] N. Gattu, M. N. Imtiaz Khan, A. De, and S. Ghosh, "Power side channel attack analysis and detection," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–7, 2020.

[38] D. Agrawal, B. Archambeault, J. Rao, and P. Rohatgi, "The em side-channel(s):attacks and assessment methodologies," 12 2008.

[39] M. Tehranipoor, N. Nalla Anandakumar, and F. Farahmandi, *EM Side-Channel Attack on AES*, pp. 163–181. Cham: Springer International Publishing, 2023.

[40] R. Wang, H. Wang, and E. Dubrova, "Far field em side-channel attack on aes using deep learning," in *Proceedings of the 4th ACM Workshop on Attacks and Solutions in Hardware Security*, ASHES'20, (New York, NY, USA), p. 35–44, Association for Computing Machinery, 2020.

[41] Q. Ge, Y. Yarom, D. Cock, and G. Heiser, "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware," *Journal of Cryptographic Engineering*, vol. 8, pp. 1–27, Apr 2018.

[42] P. Kaushik and D. Majumdar, "Timing attack analysis on aes on modern processors," pp. 462–465, 09 2017.

[43] A. Tria and H. Choukri, *Invasive Attacks*, pp. 623–629. Boston, MA: Springer US, 2011.

[44] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE Design Test of Computers*, vol. 27, no. 1, pp. 10–25, 2010.

[45] S. Ranjani and N. Devi, "Malicious hardware detection and design for trust: An analysis," 2017.

[46] Y. Jin and Y. Makris, "Hardware trojan detection using path delay fingerprint," pp. 51–57, 07 2008.

[47] A. Das, G. Memik, J. Zambreno, and A. Choudhary, "Detecting/preventing information leakage on the memory bus due to malicious hardware," pp. 861–866, 03 2010.

[48] C. Dong, Y. Xu, X. Liu, F. Zhang, G. He, and Y. Chen, "Hardware trojans in chips: A survey for detection and prevention," *Sensors (Basel, Switzerland)*, vol. 20, 09 2020.

[49] E. Oriero, F. Khalid, and H. Syed, "Demist: Detection and mitigation of stealthy analog hardware trojans," in *Proceedings of the 12th International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '23, (New York, NY, USA), p. 47–55, Association for Computing Machinery, 2023.

[50] D. Sisejkovic and R. Leupers, *Hardware Trojans*, pp. 13–24. Cham: Springer International Publishing, 2023.

[51] H. Noman, "Code injection attacks in wireless-based internet of things (iot): A comprehensive review and practical implementations," *Sensor Review*, vol. 23, 06 2023.

[52] J. V. D. Ham, "Toward a better understanding of "cybersecurity"," *Digital Threats*, vol. 2, jun 2021.

[53] E. Dushku, J. Østergaard, and N. Dragoni, "Memory offloading for remote attestation of multi-service iot devices," *Sensors*, vol. 22, p. 4340, 06 2022.

[54] P. Maniriho, A. N. Mahmood, and M. J. M. Chowdhury, "A study on malicious software behaviour analysis and detection techniques: Taxonomy, current trends and challenges," *Future Generation Computer Systems*, vol. 130, pp. 1–18, 2022.

[55] M. Azeem, D. Khan, S. Iftikhar, S. Bawazeer, and M. Alzahrani, "Analyzing and comparing the effectiveness of malware detection: A study of machine learning approaches," *Heliyon*, vol. 10, no. 1, p. e23574, 2024.

[56] D. Mitropoulos and D. Spinellis, "Fatal injection: A survey of modern code injection attack countermeasures," *PeerJ Computer Science*, vol. 3, p. e136, 11 2017.

[57] L. Davi and A.-R. Sadeghi, *Building Control-Flow Integrity Defenses*, pp. 27–54. Cham: Springer International Publishing, 2015.

[58] K. Anjaria and A. Mishra, "Information leakage analysis of software: How to make it useful to it industries?," *Future Computing and Informatics Journal*, vol. 2, no. 1, pp. 10–18, 2017.

[59] J. Heusser and P. Malacaria, "Quantifying information leaks in software," pp. 261–269, 12 2010.

[60] I. Herrera Montano, J. J. García Aranda, J. Ramos Diaz, S. Molina Cardín, I. de la Torre Díez, and J. J. P. C. Rodrigues, "Survey of techniques on data leakage protection and methods to address the insider threat," *Cluster Computing*, vol. 25, Dec 2022.

[61] P. Falcarin, M. Baldi, and D. Mazzocchi, "Software tampering detection using aop and mobile code," 04 2004.

[62] B. Yu, M. Hu, Z. Sun, and B. Chen, "Data tampering attack design for ros-based object detection and tracking robotic platform," in *2021 International Conference on Control, Automation and Information Sciences (ICCAIS)*, pp. 159–164, 2021.

[63] S. Sayeed, H. Marco-Gisbert, I. Ripoll, and M. Birch, "Control-flow integrity: Attacks and protections," *Applied Sciences*, vol. 9, no. 20, 2019.

[64] J. M. Borky and T. H. Bradley, "Protecting information with cybersecurity," in *Effective Model-Based Systems Engineering*, pp. 345–404, Cham: Springer International Publishing, 2019.

[65] F. Cremer, B. Sheehan, M. Fortmann, A. N. Kia, M. Mullins, F. Murphy, and S. Materne, "Cyber risk and cybersecurity: a systematic review of data availability," *Geneva Pap. Risk Insur. Issues Pract.*, vol. 47, pp. 698–736, Feb. 2022.

[66] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 1–19, 2019.

[67] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, (Baltimore, MD), pp. 973–990, USENIX Association, Aug. 2018.

[68] V. van der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida, "Drammer: Deterministic rowhammer attacks on mobile platforms," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, (New York, NY, USA), pp. 1675–1689, Association for Computing Machinery, 2016.

[69] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T.-H. Lai, "Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution," *IEEE Security Privacy*, vol. PP, 01 2020.

[70] H. Bar-El, "Security implications of hardware vs software cryptographic modules," 01 2002.

[71] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: extracting the keys to the intel sgx kingdom with transient out-of-order execution," SEC'18, (USA), pp. 991–1008, USENIX Association, 2018.

[72] J. Bellardo and S. Savage, "802.11 Denial-of-Service attacks: Real vulnerabilities and practical solutions," in *12th USENIX Security Symposium (USENIX Security 03)*, (Washington, D.C.), USENIX Association, Aug. 2003.

[73] D. Chaudhary, K. Bhushan, and B. B. Gupta, "Survey on ddos attacks and defense mechanisms in cloud and fog computing," *International Journal of E-Services and Mobile Applications*, vol. 10, pp. 61–83, 07 2018.

[74] T. Gilmont, J.-D. Legat, and J.-J. Quisquater, "Enhancing security in the memory management unit," in *Proceedings 25th EUROMICRO Conference. Informatics: Theory and Practice for the New Millennium*, vol. 1, pp. 449–456 vol.1, 1999.

[75] A. Saravanan, S. Sathyabama, S. Kadry, and L. Ramasamy, "A new framework to alleviate ddos vulnerabilities in cloud computing," *International Journal of Electrical and Computer Engineering*, vol. 9, pp. 4163–4175, 10 2019.

[76] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "Ddos in the iot: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[77] Y. Al-Hadhrami and F. K. Hussain, "Ddos attacks in iot networks: a comprehensive systematic literature review," *World Wide Web*, vol. 24, pp. 971–1001, May 2021.

[78] E. Kim and H.-K. Choi, "Security analysis and bypass user authentication bound to device of windows hello in the wild," *Security and Communication Networks*, vol. 2021, pp. 1–13, 07 2021.

[79] D. Ray and J. Ligatti, "Defining code-injection attacks," vol. 47, pp. 179–190, 01 2012.

[80] A. Francillon and C. Castelluccia, "Code injection attacks on harvard-architecture devices," *Proceedings of the 15th ACM conference on Computer and communications security*, 2008.

[81] S. Schaik, M. Minkin, A. Kwong, D. Genkin, and Y. Yarom, "Cacheout: Leaking data on intel cpus via cache evictions," pp. 339–354, 05 2021.

[82] Y. Oyama and A. Yonezawa, "Prevention of code-injection attacks by encrypting system call arguments," 04 2006.

[83] M. Maniatakos, "Privilege escalation attack through address space identifier corruption in untrusted modern processors," in *2013 8th International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS)*, pp. 161–166, 2013.

[84] A. Ahmed. 2021.

[85] G. Khawaja, *Windows Privilege Escalation*, pp. 273–304. 2021.

[86] J. Antunes, N. Neves, and P. Veríssimo, "Detection and prediction of resource-exhaustion vulnerabilities," pp. 87–96, 11 2008.

[87] J. Antunes, N. F. Neves, and P. J. Veríssimo, "Detection and prediction of resource-exhaustion vulnerabilities," in *2008 19th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 87–96, 2008.

[88] M. Elsabagh, D. Barbará, D. Fleck, and A. Stavrou, "On early detection of application-level resource exhaustion and starvation," *Journal of Systems and Software*, vol. 137, pp. 430–447, 2018.

[89] S. Alhusayni and D. Alsuwat, "The buffer overflow attack and how to solve buffer overflow in recent research," vol. 2, pp. 5–11, 10 2020.

[90] M. A. Butt, Z. Ajmal, Z. I. Khan, M. Idrees, and Y. Javed, "An in-depth survey of bypassing buffer overflow mitigation techniques," *Applied Sciences*, vol. 12, no. 13, 2022.

[91] "System memory protection and vulnerability assessment in presence of software attacks," 2021.

[92] C. Canella, J. Van Bulck, M. Schwarz, M. Lipp, B. Von Berg, P. Ortner, F. Piessens, D. Evtyushkin, and D. Gruss, "A systematic evaluation of transient execution attacks and defenses," in *Proceedings of the 28th USENIX Conference on Security Symposium*, SEC'19, (USA), pp. 249–266, USENIX Association, 2019.

[93] V. Srihith Indla, A. Donald, T. Aditya, T. A. Srinivas, D. Anjali Reddy, and A. Chandana, "Firmware attacks: The silent threat to your iot connected devices," *International Journal of Advanced Research in Science, Communication and Technology*, vol. 3, pp. 2581–9429, 04 2023.

[94] A. Hoeller and R. Toegl, "Trusted platform modules in cyber-physical systems: On the interference between security and dependability," in *2018 IEEE European Symposium on Security and Privacy Workshops (EuroSPW)*, pp. 136–144, 2018.

[95] J. D. Osborn and D. C. Challener, "Trusted platform module evolution," *Johns Hopkins APL Technical Digest (Applied Physics Laboratory)*, vol. 32, no. 2, pp. 536–543, 2013.

[96] W. Zheng, Y. Wu, X. Wu, C. Feng, Y. Sui, X. Luo, and Y. Zhou, "A survey of intel sgx and its applications," *Frontiers of Computer Science*, vol. 15, 06 2021.

[97] S. Mavrovouniotis and M. Ganley, "Hardware security modules," *Secure Smart Embedded Devices, Platforms and Applications*, pp. 383–405, 06 2013.

[98] M. Sommerhalder, *Hardware Security Module*, pp. 83–87. Cham: Springer Nature Switzerland, 2023.

[99] N. Haimbala, *Avoiding Dark Cloud: Secure Storage and Trusted Computing*. PhD thesis, 09 2016.

[100] K. Malinka, O. Hujnák, P. Hanacek, and L. Hellebrandt, "E-banking security study - 10 years later," *IEEE Access*, vol. 10, pp. 1–1, 01 2022.

[101] L. Davi, A. Dmitrienko, M. Egele, T. Fischer, T. Holz, R. Hund, S. Nürnberger, and A.-R. Sadeghi, "Poster: control-flow integrity for smartphones," in *Conference on Computer and Communications Security*, 2011.

[102] L. Feng, J. Huang, J. Huang, and J. Hu, "Toward taming the overhead monster for data-flow integrity," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 27, nov 2021.

[103] T. Lu and J. Wang, "Data-flow bending: On the effectiveness of data-flow integrity," *Computers Security*, vol. 84, pp. 365–375, 2019.

[104] N. Stojanovski, M. Gusev, D. Gligoroski, and S. Knapskog, "Bypassing data execution prevention on microsoftwindows xp sp2," in *The Second International Conference on Availability, Reliability and Security (ARES'07)*, pp. 1222–1226, 2007.

[105] S. Sayeed, H. Marco-Gisbert, I. Ripoll, and M. Birch, "Control-flow integrity: Attacks and protections," *Applied Sciences*, vol. 9, no. 20, 2019.

[106] T. Mishra, T. Chantem, and R. Gerdes, "Survey of control-flow integrity techniques for real-time embedded systems," *ACM Trans. Embed. Comput. Syst.*, vol. 21, oct 2022.

[107] N. Burow, S. A. Carr, J. Nash, P. Larsen, M. Franz, S. Brunthaler, and M. Payer, "Control-flow integrity: Precision, security, and performance," *ACM Comput. Surv.*, vol. 50, apr 2017.

[108] M. Klanovicz Ferreira, H. Freitas, and P. Navaux, "From intel vt-x to mips: An archc-based model to understanding the hardware virtualization support," 06 2008.

[109] Q. Yan, Y. Li, T. Li, and R. Deng, "Insights into malware detection and prevention on mobile phones," vol. 58, pp. 242–249, 12 2009.

[110] M. Grisafi, M. Ammar, and B. Crispo, "On the (in)security of memory protection units : A cautionary note," in *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*, pp. 157–162, 2022.

[111] O. Stecklina, P. Langendoerfer, and H. Menzel, "Design of a tailor-made memory protection unit for low power microcontrollers," 06 2013.

[112] L. Bossuet and A. Cherkaoui, "Hardware access control in constrained environments," 10 2014.

[113] N. Anciaux, L. Bouganim, and P. Pucheral, "A hardware approach for trusted access and usage control," 01 2008.

[114] K. Albulayhi, A. Abuhussein, F. Alsubaei, and F. T. Sheldon, "Fine-grained access control in the era of cloud computing: An analytical review," in *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 0748–0755, 2020.

[115] J. Li and X. Chen, "Fine-grained access control system based on outsourced attribute-based encryption," in *Computer Security – ESORICS 2013*, pp. 592–609, 2013.

[116] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *2010 Proceedings IEEE INFOCOM*, pp. 1–9, 2010.

[117] R. Zhang, H. Ma, and Y. Lu, "Fine-grained access control system based on fully outsourced attribute-based encryption," *Journal of Systems and Software*, vol. 125, pp. 344–353, 2017.

[118] T. Martin, ed., *Chapter 4 - Cortex Microcontroller Software Interface Standard*, Newnes, 2016.

[119] J. Chang, C. Liu, and J.-L. Gaudiot, "Hardware acceleration for cryptography algorithms by hotspot detection," vol. 7861, 05 2013.

[120] E. G. AbdAllah, Y. R. Kuang, and C. Huang, "Advanced encryption standard new instructions (aes-ni) analysis: Security, performance, and power consumption," in *Proceedings of the 2020 12th International Conference on Computer and Automation Engineering*, ICCAE 2020, (New York, NY, USA), pp. 167–172, Association for Computing Machinery, 2020.

[121] L. Leonardi, G. Lettieri, P. Perazzo, and S. Saponara, "On the hardware-software integration in cryptographic accelerators for industrial iot," *Applied Sciences*, vol. 12, no. 19, 2022.

[122] R. Abdulhammed, M. Faezipour, and K. Elleithy, "Network intrusion detection using hardware techniques: A review," pp. 1–7, 04 2016.

[123] M. Thanoun, *Analysis and Design of Network Interface Card for Computer Network Using Network Processor and Simevent Technique, Ph.D. Thesis Authors Supervised by Assistant Professor Dr. A.I.A. Jabbar Mohammed Younis Thanoun Publication date 2011 Institution University of Mosul*. PhD thesis, 07 2011.

[124] H. Hellmich, A. Erdogan, and T. Arslan, "Re-usable low power dsp ip embedded in an arm based soc architecture,"

[125] M. R. Muttaki, T. Zhang, M. Tehranipoor, and F. Farahmandi, "Ftc: A universal sensor for fault injection attack detection," in *2022 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 117–120, 2022.

[126] M. Ebrahimabadi, S. S. Mehjabin, R. Viera, S. Guilley, J.-L. Danger, J.-M. Dutertre, and N. Karimi, "Detecting laser fault injection attacks via time-to-digital converter sensors," in *2022 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 97–100, 2022.

[127] H. Wang, H. Li, F. Rahman, M. M. Tehranipoor, and F. Farahmandi, "Sofi: Security property-driven vulnerability assessments of ics against fault-injection attacks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 3, pp. 452–465, 2022.

[128] M. R. Muttaki, B. Barker, M. Tehranipoor, and F. Farahmandi, "Ftc: A universal low-overhead fault injection attack detection solution," pp. 386–391, 10 2022.

[129] I. Articles, "Address space layout randomization," 2021.