SoK: A Security Architect's View of Printed Circuit Board Attacks

Jacob Harrison
Bloomberg L.P., New York, NY. Email: jharrison183@bloomberg.net

Nathan Jessurun
Terraverum, Austin, TX. Email: nathan@terraverum.ai

Mark Tehranipoor
University of Florida, Gainesville, FL. Email: tehranipoor@ece.ufl.edu

Abstract

Many recent papers have proposed novel electrical measurements or physical inspection technologies for defending printed circuit boards (PCBs) and PCB assemblies (PCBAs) against tampering. As motivation, these papers frequently cite Bloomberg News' "The Big Hack", video game mod-chips, and "interdiction attacks" on IT equipment. We find this trend concerning for two reasons. First, implementation errors and security architecture are rarely discussed in recent PCBA security research, even though they were the root causes of these commonly-cited attacks and most other attacks that have occurred or been proposed by researchers. **This suggests that the attacks may be poorly understood.** Second, if we assume that novel countermeasures and validation methodologies are tailored to these oft-cited attacks, then **significant recent work has focused on attacks that can already be mitigated instead of on open problems.**

We write this SoK to address these concerns. We explain which tampering threats can be mitigated by a PCBA security architecture. Then, we enumerate assumptions that security architecture depends on. We compare and contrast assurances achieved by security architecture vs. by recently-proposed electrical or inspection-based tamper detection. Finally, we review over fifty PCBA attacks to show how most can be prevented by proper architecture and careful implementation.

1 Introduction

When a digital security architect looks at a PCBA, they see interactions between digital processors[1] and other subsystems. These interactions may be placed into two categories as shown in Figure 1: those that *can* be protected by cryptography-based security architecture and those that *cannot*. Architectural defenses for interactions that support cryptography are described in the following paragraphs.[2]

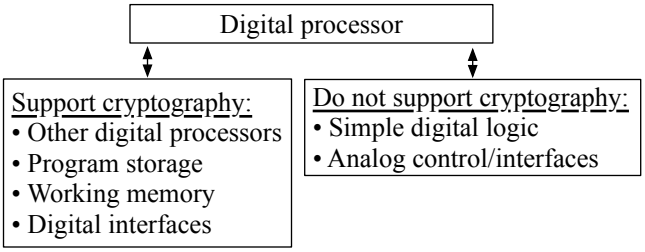

Figure 1: Only interactions where both parties can use cryptographic protocols can be protected by security architecture.

Security architecture can prevent adversaries from impersonating or replacing legitimate components, and it can prevent snooping or tampering with inter-component communications. When two digital processors interact, they can authenticate each other using public key infrastructure (PKI) or shared secrets, then use cryptographic protocols to set up an encrypted and integrity-protected channel. Adversaries cannot impersonate or substitute components unless they can learn real authentication keys. Similarly, a secure channel between components protects data confidentiality as it crosses the PCB and enables processors to detect tampered communications based on secrets that a board-level adversary will not know.

Snooping and modification of data stored in non-volatile or working memory can be prevented. When a processor reads a program or data from non-volatile storage (e.g., EEPROM, flash, or a hard disk), it can verify the authenticity and integrity of the code/data using a signature (HMAC, RSA, ECC, etc.). This prevents PCBA attacks that hijack the processor by modifying stored code or data. Similarly, signatures can prevent PCBA attacks that corrupt code or data in working

[1] We use the term "digital processor" loosely to refer to a chip that has digital inputs and outputs, can execute cryptographic protocols, and can condition its execution on signature verifications and integrity checks. This could be a general purpose CPU, a microcontroller, an FPGA, an ASIC, etc.

[2] Note that we do not equate 'security' with 'cryptography'. In the words of Peter Neumann: "if you think cryptography is the answer to your problem, then you don't know what your problem is" [76]. Rather, our point is that many PCBA attacks can be prevented by a security architecture that accounts for board-level threats.

memory (e.g., DRAM). To prevent snooping, data can always be encrypted before it leaves a processor and decrypted when it gets read back.

Peripherals can be authenticated, communications with them can be secured, and sound design can protect a system from malicious peripherals. When a processor interacts with peripheral devices (e.g., via USB or PCIe) or with a remote computer, it can use the same techniques described in the preceding paragraphs to either (a) authenticate and encrypt/integrity protect communication with the device, or (b) ensure that data written or read from the peripheral stays secret and is not corrupted. If requirements dictate that a processor must interact with arbitrary (untrusted) devices, the system can be designed to limit peripherals' access to critical resources, e.g., using an IOMMU.

On the other hand, **the digital architect cannot protect their processor's interactions with simple digital logic** (e.g., glue logic, push-buttons) **or analog circuits** (e.g., analog control, analog transducers, power regulators). This means that PCBA adversaries can potentially use these unprotected inputs to manipulate a processor, and they can prevent a processor from properly controlling analog or simple digital peripherals. If the architect is lucky, it will be possible to use heuristics or human intervention to prevent unprotected inputs from manipulating a system into doing bad things.

2 Systematization of PCBA tamper defenses

This section explains PCBA security architecture's foundational assumptions, then explores when assurances based on security architecture complement, and are redundant with, those achieved by electrical and optical tamper detection.

2.1 PCBA security architecture assumptions

The design patterns from Section 1 assume that **signals inside a chip are safe from PCBA attacks.** We assess that this is a reasonable assumption for most threat models, and that it is a *beneficial* assumption even in threat models where it is not true.

2.1.1 IC vs. PCBA attacks

Chip-level tampering is performed against the silicon or metal layers of an integrated circuit (IC) die. Examples include "Trojan horse" ICs, FIB edits, microprobing, and photon emission microscopy. These attacks cause a *chip* to violate its functional or security requirements: it either operates incorrectly with respect to its inputs or leaks internal state that could not have been inferred from the untampered chip.

On the other hand, board-level attacks operate on a PCBA's components, traces, or substrate. Examples include component addition, removal, or substitution, as well as changes to substrate or routing. Board-level adversaries are limited to manipulating or monitoring a chip's inputs and outputs to coax it into breaking the system's security.

Some might question whether our chip/board boundary accommodates side channel analysis and fault injections, which seem to infer or influence a chip's internal state even though they are board-level attacks. We believe it does: although these attacks exploit more arcane aspects of a chip's operation than, e.g., changing signals on a bus, they still use behaviors of untampered chips to break system security. In other words, operating voltage, electromagnetic fields, reset lines, etc. are simply more signals an adversary may manipulate or monitor. The security architect is responsible for understanding and mitigating their system's weaknesses to fault injection and side channels, just as with more ordinary tampering attacks.[3]

2.1.2 On the "hardware root of trust"

Hardware security folklore holds that tampering attacks undermine a system's "hardware root of trust" [12, 66]. In other words, they undermine an assumption that certain parts of a system are secure because they are outside an attacker's reach. We perceive that PCBA security researchers have generally accepted that this is true, which may explain why there has been so little attention paid to security architecture for PCBA tamper prevention.

Security architecture needs a root of trust to enforce restrictions on, and establish trust in, other parts of a system. In traditional cybersecurity, which focuses on *software* attacks, hardware is said to be a root of trust because software adversaries cannot change the metal and silicon that defines hardware behavior. On the other hand, when studying attacks *on hardware*, we must consider the possibility that attackers could tamper with a system's physical construction.

If PCBA tampering undermines a system's root of trust, then architectural defenses are hollow. Happily, we assess that this is not the case. The phrase "hardware root of trust" is too reductive; if finer distinctions are made, **hardware that is difficult to attack can serve as a root of trust against**

[3] We assume the reader is familiar with most of the architectural defenses we discuss, but we offer additional explanation of how the architect may defend against fault injection and side channels.

To mitigate fault injection, a security architect might specify that certain data reads or conditional instructions be performed multiple times with random delays inserted between each iteration. They might also be mindful of processor outputs that can serve as stable timing references before some security-critical event. If they have the luxury of working with chip designers, they can also ask for internal filtering on reset signals and power rails to change components' basic vulnerability to fault injection.

Side channels are trickier to mitigate in general [94], but, as we understand the situation, there are two classes of side channels: (1) those that leak keys given very few traces, and (2) those that require traces from many thousands of cryptographic operations. The security architect must instruct their implementors to avoid the first class of side channels. Completely mitigating the second class is challenging, but susceptibility to these attacks can be reduced by designing protocols that use keys that are frequently rotated, or by ensuring that adversaries cannot coax a system into performing many operations that use long-lived keys (e.g., by using long-lived keys only in response to a signed, replay-protected message).

attacks on more vulnerable hardware. This distinction is important because, for reasons we explain in the next subsection, PCBA tampering is a more widespread concern than IC tampering, so different systems may have more or less sophisticated adversaries.

2.1.3 Why ICs are a meaningful trust boundary

Attacking PCBAs is significantly cheaper and easier than attacking ICs.

It is *cheaper* because the features an attacker must manipulate on a PCBA are a hundred to a million times larger than on a IC. As a result, IC tampering tools (e.g., microprobing stations, FIBs, PHEMOS) are more sophisticated to build, more costly to procure and operate, and are therefore less accessible than those required for PCBA attacks (e.g., JTAG emulators, soldering irons, and stereoscopes).

It is *easier* because signals of interest on a PCBA are at a higher level of abstraction than on ICs. Whereas PCBAs are composed of tens to thousands of *components*, ICs may contain more than a billion *transistors*. The reverse engineering required to understand which transistors must be modified to backdoor a chip dwarfs that required to understand which signals on a PCBA can be compromised.

Considering the difficulty of attacking ICs, many security architects make a risk assessment that an adversary's benefit from a successful IC attack would not justify the attack's cost and complexity (or, equivalently, that the harm from system compromise does not justify the expense and complexity of implementing defenses against IC attacks). On this basis, they exclude IC tampering from their threat model.[4] *Replacement* of security-critical chips remains a concern, so ICs must be authenticated, but once a chip's identity is confirmed it is explicitly assumed to meet the manufacturer's functional description, quality standards, and security requirements.

For highly-critical systems with powerful (nation-state) adversaries, IC attacks become a realistic threat, so countermeasures against both PCBA and IC tampering are needed. However, even in this case, security architects benefit from modeling ICs as roots of trust and mitigating board-level threats via security architecture where possible. IC tamper detection will be needed regardless of how board-level threats are handled, and using security architecture to convert potential PCBA tampering into IC tampering will only make an adversary's life more difficult.

2.2 Security architecture vs. tamper detection

At the end of the day, architectural, inspection-based, and electrical tamper defenses are different ways to make physical attacks costly and difficult. Security architecture accomplishes this by converting PCBA attacks into IC attacks, while electrical and optical countermeasures aim to detect board-level probing and hardware modification. These approaches can be complementary, or redundant. This section further explores strengths and gaps in each approach.

2.2.1 A brief review of tamper detection

Recent work has focused on two kinds of tamper detection: **(1) Electrical characterization**, and **(2) physical inspection**.

Electrical characterization Electrical detection senses parameter changes caused by hardware modification or probing.

Many approaches try to measure intrinsic electrical variations from components and the manufacturing process that are (1) difficult to mimic and (2) likely to change if a system is tampered. Various measurements and equipment/sensors have been proposed, such as impedance [28, 47, 63, 75, 111], resonant frequencies [70], signal reflections [109], propagation delay [78, 81, 108], and more. A recent trend is to characterize impedance over a frequency range [74, 89, 90, 114].

Other techniques measure dynamic parameters in search of anomalies that could indicate tampering or counterfeit components. For example, [22] measures a running circuit's EM emanations, and [9, 84] measure dynamic power. Similarly, [68] proposes monitoring a digital processor's control flow via its EM leakage.

A final class of electrical tamper detection uses an **active enclosure** around sensitive subsystems to detect penetration attempts [55], similar to the old IBM HSMs [5].

Physical inspection Inspection-based tamper detection takes images of a board and searches them for signs of tampering. Innovations in this area have focused on novel imaging modalities and automated inspection algorithms.

Different imaging modalities reveal different information about a board. Lots of recent work has focused on analyzing images from simple visual light cameras [56], which quickly and cheaply reveal colors, textures, part markings and logos, silkscreen, and other features. X-ray, both two-dimensional and computed tomography, has been investigated for viewing PCB connectivity, inspecting inside components, and finding components hidden between PCB layers [25]. Other, more exotic modalities have also been proposed, such as terahertz for material analysis [102]. A summary of research on additional modalities may be found in [71].

Images of a board are analyzed with computer vision, image processing, and AI algorithms. Different algorithms process information from different modalities, but visual light and x-ray inspection have received the lion's share of attention because these are the most mature imaging technologies.

[4] For an example, we look to video game modchips, which have been widely cited in PCB security research: Tony Chen says that Microsoft was not worried about IC tampering when designing the Xbox One, but that "every exposed pin" on the PCBA was an attack surface [19]. This was based on an assessment that IC tampering costs more than several video games.

For visual light images, algorithms for component classification [21], board text, part number, and logo extraction [93], texture analysis [27, 41], pin counting and measurement [57], laser mark characteristics [49], and many others have been proposed to assist in deciding whether a board is as expected. For X-ray inspection, researches have focused on extracting PCB connectivity [16, 61]. At least nine different PCB image data sets have been gathered and annotated to contribute to machine learning model training [56], including of x-ray images [72], and tools have been built to accelerate and refine annotation of new data [58].

Online vs. offline Offline techniques check for attacks at a moment in time but do not protect a system while it is in service. Online techniques protect a system continuously once they are installed and activated. Offline electrical techniques use bench top instruments, ranging from multimeters, to oscilloscopes, to vector network analyzers (VNAs), whereas online approaches incorporate sensing circuitry into a system's design. Most inspection-based approaches are offline, but a few online approaches, e.g., using IR to monitor for temperature changes that could indicate runtime anomalies [71], have been proposed.

Golden-based vs. golden-free Tamper detection may compare measurements of a suspect system to a "golden" model, or to an earlier measurement of the system. Often, golden models are difficult to obtain because they require either (a) extensive *a priori* knowledge of a system and sophisticated modeling, or (b) a trusted sample from which measurements can be taken. Additionally, golden-based analysis is challenging because measurements must simultaneously be *sensitive* to changes caused by hardware manipulation and *insensitive* to changes from environmental or processing variation. In electrical tamper detection, both golden-based and golden-free techniques have been proposed. Physical inspection usually needs golden images because reliable measurements for golden-free assessment are difficult to obtain.[5]

2.2.2 Reasoning about security

Even though architecture and tamper detection both work by increasing attack cost and difficulty, reasoning about security from tamper detection requires more knowledge of attacks and measurements of countermeasure performance.

Security architecture operates on mathematical models. Systems can be abstracted as automata, security properties can be formally specified, and exploits are, to use the words of Shubina and Bratus, "constructive proofs" that a system's architecture or implementation do not satisfy its security requirements [18]. When dealing with such models, it is not important whether a malicious device is large, small, or buried between the layers of a PCB. E.g., if the wire targeted by a PCBA attack carries encrypted and authenticated data, it will not be possible to snoop data carried over the wire or to modify it without detection, no matter how physically subtle an attack may be. This abstraction is enabled by the model of ICs as trust boundaries. In sum, **architectural security is indifferent to an attack's physical properties.**

Conversely, electrical and optical tamper detection make *physical measurements* of an attack. This means that factors like the amount of capacitance that a probe adds to a victim circuit, or whether an attack is implemented on the surface of a PCB or between its layers, materially impact countermeasure performance. More accurate tamper detection schemes can detect more subtle attacks, thereby increasing cost and difficulty of tampering. Also, tamper detection is affected by uncertainties from measurements, manufacturing process variation, and the environment.[6] Thus, **to reason about security from tamper detection, we need physical measurements of how subtle attacks can be, how sensitive a particular tamper detection technique is, and how much uncertainty should be expected in measurements.**

Unfortunately, we currently lack much of this information. Little research has been conducted on how PCBA attacks can be made subtle. We have found few works that use physical measurements to describe disturbance caused by realistic attacks or tamper detection performance, and we know of no comprehensive studies of environmental or manufacturing uncertainties in tamper detection. Instead, countermeasure performance is typically expressed in terms of concrete test cases detected, and most countermeasures are evaluated with different test cases. Validation using ad-hoc test cases, or with benchmarks whose relative difficulties and relation to the general tampering threat is unknown, make it difficult to compare different techniques' performance or assess how effective tamper detection techniques will be against realistic attacks.

2.2.3 Picking appropriate defenses

With all of this background in mind, we consider strengths and weaknesses of tamper detection and architecture. Different defensive strategies are appropriate for different systems and threat models, but as a general rule of thumb:

Start by applying the security architecture techniques from Section 1 to protect a system's digital functions. There are several reasons that security architecture should be a system's primary defense where it can be applied. (1) Some

[5] Optical traits such as device footprints, chip texture, marking characteristics, etc. are rarely specified in enough detail in datasheets to provide meaningful golden-free measurements. Moreover, allowable tolerances on each feature (pin pitch/length/spacing, package height/width) vary significantly, making it harder to identify suspicious components in a golden-free environment.

[6] Note that the presence of measurement uncertainty implies security assessments based on tamper detection should be considered *statistical* problems. As tamper detection research matures, we hope to see countermeasures evaluated in terms of the *likelihood* that they detect a particular attack.

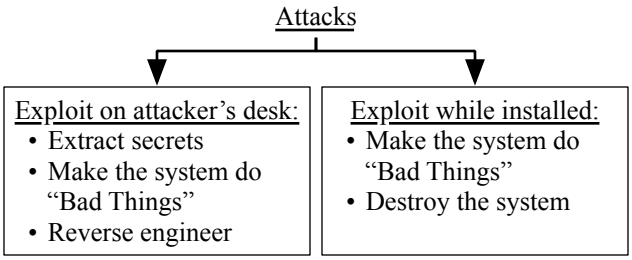

Figure 2: Depending on what attacks must be stopped and when attacks are anticipated, different defenses are needed.

secure PCBA design patterns, such as cryptographic firmware verification and integrity checks of critical data, will already be needed because they protect a system against software-only/remote attacks. (2) Because it has been used for years to prevent software exploits, security architecture is well-understood, widely used, and supported by commercial hardware. (3) The cost and difficulty that proper security architecture incurs to attackers is easy to reason about. (4) With a bit of support from component vendors, architectural defenses can secure a system through its entire lifecycle, from before a PCBA is assembled[7] to when it sits in an adversary's lab.

Next, **design tamper detection to fill security architecture gaps or provide defense in depth.** We consider four ways that electrical or inspection-based tamper detection can complement or augment security architecture. (1) Tamper detection must be used to defend analog or simple digital functions because they do not support cryptography, as explained in Section 1. (2) There are some attacks on system availability that security architecture cannot prevent. (3) Tamper detection may enable a system to respond to sophisticated secret extraction attacks. (4) Defense-in-depth, where tamper detection and security architecture are deployed redundantly, may be desired as a hedge against implementation errors or security architecture failures.

Finally, **recognize that some tamper detection approaches only work on certain kinds of attacks.** It is important to ensure that implemented tamper detection techniques match threats to-be-defended.

System lifecycle considerations Figure 2 enumerates four goals an attacker might have for a tampering attack. Some goals only require an attack to succeed once, while a PCBA is in the adversary's hands. Others require that hardware modifications persist until the PCBA is installed and in service.

Offline tamper detection techniques can stop attacks that occur before a system is fielded, but they cannot protect a system while it is on the attacker's desk. To protect a system after it enters service, security architecture or online tamper detection are required. However, even though there are online inspection-based defenses, these can only protect a PCBA that stays within its imaging environment. Therefore, **to prevent attacks performed on an attacker's desk, architecture or online electrical tamper detection are needed.**

If a system's threat model includes manufacturing-time attacks from a board fabricator, assembler, or component vendor, **"golden-based" tamper detection or security architecture are needed to establish initial trust in a system.** Golden-free techniques can prevent future tampering, but, if manufacturing-time attacks are in scope, the system could be compromised before the reference measurement or image is taken. Security architecture is effective for checking authenticity of digital processors and peripherals. Golden-based tamper detection must be used to find other differences between a PCBA's expected and actual construction, such as authenticity checks for analog or simple digital components, copper and substrate verification, and detecting added or removed components on analog signal paths.

Attacks architecture cannot defend As explained in Section 1, security architecture does not protect analog control, simple digital logic, transducers, or similar circuits. These gaps must be filled by online electrical tamper detection if attacks can succeed in the adversary's hands, or by offline electrical or inspection-based countermeasures if it is sufficient to check for tampering before the system enters service.

Security architecture prevents attacks from tampering with data in digital system interactions, but it does not prevent hardware modification *per se*. **If a PCBA attack does not aim to break confidentiality or integrity of digital data, tamper detection may be necessary even on digital systems.** For example, consider destructive attacks: some do tamper with code or digital data,[8] but there are many creative ways to physically destroy a digital PCBA or the system it controls that have nothing to do with code or data. A few examples have been proposed in the literature and will be reviewed in Section 3. Such attacks must be detected via either offline or online tamper detection, depending on a system's threat model.

Finally, tamper detection may assist in preventing some kinds of reverse engineering. Security architecture should be used to prevent adversaries from reading firmware, contents of working memory, inter-IC communications, and other information that would help them understand how a system

[7] For example, [97] is a commercially-available implementation of component authenticity checks for establishing initial trust in a system before imbuing it with secrets or firmware. When the system is first provisioned (perhaps at an untrusted facility), the processor's authenticity is established via PKI, secrets are protected all the way down to the IC die, and security configuration is set atomically with firmware installation. This establishes strong initial trust as long as IC tampering is not in scope, and, as discussed in Section 2.1.3, if it *is* in scope, you're going to need chip-level defenses regardless of how you deal with board-level threats.

[8] Attacks like Stuxnet are not PCBA attacks, but they demonstrate that control of a processor can enable physical damage to a PCBA or a system it controls. Without proper PCBA security architecture, similar attacks can be performed via tampering.

works, but if a system also has sensitive analog functions, online electrical tamper detection ranging from tamper-resistant enclosures, to photon detectors, to trace characteristic monitors are needed. If the goal is to prevent reverse engineering the PCB hardware design, your best bet is a thick layer of hard black epoxy imbued with dense X-ray-blocking material.

Reinforcing against secret extraction To prevent attacks that extract cryptographic keys, sensitive data, plaintext firmware, etc., start with security architecture and augment with tamper detection. Low- to medium-sophistication attacks on digital assets can be effectively foiled by using security architecture: don't let cleartext secrets leave IC trust boundaries, check code and data integrity to prevent adversaries from hijacking a processor to dump secrets, and reduce susceptibility to side channels and fault injection through circuit design, careful firmware implementation, and key usage policies. For the most sophisticated secret extraction attacks, online electrical tamper detection can complement architectural defenses by enabling the system to react to abnormal operating environments. These sophisticated attacks may involve removing a chip from one PCBA and placing it onto a different board that is, e.g., optimized for side channel analysis, or for repeatedly stimulating the chip in a specific way. Or, it could involve decapsulation, polishing backside silicon, or other physical changes in preparation for chip-level tampering. Although some of these attacks blur the line between PCBA and IC attacks, PCBA electrical defenses can nonetheless create hurdles that increase their cost and difficulty.

Defense in depth Of course, it is possible to protect the same attack surfaces with both tamper detection and security architecture. Redundant protections may be desired as insurance against unforeseen gaps in security architecture or implementation errors: Section 3.2.2 explores how such gaps and mistakes can compromise an otherwise secure system.

3 Study of prior PCBA attacks

Having discussed, in abstract, how different attacks and adversaries can be foiled by security architecture or tamper detection, we examine actual and proposed PCBA attacks to show that our theoretical analysis also applies to real problems. We give special emphasis to Bloomberg News' "The Big Hack", video game modchips, and "interdiction" attacks because, as we have already noted, these attacks are commonly cited as motivation for novel tamper detection techniques.

To structure our analysis, Table 1 enumerates classes of vulnerabilities that result from failure to employ security architecture or implement it properly. It also defines a final category that we use for any attack that cannot be prevented with security architecture.

	Failure to...
1	Check integrity or authenticity of stored code/data.
2	Encrypt secrets outside IC trust boundary.
3	Authenticate a peripheral component.
4	Integrity-check data at IC trust boundary.
5	Enforce restrictions on untrustworthy peripherals.
6	No failure – security architecture can't help.

Table 1: Classes of PCBA vulnerabilities caused by security architecture gaps or implementation failures.

This section omits substantial detail due to paper length constraints. We strongly encourage interested readers to review the unabridged manuscript fount at https://arxiv.org/abs/2410.09993.

3.1 "The Big Hack"

"The Big Hack", an article from Bloomberg News [85], is cited as motivation by many recent tamper detection papers [10, 11, 17, 56, 59, 71, 81–84, 88, 102, 114].

Limited technical details about the actual attack. The article alleges that a spy implant on server motherboards compromised high-profile American companies as well as U.S. intelligence agencies. According to the article, the implants were "not much bigger than a grain of rice", they were "gray or off-white in color", they "looked more like signal conditioning couplers... than microchips", and they "varied in size" between victim boards [85]. Regarding their function, the article says the chips contained only a small amount of code and attacked the baseboard management controller (BMC) [85], a highly privileged peripheral with characteristically poor security [31, 54]. The implants' payload allegedly opened backdoors on the BMC and retrieved exploit code from command and control servers [85]. As a concrete example of attacks enabled by such a payload, the article discusses how the implant might change a password [85].

Unfortunately, these details are insufficient to analyze the implant. The only things that the article states definitively are that the implant hijacked control of a processor and that its payload involved the BMC (but not necessarily that it hijacks the BMC). Given these leads, researchers have prototyped plausible attacks. We study these attacks in lieu of the actual "Big Hack" implant.

No secure boot on BMCs. Trammel Hudson proposes that, to hijack control of a BMC, attackers could simply replace or reprogram the flash chip that stores the BMC's firmware [54], as typical BMCs load unencrypted firmware without checking its authenticity or integrity.

However, a reprogrammed flash chip could be detected if a security auditor, like the one who allegedly found the "Big Hack" implant, reads the flash and compares its contents with expected values. To demo a more discreet, nation-state-

level attack, [54] proposes that a series resistor on the SPI data line between the BMC and its flash could be replaced with an active component that monitors sequences of bits read and creates an open circuit at critical moments so that the SPI line's pull-down resistors change affected bits from '1' to '0'. Unprogrammed flash sectors read as long strings of '1' bits, creating a blank canvas for such an implant to inject a software payload. Assuming that the CPU reads at least a handful of such unprogrammed bytes at the end of its firmware, all that remains is to manipulate a branch target in the initial firmware to jump to the payload bytes and the attacker can hijack control of the BMC. An FPGA-based proof of concept demonstrates the viability of this in-flight firmware modification [54].

Regardless of precisely how it manipulates a firmware, this Class 1 vulnerability could be closed by industry-standard boot security. Signature verification prevents firmware modification by PCBA implants, no matter where they are located or how small they are.

Use an unprotected root shell. Hudson notes that a far easier way for a hardware implant to take over a BMC would be to attach the implant to the BMC's serial console header, wait for the BMC to finish booting and print "press enter to activate this console", then emulate an "enter" key stroke and receive a root shell [54]. From here, more emulated keystrokes could direct the device to change passwords and configurations in the same manner as an administrator who has connected their server console to the BMC.

Monta Elkins explores this attack further. He demonstrated that an AT-Tiny microcontroller attached to the UART port on a commercial router could access trigger password resets and change firewall configurations to give an attacker remote access [29, 45].

The root of this problem is that the administrator shell is not protected by a password. This is a sane policy if physical access to a server is deemed a sufficient barrier to defeat adversaries, but if PCBA attacks are in scope, trusting a peripheral simply because it is physically connected constitutes a Class 5 vulnerability.

3.2 Video game console modchips

Many recent papers [11, 47, 48, 56, 59, 78, 80, 81, 84, 112, 113] have cited 'modchips' as motivation for novel PCBA defenses. However, to the best of our knowledge, none have analyzed any *specific* modchip attacks. There were many [19, 30, 53]. Analyzing specific attacks reveals that they were mostly caused by design and implementation mistakes in the first two console generations. *The latest consoles with strong, properly-implemented security architecture have no known modchip vulnerabilities.*

Video game consoles must prevent piracy and cheating [19]. Anti-piracy is important because game creators like to get paid, and because console manufacturers sell hardware below cost to entice customers while recovering profits through game sale royalties. Anti-cheat ensures that games are fun and fair so that customers will keep buying games and subscriptions for competitive online play. Generally, meeting these goals requires that Microsoft maintains control of 1) the Xbox's processor and 2) the foundational secrets in the console's security architecture.

3.2.1 Original Xbox modchips

Failures of the original Xbox's security system were enabled by implementation mistakes and poorly-calculated cost/security tradeoffs. Control of the processor and cryptographic secrets were both lost repeatedly through various mechanisms.

First-stage boot loader mistakes. The original Xbox's first-stage boot loader (1BL) and the key for decrypting its second-stage boot loader (2BL) were *transmitted in the clear across the PCB* from the South Bridge die to the CPU where the 1BL was executed [52, 53]. Cleartext transmission of cryptographic secrets across the PCB is an obvious Class 2 vulnerability, but it was viewed as a good cost/security compromise[9] because Microsoft assumed the relevant buses were too fast to sniff with amateur equipment [99] and that nobody with advanced tools would want to dump the Xbox's boot ROM. This was a costly miscalculation. The Xbox's 1BL, including its 2BL encryption keys, were dumped and published [52, 53]. Then, hackers studied the 1BL and discovered that an implementation error rendered its 2BL integrity check ineffective, a Class 1 vulnerability. The combination of an impotent integrity check and a leaked 2BL encryption key enabled modchips that replaced the 2BL, either by modifying it on the fly as the CPU fetched it or by replacing/overwriting the flash chip [99].

A second failure to implement hashing. Microsoft responded to their console's first compromise by 1) updating the 2BL encryption key and 2) attempting to fix their broken 2BL authenticity check. Both measures were failures.

Updating the 2BL encryption key was ineffective because the Xbox's hardware architecture did not change, so the new key could be dumped in the same manner as the first (or by anybody who managed to read the 1BL). This remains a Class 2 vulnerability.

Fixing 2BL verification *should* have prevented 2BL modification, but Microsoft made an exceptionally poor choice of hashing algorithm. They used the Tiny Encryption Algorithm (TEA),[10] which yields the same digest for multiple inputs if pairs of input bits are manipulated in a specific way [53, 99]. Hackers performed such a manipulation to

[9] Custom CPUs with sufficient ROM for the 1BL would have been more expensive than adding a ROM to the South Bridge [99].

[10] TEA was probably chosen due to the very limited space remaining in the 512B boot ROM [100]. We can speculate that without this cost constraint, a more widely-used and secure hash like SHA1 may have been chosen.

change a 2BL branch target, thereby redirecting the CPU into attacker-controlled flash. This remains a Class 1 vulnerability.

Some hackers did not want to release exploits that depended on Microsoft's 2BL encryption key for fear of violating the Digital Millennium Copyright Act. They found the following other PCBA vulnerabilities that did not need this key.

Failure to verify that an important CPU fault actually happens. The "Visor Vulnerability", came from an incorrect assumption about the CPU's behavior: Microsoft thought a CPU fault would occur if the instruction pointer rolled over from `0xFFFF_FFFF` (the highest address in the 32-bit system) to `0x0`. They relied on this behavior to stop the Xbox's CPU in case of failed 2BL authentication. In fact, a hacker tried it and discovered that execution would happily continue at `0x0` [53, 99]. Thus, by overwriting unprotected flash data or wiring up a modchip, attackers could control the Xbox's CPU, dump the 1BL, and recover the 2BL encryption keys [53]. The Visor Vulnerability is Class 1 because it was an implementation error that broke stored program verification.

"Jam code" interpreter vulnerabilities. Another attack, the "MIST premature unmapping attack", was enabled by overwriting or interposing on the same flash data as the Visor Vulnerability.

The Xbox 1BL needed to stuff several complex functions, including RAM initialization, 2BL decryption and integrity checks, and more, into a mere 512 bytes. To help save space, several of these operations were performed by a virtual machine in the 1BL that executed a sequence of "jam codes" stored in flash [99].

The jam codes were stored unencrypted and the 1BL did not authenticate them. To mitigate this obvious Class 1 vulnerability, the Xbox's engineers tried to make the 1BL virtual machine incapable of doing bad things by blacklisting jam code byte patterns. However, their blacklist did not account for the flash chip's address aliasing and attackers exploited this oversight to more easily dump the 1BL [99]. Attackers could also sidestep the blacklists by assembling malicious commands byte-by-byte [99]; this was exploited to disable the secret ROM while the 1BL was running, causing execution to continue in attacker-controlled flash. The consequences for the Xbox were lost secrets and loss of control of the CPU [53, 99].

Vulnerabilities from a forgotten legacy behavior. Yet another attack leveraged a legacy CPU behavior that Microsoft's engineers had overlooked. In brief: grounding the `A20#` CPU pin ("the A20 gate") caused the 20th bit of whatever address was requested by the CPU to be set to zero. This was useful because grounding the pin changed the Xbox's boot vector to attacker-controlled flash instead of its secret ROM [99, 100]. Once more, overwriting flash or adding a modchip gave attackers control of the system. This "A20 Bug" was also useful for dumping the 1BL.

The A20 Bug allowed attackers to sidestep stored program verification. It is yet another Class 1 vulnerability.

3.2.2 Xbox 360 modchips

The Xbox 360's security architecture was greatly improved compared to the original Xbox, but software vulnerabilities, ineffective glitching countermeasures, timing side-channels, and various architectural and implementation mistakes related to the disc drive enabled PCBA attacks against the console. However, *the final Xbox 360 motherboard revisions, which corrected many implementation and architectural failures, were not vulnerable to any PCBA attacks.*

JTAG/SMC: a physical method for exploiting a hypervisor vulnerability. It is ironic that many video gamers equate the term "JTAG" with "hardware modchip" because JTAG was only a convenient way to deliver a software exploit: "a new way to exploit the well-known 4532 kernel" [38]. When the software vulnerability was closed, JTAG modchips no longer worked. The exploit behind JTAG'd consoles was known as the "King Kong" attack because the original delivery mode was to make the King Kong video game load a malicious saved game file. A hardware attack was devised because attackers felt that loading the King Kong game every time they wanted to launch Homebrew was too *inconvenient*.

We encourage readers to review the extended version of this paper for a fascinating technical explanation of the JTAG/SMC attack. In brief, a combination of unprotected System Management Controller (SMC) firmware in flash (a Class 1 vulnerability) and active JTAG on the GPU (a Class 5 vulnerability, in this context) were exploited to load the King Kong attack software payload after the kernel had booted.

Exploiting timing side channels to guess an HMAC digest. Another physical attack on Xbox 360, the "timing attack", bypassed fuse checks to downgrade consoles to a King-Kong-vulnerable hypervisor. The root cause of the vulnerability was a `memcmp` function that worked byte-by-byte, returning an error as soon as a difference was found. Minimum fuse values were protected by an HMAC, but the non-constant-time memory comparison allowed attackers to guess the correct hash byte-by-byte using the time before HMAC comparison failure to indicate whether their byte guess was correct. This implementation error amounts to a failure to correctly verify data stored off-chip: a Class 1 vulnerability.

A microcontroller could be attached to the PCBA to repeatedly 1) reflash NAND with a new base kernel and header that contained a hash guess, 2) reset the CPU, 3) measure how long it took for hash comparison to fail, and 4) repeat the procedure with a new guess. Codes output on the console's POST out pins, which changed at known points in its boot process, were used as timing references [38]. This procedure could bypass fuse checks within a day [1, 23, 38].

Glitch attacks. The third type of compromise affecting the 360 was fault injection. These so-called Reset Glitch Hack (RGH) attacks used FPGA modchips to inject faults that skipped over buffer comparison failures in the 360's 2BL. At the moment when hashes were being compared to deter-

mine whether software was authentic and should be booted, the modchip pulsed the reset signal. This caused the chip's reset procedure to begin, but the pulse was so short that the procedure did not complete and the CPU continued executing [3]. However, the glitch caused the memory comparison result to read as zero, indicating that the comparison had found no differences even though the 4BL was tampered. Boot proceeded into attacker code. The RGH attacks are Class 1: they exploit implementation errors that undermine secure boot.

Different console motherboard revisions required different variants of the RGH. In all variants, the first step was to slow the 360's CPU because the reset pulse had to be sent at *exactly* the right moment and the Xbox's native speed was too fast for modchips to glitch accurately. For early console revisions, this was accomplished by asserting the CPU's `PLL_BYPASS` signal [3, 39], which was exposed on the PCB, causing the CPU to execute at the frequency of its external oscillator. This bypass signal was removed in later console revisions, but hackers discovered that the HANA peripheral, which controlled the CPU clock frequency, would reduce the frequency in response to I2C commands [3, 39].

Next, the reset pulse was sent. The reference point for determining when the pulse should be sent was, as in the HMAC timing attack, the CPU's POST output pins [3].

Microsoft tried a variety of mitigations, including removing POST debug output traces to remove the RGH timing reference and integrating the HANA chip into the South Bridge, but attackers found ways around these obstacles [3]. In the final Xbox 360 motherboard revision, Microsoft finally killed RGH by implementing random delays and redundant checks in their 2BL [3], as well as improved reset line filtering on the CPU die. POST signals were disabled (instead of merely removing their PCB traces). This final 360 hardware revision is widely considered secure against physical attacks. Interested readers may refer to [3] for more details on RGH.

An insufficiently-secure security-critical peripheral. When the CPU fetched anti-piracy data from the Xbox 360's disc drive, the exchange was encrypted with a console-unique key. This attempt to secure communications with the disc reader was undermined by Class 1 and Class 2 vulnerabilities in the disc drive controller.

The disc drives on early Xbox 360 generations were trivially broken. One early disc reader stored its drive key unencrypted in a discrete flash chip on the drive PCBA [1, 23], a Class 2 vulnerability. The drive controller also did not use cryptographic firmware verification [1, 23], a Class 1 vulnerability. Attackers dumped the key, programmed a drive controller with a firmware that enabled piracy, and re-paired the modified drive firmware to their CPU using the dumped key. The next hardware revision's disc reader *did* encrypt and integrity-check its firmware and drive key, but it had a manufacturing mode [1] which could be entered by presenting a special boot disk or shorting a few of the drive's pins together. Once in manufacturing mode, the drive could be reprogrammed. The manufacturing mode opens another Class 1 vulnerability.

A series of hardware revisions followed, where each version was compromised by glaring security architecture and implementation failures. We strongly encourage readers to read the extended version of this paper, which elaborates on the escalating cat-and-mouse game between Microsoft and Xbox hackers. In brief, some drives would spit out their keys over UART or SATA if their trays were forced half-open while power was disconnected [2] (Class 2); drive controllers were repeatedly compromised because they did not cryptographically verify their firmware (Class 1); Microsoft thought they got the better of attackers by bonding the write protect pin on a flash die to ground inside the disc drive controller package, but this was defeated when adversaries figured out they could cut the bond wire *by drilling into the drive controller package (!)* [2, 19]. In the end, Microsoft finally released a drive revision that cryptographically checked its firmware and did not have a silly key dump feature, paired with a CPU that would not cough up its drive key in response to RGH attacks [2].

3.2.3 There weren't any Xbox One modchips

In Microsoft's latest consoles, strong digital security architecture and careful implementation appear to have succeeded against PCBA tampering adversaries [19].

Distrust everything outside a trusted IC. A system on chip (SoC) was the console's root of trust and signals from every other component on the board were treated with suspicion [19]. The only other security-critical IC, the disc reader, used proper secure boot [19] and incorporated all of the implementation lessons that Microsoft and its vendors learned from disc drive attacks on the Xbox 360.

Dedicated security coprocessor. One of the security IPs in the Xbox One's SoC was a security coprocessor that operated on all of the console's secrets and did not permit any other IP in the SoC to read them under any circumstances [19]. This corrected a major flaw from the Xbox 360: the CPU could read important secrets, enabling any adversary who managed to hijack the CPU to learn them. Besides isolating secrets from the CPU, the security core had dedicated communication channels with the CPU and RAM encryption engine that prevented other IP blocks from interacting with the security engine in unintended ways (i.e., communications with the core would never appear on common buses).

Hypervisor with stronger VM isolation. Like the 360, the Xbox One uses a hypervisor to check code authenticity and integrity before marking any page executable. However, the Xbox One hypervisor runs games, system functions, and hardware drivers in separate virtual machines [19] whose memory is encrypted using different keys [19, 69].[11] To achieve this, the security coprocessor loads a different key into the RAM

[11] This architecture is now deployed in server-class CPUs to strengthen isolation between tenant virtual machines (VMs) [4, 69].

encryption engine for each VM in the system [4]. Per-VM encryption prevents software vulnerabilities in one VM from affecting any of the others: any values written to DRAM under one VM's key decrypt as garbage when read by a different VM. In particular, this design prevents game or OS vulnerabilities from coaxing the hypervisor into using unencrypted memory as the Xbox 360 King Kong attack did.

3.3 The NSA ANT Catalog

Hardware modification of IT equipment in transit ("interdiction") is another common motivation for recent work. Examples of citing papers include, but are not limited to: [11, 15, 47, 56, 71, 113].

The NSA ANT Catalog is a leaked U.S. government document that describes several hardware implants available to U.S. intelligence [7]. According to [46], some were installed by interdiction. The NSA Playset [35, 37, 79, 96] is an effort by hardware hackers to make open-source versions of the ANT Catalog attacks. Others have also studied the technical feasibility of these implants [91, 92, 98, 105].

Even the nation-state spy implants described in the ANT Catalog can, with a few exceptions, be mitigated by security architecture or implementation fixes. As the NSA Playset's authors put it, these implants are the result of "design flaws" exploitable by "12-year olds" [37].

Abuse of active debug infrastructure. One NSA implant attaches to a server motherboard's debug infrastructure to install malware on the CPU. Details of the malware are irrelevant; active debug interfaces are widely understood to be a vulnerability if physical attacks are in scope [86, 98]. The NSA Playset has demonstrated that a wide variety of software payloads can be installed using the same JTAG Trojan: they have changed file permissions in Linux [36], bypassed Linux login password checks [35], and manipulated outputs of an industrial PLC [35].

Most microprocessors are capable of disabling JTAG infrastructure permanently via eFuse or write-protected flash configuration bits. JTAG is helpful for firmware development, but it should always be disabled in deployed systems to prevent easy, catastrophic attacks. Failure to do so could lead to many classes of vulnerabilities, depending on precisely how the active debug infrastructure is abused.

Protection against malicious peripherals. The Catalog describes several implants installed in USB cables or host ports that inject software exploits and include a radio transceiver to bridge air gaps. System protections cannot prevent transceivers being installed to bridge airgaps, but software and USB driver vulnerabilities are garden-variety Class 5 implementation errors [51].

Another implant replaces hard drive firmware. The Catalog gives the impression that the implant exploited vulnerabilities in master boot record (MBR) parsing. If this is true, the underlying issue is the software vulnerability, not the hardware delivery mode. However, the more important physical vulnerability is an untrustworthy peripheral: if a system component is unauthenticated, all bets are off. [110] illustrates how a hard disk without protected boot can modify written data or create a backdoor for exfiltrating information. Firmware signing is listed in the paper as an effective countermeasure [110]. As the authors point out, firmware signing does not prevent complete replacement of the hard drive, but authenticating the peripheral IC closes this gap. Yet another attack leverages a malicious hard drive firmware to turn the hard drive into a microphone: the malicious firmware keeps track of position error measurements of the disc reading head, which is highly sensitive to vibrations such as those caused by nearby speech, and using them to reconstruct an audio signal [64]. This is cool, but we reiterate: if PCB attacks are in-scope, system architects should authenticate peripheral ICs to mitigate Class 5 vulnerabilities and should be very careful incorporating any peripherals that do not check signatures on their firmware, i.e., that have Class 1 vulnerabilities.

Snooping and spoofing ethernet traffic. One NSA implant lives in ethernet jacks where it snoops traffic and injects packets onto a target network. It is coupled with an RF transceiver for remote control and crossing airgaps. The NCC Group [26] prototyped a similar attack.

An implant in an ethernet jack 'taps the wire' in the most traditional sense: a processor on the PCBA is the cryptographer's "Alice", a remote computer is "Bob", and the implant is "Eve" if it is passive or "Mallory" if it injects packets. These implants are precisely the adversaries that cryptographic protocols were designed to protect against. Any connection using industry-standard security (e.g., TLS) closes the Class 2 vulnerabilities associated with lost secrets, the Class 3 vulnerabilities associated with failing to authenticate a remote computer, and the Class 4 vulnerabilities from not integrity-protecting communications.

DMA attacks by PCI(e) devices. A fifth NSA implant was installed "plug-and-play" style into a PCI adapter to bridge airgaps and install malware on the host. The NSA Playset prototyped this capability with an off-the-shelf PCI development kit [37] and demonstrated a password check bypass on then-modern Macs. The password bypass modifies code in RAM so that the password verification function accepts any non-empty password as correct [34]. Later, PCILEECH [40] achieved similar payloads with higher read and write speeds. [40] is still maintained as a penetration testing tool.

As explained in [14, 50], older computers typically imposed no limits on memory accessible to DMA peripherals, so DMA devices could read or modify critical code or data in RAM. The Windows 9x, 2000, 20003, XP, or Vista systems that this NSA implant targeted almost certainly fell into this category. These attacks are partially mitigated by enabling an I/O memory management unit (IOMMU) [40], a hardware block built into most modern processors [8, 73] that restricts which memory is available to different I/O devices. IOMMUs

prevent Class 5 "plug-and-play" attacks, but, similar to other blacklisting-based mitigations, they can be bypassed by PCBA attacks located after the IOMMU. The real vulnerability at the root of DMA attacks are in Class 4: sensitive data written to working memory by the CPU are neither encrypted nor integrity-checked. Though we do not know of any examples, it is also easy to imagine Class 2 vulnerabilities from writing unprotected secrets to RAM that are exploited via DMA.

Retroreflectors. The Catalog includes several "retroreflectors", which use a signal of interest to modulate radar waves broadcasted by an attacker who monitors the reflected waves from a distance. The modulations in the reflected waves allow recovery of the signal of interest without high-power transmitters [79, 105]. ANT Catalog implants can broadcast digital data, analog video, ambient audio, or simply act as a location beacon. Green Bay Professional Packet Radio has a series of YouTube videos that analyzes and demonstrates their principles of operation [44].

The NSA Playset has prototyped retroreflectors that work with commercial software-defined radios. Their PoCs can monitor PS/2 keyboards, low- to mid-speed USB, and, to a limited extent, VGA [79]. A few years later, [105] took a second look at the attack and affirmed retroreflector viability at short range with a PoC attack on a USB keyboard. Both used a rudimentary setup and simple signal recovery; more sophisticated analysis could increase range and accuracy.

Systems defenses cannot prevent devices that broadcast ambient audio or act as a beacon; these devices leech power from a host system but do not otherwise interact with it. These are our first Class 6 attacks. The retroreflectors that broadcast digital data are Class 2 problems that can be mitigated by encrypting signals between digital components.[12] The retroreflector that broadcasts analog video is more difficult to address because there are no accepted solutions for analog signal encryption; this vulnerability is Class 6.[13]

3.4 Attacks on TPMs

Trusted platform modules (TPMs) are dedicated chips[14] for storing secrets, making digital signatures, generating entropy, and other security-critical operations. Sometimes TPMs are used just to isolate these operations from potentially vulnerable software on a CPU. However, another usage model, 'remote attestation', relies on TPMs as trustworthy reporters of system integrity in untrusted systems. We refer readers to the extended of this paper for an explanation of TPM attestation.

The TPM threat model centers on keeping secrets: endorsement keys must be protected so that attestation is meaningful, and keys entrusted to the TPM by a system must also be protected. Additionally, it must not be possible to make a TPM tell a remote verifier that a system is running 'safe' software when it is not, and, if a processor relies on a TPM for strong entropy, received entropy must not be compromised.

Researchers have prototyped several PCBA attacks against TPMs: 1) an attack on TPM-supplied entropy. 2) a snooping attack that recovers cryptographic keys stored in TPMs, and 3) two attacks on remote attestation.

Implant-supplied randomness. In platforms that rely on a TPM to generate cryptographically secure random numbers, a hardware interposer could respond to CPU requests for random numbers with known values if CPU↔TPM communications are not integrity-protected, thereby undermining cryptographic protocols that require strong entropy [15]. If PCBA security is part of a system's threat model, this is a Class 4 security architecture failure.

Snooping cleartext secrets off a bus. In the default configuration, most TPMs do not encrypt traffic between the TPM and CPU, allowing adversaries with inexpensive equipment to sniff keys off the bus. [6] describes how this attack could retrieve disk encryption keys.

If TPMs are only intended to protect secrets from CPU software vulnerabilities, transmitting keys in the clear between the CPU and TPM is acceptable. But if the TPM is supposed to, e.g., protect disk encryption keys in the event that a laptop is stolen, then transmitting secrets in cleartext across the PCBA is a Class 2 security architecture gap.[15]

Assumptions that do not hold for PCB adveraries. Remote attestation relies on a handful of assumptions, one being that TPM initialization is always performed by a piece of trusted code that runs immediately after a CPU is reset [101]. In other words, it is assumed that the TPM and CPU will always be reset together. A physical adversary can violate this assumption by electrically isolating the TPM's reset signal from the CPU's. This allows the attacker to reset the TPM and re-initialize it with forged measurements to deceive remote verifiers [60]. This is known as a "TPM reset attack".

[107] discusses how, with the help of a PCBA implant, TPM reset attacks can also be applied in the reverse direction: if a CPU is reset but its TPM is not and the CPU's early boot stages are prevented from communicating with the TPM, the TPM will still contain software measurements from the previous boot, enabling untrusted software to deceive a remote verifier into trusting a device based on stale measurements. The PCBA implant prevents communication with the TPM by unconditionally asserting a signal on the CPU ↔ TPM bus.

[12] These attacks are Class 3 *as proposed by the NSA Playset and Catalog*, but keyboards, mice, and other digital or analog sensors generally create Class 6 vulnerabilities. For example, encryption between a keyboard and host can rule out trojans in USB cables, but the keys themselves are simple pushbuttons that cannot be protected by system security.

[13] Note that modern digital video signals are typically encrypted, e.g., using HDCP on HDMI [62]. A similar attack on *digital* video would fall into Class 2.

[14] Some SoCs incorporate TPM logic on-chip. There are also "soft" TPMs. But our discussion is only concerned with discrete TPMs.

[15] TPM v2 standards offer optional safeguards against PCBA tampering by encrypting and authenticating the channel between a CPU and TPM [15], but these features are optional and, according to [15], were almost universally absent from TPM drivers as of 2018.

At first glance, reset attacks look like they cannot be countered by digital architecture: system security primitives cannot prevent two discrete chips from being reset separately. However, closer examination reveals that separate resets are a distraction rather than the main challenge. The real problem is in Class 3: TPMs assume they are initialized by a trusted device but they do not authenticate that device. A more robust solution could leverage PKI to enable TPMs to verify they are being initialized by an authorized device, e.g., a genuine CPU from a trusted manufacturer.

Dynamic root of trust measurement (D-RTM), which is discussed next, was once thought to be a solution to reset attacks [60]. However, the remainder of this section explains that D-RTM is also vulnerable to physical adversaries.

Lack of integrity protection enabling packet-in-packet attacks on D-RTM. There are two flavors of TPM-based attestation. So far, our discussion has focused on *static* TPM attestation. Static attestation has the drawback that verifiers must maintain lists of attestation values associated with software and hardware configurations. Such a list quickly becomes unwieldy because a typical PC loads many pieces of device-specific software during boot and there may be multiple authentic versions of each boot loader stage (e.g., due to updates). *Dynamic* attestation,[16] or D-RTM, aims to solve this problem by allowing untrusted devices to transition to a trusted state via a short chain of hashes that is common across many platforms.

A D-RTM launch begins when control of the CPU is transferred to trusted code via a special CPU instruction. This code sets up an environment that is protected from untrusted program snooping or interference [103]. Sensitive data is then processed only within this secure environment.

The D-RTM launch sequence gets recorded with a special set of platform configuration registers (PCRs). Restrictions on writing D-RTM PCRs are hardware-enforced: on Intel chipsets, for example, the South Bridge drops any writes to the D-RTM PCRs that do not originate from the D-RTM launch code [106]. D-RTM's security rests on the inability of 'normal' software, including privileged software running on the CPU, to write these special PCRs [103, 106, 107].

Unfortunately, [107] demonstrates that a hardware implant can transform a seemingly innocuous bus transaction into a D-RTM initialization sequence after hardware blacklisting has already passed. The authors demonstrate attacks against both LPC bus TPMs (using the `LFRAME` signal) and I2C bus TPMs (by manipulating the I2C clock). These attacks mount a packet-in-packet attack: they build an LPC or I2C transaction where the first half is a trigger sequence that the hardware implant will recognize as a cue to break the malicious packet by manipulating bus signals, and the second half is a D-RTM register extension command.

These D-RTM attacks are enabled by a straightforward Class 4 vulnerability: communications between the CPU and TPM must be integrity-checked to prevent a man-in-the-middle from mangling messages. Also, as with the Xbox jam tables, we see that blacklists are not a good security strategy.

[16] It is called "dynamic" measurement because it occurs on a running system without requiring a system reset [103].

3.5 More attacks

This section reviews PCBA implant research that is not part of a 'family' – these attacks don't target a common system (e.g., TPMs, game consoles, etc.), and they were not perpetrated by a single party like the ANT Catalog.

General purpose computers not designed for PCBA security. [26] crams a single-board computer with a 3G modem into a hollow nook in a laptop dock and solders in circuits to snoop or modify signals from dock peripherals. It can grab or inject keystrokes on the USB bus (Class 2), record video frames from a USB webcam (Class 2) or an analog monitor (Class 6), snoop or inject ethernet traffic (Class 2/4), snoop general USB traffic (Class 2), and record audio [26] (Class 6). Although there's not much that can be done for analog audio streams or video signals, the rest are solvable with systems approaches. The issues of keystrokes and ethernet taps were discussed in Section 3.3. Digital video and webcams can use standard encryption between ICs to mitigate tampering.

Voting machines from the early 2000's were not designed to withstand physical attacks. [32, 43] both found that leading voting machine manufacturers had no boot security in place whatsoever (Class 1). [32] shows that, in their machine, the boot address mapped to one of three memory chips on the board, any of which could be reprogrammed or replaced by an adversary. For example, [43] replaces voting machine firmware on one machine with a chess program. Def Con teardowns of other voting machines have found that similar attacks are possible [13]. Both papers suggest cryptographic boot verification as a mitigation against tampered flash chips [32, 43].

Insecure communication with a critical peripheral. [33] reverse engineers a fingerprint smart card and shows that, by interposing between the fingerprint sensor and the CPU, it is possible to replay old fingerprints, "brute force the matcher" by injecting many images of fingerprints, and more. [20] does the same on real-world smartphones. This was all possible because there signals between the sensor and CPU were unencrypted and not integrity protected, Class 2 and Class 4 vulnerabilities. The researchers state that their PCBA attacks all boil down to a failure to use standard security architecture. Note that it is also important to verify the authenticity of critical peripherals like fingerprint sensors to prevent Class 3 vulnerabilities. The attackers in these instances simply didn't need to get that creative.

[95] demonstrates two attacks that can be launched by a "Trojan" smartphone screen: 1) the screen exploits vulnerable device drivers to hijack control flow of the main CPU, and 2) it logs a user's touches and sends false touch events to the

CPU. The screen is simply an I2C peripheral: its hardware provenance and integrity should be established with PKI to prevent Class 3 vulnerabilities, and its firmware should be authenticated to prevent Class 1.[17]

Voltage glitching in embedded systems. Chip.fail [87] is an FPGA development board connected to a 3-channel switch that enables experimenting with voltage glitching. The device has successfully attacked several common IoT microcontrollers, including some with brownout reset (BOR) that was advertised as a glitching countermeasure, and showed that naive comparisons in firmware could be bypassed with voltage glitching. They also achieved flash option byte downgrade on an STMF2 microcontroller, which re-enables the chip's debug infrastructure. Section 3.3 has already discussed the consequences of active JTAG in deployed systems.

Insofar as glitching is an implementation error that undermines stored program verification, it is a Class 1 vulnerability. As discussed in Section 3.2.2, random delays and removal of timing reference signals make it near impossible for attackers to create stable attacks. A specific mitigation for the flash option byte downgrade in STM microcontrollers is to check status registers shortly after boot and ensure that they read as expected before reading any sensitive values from RAM [77].

Inducing crosstalk by moving PCB traces. [42] proposes that the location or dimensions of PCB traces could be modified to enable signal changes on one trace to impact a neighbor. A simulation is conducted to demonstrate that doubling a trace's width and introducing a long parallel section in two traces results in a few volts of crosstalk. The authors do not present a specific attack PoC, so we reason about how the attack might interact with a full system. If the victim trace is responsible for data transmission, this is a Class 4 vulnerability that can be mitigated by checksums. If the victim is a different kind of trace – an input that triggers a processor interrupt (Class 6), a discrete logic input or output (Class 6), or an analog control signal (Class 6), then systems security has no ready answer.

Adding a covert channel to wireless communications by moving PCB traces. [24] proposes that various hardware modifications could be used to influence a radio transmitter to add a covert "polyglot" transmission to a legitimate radio signal. The Trojan influences the RF frontend's oscillator frequency using crosstalk to add an extra modulation on top of a legitimate signal. The legitimate signal can still be demodulated. Digital security primitives cannot prevent modulations to an oscillator frequency – this vulnerability is Class 6.

Packet-in-packet attacks against TTE. [67] describes a more specific attack involving crosstalk that enables a Trojan low-priority device in a time-triggered ethernet (TTE) network to disrupt the network's high-priority traffic. The resulting loss of synchronization, the authors say, is sufficient to cause unrecoverable errors in spacecraft, aircraft, automobiles, industrial control systems, and other real-time systems.

TTE enables mixed-criticality communication, meaning it sends low-priority and high-priority messages on the same physical infrastructure while guaranteeing that high-priority messages cannot be impacted by low-priority ones. In particular, special "protocol control frames (PCFs)" may only be sent by designated high-priority devices because these are essential for maintaining synchronization in the network. If a low-priority device tries to send PCF, the TTE switches drop the packet [67].

However, [67] found that it was possible to modify low-priority traffic when it was on an outbound port (*after* it had already been accepted by the TTE switch). This is accomplished by conducting a high voltage pulse (on the order of kilovolts) over a low-priority ethernet cable into a port on the TTE switch. The electrical design of ethernet ports, which includes galvanic isolation, ensures that the victim port will not be destroyed by the high voltage pulse, but the pulse will cause significant EMI that impacts adjacent switch ports. If an apparently-benign low-priority packet was being transmitted out of the switch at the same instant that a high voltage pulse induces EMI on that outbound port, a connection reset event may occur on the line, enabling a packet-in-packet attack. The 'inner packet' could be a valid PCF, and the authors demonstrate that a PCF generated in this manner has a good chance at causing lack of synchronization and unsafe delays in realistic TTE environments [67].

The authors suggest various solutions to this vulnerability. The fundamental flaws are Class 2/4, and link-layer encryption for high-priority traffic, is among the authors' suggested mitigations [67]. Note that this attack is another failed case study of blacklisting unwanted behavior as a security strategy.

Tampering analog control circuitry [42] adds a resistor, capacitor, and PMOS to an op-amp circuit. These modifications have the effect of grounding a microcontroller input after the circuit has been active for a while. This was done in a fan controller circuit and the grounded pin deactivated the fan. Systems security has little to say here. This is a Class 6 vulnerability.

Destructive high voltage. It is possible to design a Trojan component that deliberately destroys components on a PCBA. For example, [104] is a USB device that applies large positive and negative voltage to a host's USB ports. It is marketed as a tool for pentesters. A few hardware design techniques can provide robustness against extreme voltages, but systems security has little to offer. This is Class 6.

Trace breakage due to electromigration. [42, 70] discuss attacks where a malicious fabricator thins PCB traces. [42] suggests that this could cause systems to overheat by increasing trace resistance. [70] proposes such an attack could induce electromigration at a (somewhat) predictable point in the future. Security architecture cannot prevent a too-thin trace from eventually separating from high current density: Class 6.

[17] Ironically, smartphone makers have tried to authenticate their components using PKI but right-to-repair legislation prevents them from securing consumers against malicious replacement parts.

4 Conclusions and future directions

We explained how security architecture can mitigate PCBA tampering and why its assumptions are reasonable. Then, we considered how it complements, and overlaps with, tamper detection. Finally, we examined over fifty real-world PCBA attacks to show that the PCBA attacks most commonly-cited as motivation by recent tamper detection research can be mitigated by basic security architecture or by fixing implementation errors. Our discussion of security architecture's role in PCBA tamper detection fills a gap in prior work on this topic.

Most of the attacks we reviewed affected systems that either did not implement a PCBA security architecture, or implemented one that is obviously inadequate if board-level attacks were part of the system's threat model. This set includes the BMC attacked in Bloomberg News' "The Big Hack", the original Xbox, TPMs, computers targeted by NSA ANT attacks, voting machines, cell phone screens, fingerprint sensors, and others. **Except for voltage glitching in embedded bootloaders, the Xbox and Xbox 360 were the only systems we studied that tried, but failed, to defend against PCBA attacks with security architecture.** The Xbox's vulnerabilities came from ruthlessly-exploited security architecture oversights and from implementation mistakes. Crucially, **improved implementation and architecture in later Xbox 360s and the Xbox One seem to have mitigated tampering**, as there are no known tampering attacks against these consoles [19].

This raises an important question: *why have so few systems used security architecture to protect themselves from PCBA attacks?* We have no special insight into decisions made by most of the companies whose systems were compromised in our survey, but the Trusted Computing Group (TCG) tells us clearly that security against PCBA tampering was not an objective of the v1 TPM protocol [65, 101]. In light of this, it is no surprise that PCBA attacks on TPMs could undermine randomness, read keys off buses, and more. We speculate that, similar to TPMs, **most of the systems we studied were not designed to withstand PCBA attacks.** PCBA security is expensive to design and implement; if defending against PCBA attacks is not a business requirement, why incur this expense?

Attacks on systems that did not even bother to implement basic security architecture should not be cited as motivation for developing new countermeasures. These attacks demonstrate that PCBA defenses *are needed*, not that existing solutions are inadequate. Although a few attacks have been proposed that security architecture could not have helped, for the majority of attacks seen to-date, if only one defense can be deployed, security architecture is the correct choice. And this is a happy outcome: as we explained in Section 2.2.3, PCBA security architecture has become increasingly accessible because it uses the same cryptographic primitives as software defenses, which are in high demand. Today, commercial microcontrollers advertise side channel-hardened encryption engines, manufacturer-attested cryptographic identities, and secure provisioning solutions. In contrast, when Microsoft designed the Xbox 360, it was eventually forced to co-designed a secure disc drive controller with its suppliers after years of exploits enabled by COTS parts with inadequate security.

Accurately understanding the problem that motivates new work is not just a pedantic concern. It affects design choices and evaluation methods for new research. An important question for future work is how precisely a tamper detection approach must be tailored to particular kinds of PCBA attacks. For example: does a technique designed to enable a processor to detect if it has been moved to a different PCBA as part of a secret extraction attempt need different sensing methods and resolution than one designed to detect probing an analog signal? We expect this question to be answered in the affirmative: for example, a periodic probe signal designed to characterize trace impedance may disrupt an analog control circuit more than a digital processor, so different methods may be required to defend analog vs. digital circuitry.

Looking ahead, practitioners should mitigate PCBA tampering via security architecture based on new commercially-available parts with the necessary security features, and via careful implementation based on lessons learned from past systems' mistakes. Meanwhile, researchers should focus on attacks that cannot be defended by security architecture and on the specifics of use cases where tamper detection complements architecture.

Additionally, researchers should focus on the problems highlighted in Section 2.2.2: tamper detection research would benefit from physical measurements of attacks, countermeasure performance, and the effects of process/environmental uncertainty. Several promising PCBA tamper defense concepts have been proposed, but we know too little about real-world attacks to assess how effective these approaches will be in practice. Before we can assess the likelihood that, e.g., an optical inspection pipeline will detect an attack, we must learn how similar a "Trojan" chip's pixel intensity histogram could be, when imaged under a microscope, compared with a legitimate chip. Or: before we can say whether electrical sensors are capable of detecting a probe wire on a bus, we must understand the effects of different probes and probing methods on, e.g., impedance and signal reflections. We need to prove to ourselves that, when reasonable amounts of noise and process variation are taken into account, it is not possible for an attacker to implement a malicious chip that looks exactly like an authentic one under a camera or X-ray, or that it is not possible to mask signal reflections from probing via precise termination or other counter-effects. If such subtle attacks are possible, we need to be able to reason about their cost and difficulty so that we are better informed of the "security return on investment" of different approaches. Until we can more rigorously measure attacks and countermeasures, tamper detection implementors will need significant research and engineering to validate their implementation's security.

References

[1] 15432. Protecting and hacking the Xbox 360 (part 1). discourse.world, March 2020.

[2] 15432. Protecting and hacking the Xbox 360 (part 2). discourse.world, April 2020.

[3] 15432. Protecting and hacking the Xbox 360 (part 3). discourse.world, May 2020.

[4] AMD. AMD secure encrypted virtualization (SEV). Developer Central.

[5] Ross Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley, Newark, New Jersey, 2020.

[6] Denis Andzakovic. Extracting Bitlocker keys from a TPM. Pulse Security, March 2019.

[7] Jacob Appelbaum, Judith Horchert, Ole Reissmann, Marcel Rosenbach, Jörg Schindler, and Christian Stöcker. NSA's secret toolbox: Unit offers spy gadgets for every need. *Der Spiegel*, December 2013.

[8] Apple. Direct memory access protections for Mac computers. Apple Platform Security, 2021.

[9] Thomas D. Bergman, Cyber Program Manager, and Katie T. Liszewski. Battelle barricade: A nondestructive electronic component authentication and counterfeit detection technology. In *2016 IEEE THS*, pages 1–6, 2016.

[10] Aritra Bhattacharyay, Prabuddha Chakraborty, Jonathan Cruz, and Swarup Bhunia. VIPR-PCB: A machine learning based golden-free PCB assurance framework. In *ACM/IEEE DAC*, pages 793–798, 2022.

[11] Aritra Bhattacharyay, Shuo Yang, Jonathan Cruz, Prabuddha Chakraborty, Swarup Bhunia, and Tamzidul Hoque. An Automated Framework for Board-level Trojan Benchmarking. *IEEE TCAD*, 2022.

[12] Swarup Bhunia and Mark Tehranipoor. Editorial for the Introductory Issue of the Journal of Hardware and Systems Security (HaSS). *Journal of Hardware and Systems Security*, pages 1–2, March 2017.

[13] Matt Blaze, Jake Braun, Harri Hursti, Joseph Lorenzo Hall, Margaret MacAlpine, and Jeff Moss. Report on Cyber Vulnerabilities in U.S. Election Equipment, Databases, and Infrastructure. Def Con 25 Voting Machine Hacking Village, 2017.

[14] Adam Boileau. Hit by a bus: physical access attacks with Firewire. In *Ruxcon*, 2006.

[15] Jeremy Boone. TPM Genie: Interposer Attacks Against the Trusted Platform Module Serial Bus, 2018.

[16] Ulbert J. Botero, Fatemeh Ganji, Damon L. Woodard, and Domenic Forte. Automated trace and copper plane extraction of x-ray tomography imaged pcbs. In *IEEE PAINE*, pages 1–8, 2021.

[17] Ulbert J. Botero, Fatemeh Ganji, Damon L. Woodard, and Domenic Forte. Automated Trace and Copper Plane Extraction of X-ray Tomography Imaged PCBs. In *IEEE PAINE*, pages 1–8, November 2021.

[18] Sergey Bratus and Anna Shubina. Overlooked foundations: Exploits as experiments and constructive proofs in the science-of-security. In *USENIX CSET*, 2017.

[19] Tony Chen. Guarding Against Physical Attacks: The Xbox One Story. Platform Security Summit, 2019.

[20] Yu Chen and Yiling He. Bruteprint: Expose smartphone fingerprint authentication to brute-force attack. arXiv 2305.10791, 2023.

[21] Deruo Cheng, Jingyang Dai, Yee-Yang Tee, Yiqiong Shi, and Bah-Hwee Gwee. PCB surface component detection with computer vision assisted label generation. In *IEEE IPFA*, 2024.

[22] William E. Cobb, Eric D. Laspe, Rusty O. Baldwin, Michael A. Temple, and Yong C. Kim. Intrinsic physical-layer authentication of integrated circuits. *IEEE TIFS*, 7(1):14–24, 2012.

[23] Rodrigo Copetti. Xbox 360 architecture - a practical analysis, 2022.

[24] Emmanuel Cottais, Jose Lopes Esteves, and Chaouki Kasmi. Second order soft-TEMPEST in RF front-ends: Design and detection of polyglot modulations. In *EMC EUROPE*, pages 166–171, 2018.

[25] Patrick Craig, Antika Roy, Nitin Varshney, and Navid Asadizanjani. Advancing PCB assurance towards netlist extraction with the integration of X-Ray imaging and semi-supervised learning techniques. In *IEEE RAPID*, pages 1–2, 2024.

[26] Andy Davis. To dock or not to dock, that is the question: using laptop docking stations as hardware-based attack platforms. In *BlackHat Europe*, 2013.

[27] Siva Nishok Dhanuskodi, Xiang Li, and Daniel Holcomb. COUNTERFOIL: Verifying provenanc of integrated circuits using intrinsic package fingerprints and inexpensive cameras. In *USENIX Security*, 2020.

[28] Nathan Edwards, Jason Hamlet, and Mitchell T. Martin. Authenticating a printed circuit board. US patent 10594492B1, 2017.

[29] Monta Elkins. Nation-state supply chain attacks for dummies and you too, 2019.

[30] fail0verflow. Console Hacking 2010. 27C3.

[31] Dan Farmer. Sold down the river, 2014.

[32] Ariel J. Feldman, J. Alex Halderman, and Edward W. Felten. Security analysis of the Diebold AccuVote-TS voting machine. In *USENIX EVT*, 2007.

[33] Julian Fietkau, Starbug, and Jean Pierre Seifert. Swipe your fingerprints! How biometric authentication simplifies payment, access and identity fraud. In *USENIX WOOT*, 2018.

[34] Fist0urs, maxgrim, carmaa, and rexploit. Inception password unlocking payload: inception/modules/unlock.py. GitHub, 2017.

[35] Joe Fitzpatrick. The Tao of Hardware, The Te of Implants. Blackhat USA, 2016.

[36] Joe FitzPatrick and Matt King. NSA Playset: JTAG implants. In *Defcon 23*, 2015.

[37] Joe FitzPatrick and Mike Ryan. Tools of the NSA playset. In *Ruxcon*, 2014.

[38] The JTAG/SMC hack. Technical description of JTAG/SMC hack, bundled with the utility as a ReadMe. The Free60 version contains images that are not in the ReadMe., Nov 2009.

[39] The Xbox 360 reset glitch hack. Free60 Wiki Archive, February 2022.

[40] Ulf Frisk. Direct Memory Attack the Kernel. In *DEF CON 24*, 2016.

[41] Pallabi Ghosh and Rajat Subhra Chakraborty. Recycled and remarked counterfeit integrated circuit detection by image-processing-based package texture and indent analysis. *IEEE TII*, 15(4):1966–1974, 2019.

[42] Swaroop Ghosh, Abhishek Basak, and Swarup Bhunia. How secure are printed circuit boards against trojan attacks? *IEEE Design Test*, 32(2):7–16, 2015.

[43] Rop Gonggrijp and Willem Jan Hengeveld. Studying the Nedap/Groenendaal ES3B voting computer: A computer security perspective. In *USENIX EVT*, 2007.

[44] Green Bay Professional Packet Radio. GBPPR2. YouTube channel.

[45] Andy Greenberg. Planting tiny spy chips in hardware can cost as little as $200. *WIRED*, October 2019.

[46] Glenn Greenwald. How the NSA tampers with US-made internet routers. *The Guardian*, May 2014.

[47] Zimu Guo, Xiaolin Xu, Mark M. Tehranipoor, and Domenic Forte. MPA: Model-assisted PCB attestation via board-level RO and temperature compensation. In *AsianHOST*, 2017.

[48] Zimu Guo, Xiaolin Xu, Mark M. Tehranipoor, and Domenic Forte. EOP: An encryption-obfuscation solution for protecting PCBs against tampering and reverse engineering. *arXiv: 1904.09516*, 2019.

[49] Jacob Harrison, Nathan Jessurun, Raphael R. Dos Santos, Shajib Ghosh, Navid Asadi, and Mark Tehranipoor. Analysis of etcher configuration on part marking characteristics for counterfeit identification. In *IEEE IPFA*, 2024.

[50] John Heasman. Implementing and detecting a PCI rootkit, 2007.

[51] Grant Hernandez, Farhaan Fowze, Dave (Jing) Tian, Tuba Yavuz, and Kevin R.B. Butler. FirmUSB: Vetting USB device firmware using domain informed symbolic execution. In *ACM SIGSAC*, 2017.

[52] Andrew Huang. Keeping secrets in hardware: the Microsoft XBox case study, May 2002. AI Memo 2002-08.

[53] Andrew Huang. *Hacking the Xbox: An introduction to reverse engineering*. No Starch Press, 2003.

[54] Trammel Hudson. Modchips of the state. 35C3, 2018.

[55] Vincent Immler, Johannes Obermaier, Martin König, Matthias Hiller, and Georg Sig. B-TREPID: Batteryless tamper-resistant envelope with a PUF and integrity detection. In *IEEE HOST*, pages 49–56, 2018.

[56] Nathan Jessurun, Olivia P. Dizon-Paradis, Jacob Harrison, Shajib Ghosh, Mark M. Tehranipoor, Damon L. Woodard, and Navid Asadizanjani. FPIC: A novel semantic dataset for optical PCB assurance. *ACM JETC*, 19(2), 2023.

[57] Nathan Jessurun, Jacob Harrison, Mark M. Tehranipoor, and Navid Asadizanjani. Pinpoint: An SMD pin localization method. In *IEEE IPFA*, 2022.

[58] Nathan Jessurun, Olivia Paradis, Alexandra Roberts, and Navid Asadizanjani. Component detection and evaluation framework (CDEF): A semantic annotation tool. *Microscopy and Microanalysis*, 26(S2):1470–1474, August 2020.

[59] R Karri, F Khorrami, and P Krishnamurthy. Fuzzing/controlled excitation and multi-modal sensor monitoring/fusion for hardware-firmware-software integrity verification. Technical report, NYU Tandon School of Engineering, July 2021.

[60] Bernhard Kauer. OSLO: Improving the security of trusted computing. In *USENIX Security*, 2007.

[61] David Selasi Koblah, Ulbert J. Botero, Sean P. Costello, Olivia P. Dizon-Paradis, Fatemeh Ganji, Damon L. Woodard, and Domenic Forte. A fast object detection-based framework for via modeling on PCB X-Ray CT images. *ACM JETC*, 19(4), oct 2023.

[62] Markus G. Kuhn. Electromagnetic eavesdropping risks of flat-panel displays. In *Privacy Enhancing Technologies*, 2005.

[63] Vijay Kumar and Kolin Paul. DevFing: Robust LCR based device fingerprinting. In *MECO*, pages 1–6, 2021.

[64] Andrew Kwong, Wenyuan Xu, and Kevin Fu. Hard Drive of Hearing: Disks that Eavesdrop with a Synthesized Microphone. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 905–919, 2019.

[65] Nate Lawson. TPM hardware attacks (part 2). rdist, July 2007.

[66] Serge Leef. Supply chain hardware integrity for electronics defense (SHIELD). DARPA Software and Supply Chain Assurance Winter Forum, December 2018.

[67] A. Loveless, L. Phan, R. Dreslinski, and B. Kasikci. PCspooF: Compromising the safety of time-triggered ethernet. In *IEEE S&P*, pages 572–587, 2023.

[68] Julien Maillard, Thomas Hiscock, Maxime Lecomte, and Christophe Clavier. Side-channel disassembly on a system-on-chip: A practical feasibility study. *Microprocessors and Microsystems*, 101, 2023.

[69] Dylan Martin. AMD's Xbox, PlayStation work led to a big security feature in EPYC. *CRN*, August 2019.

[70] Matthew McGuire, Umit Ogras, and Sule Ozev. PCB hardware trojans: Attack modes and detection strategies. In *IEEE VTS*, pages 1–6, 2019.

[71] Dhwani Mehta, Hangwei Lu, Olivia P. Paradis, Mukhil M.S. Azhagan, M. Tanjidur Rahman, Yousef Iskander, Praveen Chawla, Damon L. Woodard, Mark Tehranipoor, and Navid Asadizanjani. The big hack explained: Detection and prevention of PCB supply chain implants. *ACM JETC*, 16(4), August 2020.

[72] Dhwani Mehta, John True, Olivia P. Dizon-Paradis, Nathan Jessurun, Damon L. Woodard, Navid Asadizanjani, and Mark Tehranipoor. FICS PCB x-ray: A dataset for automated printed circuit board inter-layers inspection. Cryptology ePrint Archive, Paper 2022/924, 2022.

[73] Microsoft. Virtualization-based security. Windows Hardware Developer documentation, 2017.

[74] Tahoura Mosavirik, Patrick Schaumont, and Shahin Tajik. ImpedanceVerif: On-chip impedance sensing for system-level tampering detection. *IACR TCHES*, 2023(1):301–325, Nov. 2022.

[75] Makoto Nishizawa, Kento Hasegawa, and Nozomu Togawa. Capacitance measurement of running hardware devices and its application to malicious modification detection. In *IEEE APCCAS*, pages 362–365, 2018.

[76] NSA. Developing a blueprint for a science of cybersecurity. *The Next Wave*, 19(2), 2012.

[77] Johannes Obermaier and Stefan Tatschner. Shedding too much Light on a Microcontroller's Firmware Protection. In *USENIX WOOT*, 2017.

[78] Aapo Oksman. *A Method for Detecting DRAM Bus Tampering*. PhD thesis, Aalto University, 2020.

[79] Michael Ossmann. The NSA Playset: RF retroreflectors. In *Defcon 22*, Las Vegas, NV, USA, 2014.

[80] Steven Paley, Tamzidul Hoque, and Swarup Bhunia. Active protection against PCB physical tampering. In *ISQED*, pages 356–361, 2016.

[81] Shubhra Deb Paul and Swarup Bhunia. SILVerIn: Systematic integrity verification of printed circuit board using JTAG infrastructure. *ACM JETC*, 17(3), 2021.

[82] Hammond Pearce, Virinchi Roy Surabhi, Prashanth Krishnamurthy, Joshua Trujillo, Ramesh Karri, and Farshad Khorrami. Detecting hardware trojans in PCBs using side channel loopbacks. *IEEE TVLSI*, 30(7):926–937, 2022.

[83] Gor Piliposyan and Saqib Khursheed. Computer Vision for Hardware Trojan Detection on a PCB Using Siamese Neural Network. In *IEEE PAINE*, 2022.

[84] Gor Piliposyan, Saqib Khursheed, and Daniele Rossi. Hardware trojan detection on a PCB through differential power monitoring. *IEEE TETC*, 2020.

[85] J. Robertson and M. Riley. The big hack: How China used a tiny chip to infiltrate U.S. companies. Bloomberg Businessweek, 2018.

[86] Kurt Rosenfeld and Ramesh Karri. Attacks and defenses for JTAG. *IEEE Design Test of Computers*, 27(1):36–47, 2010. Number: 1.

[87] Thomas Roth, Josh Datko, and Dmitry Nedospasov. Chip.fail, 2019.

[88] Samuel Russ and Jacob Gatlin. Three Ways to Hack a Printed Circuit Board. *IEEE Spectrum*, August 2020.

[89] Maryam Saadat Safa, Tahoura Mosavirik, and Shahin Tajik. Counterfeit chip detection using scattering parameter analysis. In *26th DDECS*, pages 99–104, 2023.

[90] Maryam Saadat Safa, Patrick Schaumont, and Shahin Tajik. Parasitic circus: On the feasibility of golden free PCB verification. In *IEEE IPFA*, 2024.

[91] Darmawan Salihun. NSA backdoor part 2, BULLDOZER: And, learn how to DIY a NSA hardware implant. Infosec, February 2014.

[92] Darmawan Salihun. NSA BIOS Backdoor a.k.a. God Mode Malware Part 1: DEITYBOUNCE. Infosec, January 2014.

[93] Mukhil Azhagan Mallaiyan Sathiaseelan, Olivia P. Paradis, Rajat Rai, Suryaprakash Vasudev Pandurangi, Manoj Yasaswi Vutukuru, Shayan Taheri, and Navid Asadizanjani. Logo classification and data augmentation techniques for PCB assurance and counterfeit detection. In *IEEE ISTFA*, 2021.

[94] Patrick Schaumont. You can hide but you can't verify: On side-channel countermeasure verification. In *Workshop on SSH-SoC*, 2023.

[95] Omer Shwartz, Amir Cohen, Asaf Shabtai, and Yossi Oren. Shattered trust: When replacement smartphone components attack. In *USENIX WOOT*, 2017.

[96] Dominic Spill. NSA Playset: USB Tools. Shmoocon, 2015.

[97] ST Microelectronics. Introduction to secure firmware install (SFI) for STM32 MCUs. AN4992, 2023.

[98] Stanislav. Mechanics of FLUXBABBITT. Loper OS, January 2014.

[99] Michaael Steil. 17 mistakes Microsoft made in the Xbox security system, 2005.

[100] Michaael Steil. Deconstructing the Xbox 'security system', December 2006.

[101] TCG. TPM main specification: Part 1 design principles, March 2011. v1.2.

[102] John True, Chengjie Xi, Nathan Jessurun, Kiarash Ahi, and Navid Asadizanjani. Review of THz-based semiconductor assurance. *Optical Engineering*, 60(6):1 – 52, 2021.

[103] Trusted Computing Group. TCG D-RTM architecture. Specification, June 2013.

[104] USBKill.

[105] Satohiro Wakabayashi, Seita Maruyama, Tatsuya Mori, Shigeki Goto, Masahiro Kinugawa, Yu-ichi Hayashi, and Michael Smith. A feasibility study of radio-frequency retroreflector attack. In *USENIX WOOT*, 2018.

[106] Johannes Winter and Kurt Dietrich. A hijacker's guide to the LPC bus. In *Public Key Infrastructures, Services and Applications*, pages 176–193, 2012.

[107] Johannes Winter and Kurt Dietrich. A hijacker's guide to communication interfaces of the trusted platform module. *Computers & Mathematics with Applications*, 65(5):748–761, 2013.

[108] Zhenyu Xu, Thomas Mauldin, Qing Yang, and Tao Wei. Runtime detection of probing/tampering on interconnecting buses. In *IEEE FCCM*, 2021.

[109] Zhenyu Xu, Thomas Mauldin, Zheyi Yao, Shuyi Pei, Tao Wei, and Qing Yang. A bus authentication and anti-probing architecture extending hardware trusted computing base off CPU chips and beyond. In *ACM/IEEE ISCA*, 2020.

[110] Jonas Zaddach, Anil Kurmus, Davide Balzarotti, Erik-Oliver Blass, Aurélien Francillon, Travis Goodspeed, Moitrayee Gupta, and Ioannis Koltsidas. Implementation and implications of a stealth hard-drive backdoor. In *ACSAC*, pages 279–288, 2013.

[111] Fengchao Zhang, Andrew Hennessy, and Swarup Bhunia. Robust counterfeit PCB detection exploiting intrinsic trace impedance variations. In *IEEE VTS*, 2015.

[112] Fengchao Zhang, Shubhra Deb Paul, Patanjali Slpsk, Amit Ranjan Trivedi, and Swarup Bhunia. On database-free authentication of microelectronic components. *IEEE TVLSI*, 29(1), 2021.

[113] Huifeng Zhu, Xiaolong Guo, Yier Jin, and Xuan Zhang. PCBench: Benchmarking of board-level hardware attacks and trojans. In *ASP-DAC*, 2021.

[114] Huifeng Zhu, Haoqi Shan, Dean Sullivan, Xiaolong Guo, Yier Jin, and Xuan Zhang. PDNPulse: Sensing PCB anomaly with the intrinsic power delivery network. *IEEE TIFS*, 18:3590–3605, 2023.