Embedded System Anomaly Detection via Boot Power Trace Analysis

by

Sky Qiao

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2025

© Sky Qiao 2025

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Embedded systems play a crucial role in safety-critical domains, and it is essential to maintain their integrity. This thesis presents a robust framework for detecting hardware and firmware anomalies in embedded systems through boot-phase power consumption analysis. The proposed Sliding Window Anomaly Detection (SWAD) method establishes a nominal boot power profile and compares new boot traces against this baseline using sliding windows. By analyzing localized power dynamics, SWAD detects deviations caused by hardware or firmware modifications while accommodating natural variations in power behaviour. Experimental validation on single-board computers and flight controllers demonstrates the method's effectiveness in identifying diverse hardware and firmware attacks, achieving overall F1 scores of 98%, 96%, and 85% across three systems used in the case studies. These results highlight the promising role of power side-channel analysis in enhancing security in complex embedded systems.

Acknowledgements

Thanks to Dr. Sebastian Fischmeister, my supervisor, for his invaluable guidance, support, and the opportunity to pursue this research. This work would not be the same without the insightful feedback of Arthur Grisel-Davy, whose rigor has continually pushed me to improve.

Thank you to Dr. Carlos Moreno, Lukas Schmidt, and Kavish Mehta—your work and support have made this thesis possible.

Special thanks to Diana for your help and encouragement.

Dedication

I dedicate this work to my partner, Shujie Liu. Thank you for your unwavering love and support, which have helped me grow into my better self.

I also dedicate this work to my parents and my brother.

Table of Contents

Author's Declaration ii

Abstract iii

Acknowledgements iv

Dedication v

List of Figures x

List of Tables xiii

List of Abbreviations xiv

1 Introduction 1

2 Background 3

2.1 Embedded Systems 3

2.2 Integrity Verification for Hardware and Firmware 4

2.2.1 Hardware Integrity Verification 5

2.2.2 Firmware Integrity Verification 5

2.3 Common Attacks 6

2.4 Side-Channel Information 8

2.4.1 Overview of Side-Channel Information 8
2.4.2 Side-Channel Information in Cybersecurity 9
2.4.3 Power Side-Channel Information 10

3 Methodology **12**

3.1 Problem Statement and Assumptions 12
3.2 Guiding Example 15
3.3 Overview of Approach 17
3.4 Data Collection 18
3.5 Data Preprocessing 20
3.5.1 Power Trace Compression via Piecewise Aggregate Approximation . 21
3.5.2 Impulse Noise Suppression via Median Filtering 24
3.6 Trace Alignment and Amplitude Normalization 26
3.6.1 Anchor Trace Establishment 26
3.6.2 Boot Start Detection 26
3.6.3 Temporal Alignment 28
3.6.4 Amplitude Normalization 31
3.7 Construction of Training and Test Datasets 32
3.8 Training Phase 33
3.8.1 Reference Trace Construction 33
3.8.2 Sliding Window Segmentation 33
3.8.3 Anomaly Score Calculation 34
3.8.4 Threshold Construction 34
3.8.5 Overall Training Algorithm 35
3.9 Testing Phase 35
3.10 Advantages of SWAD 38

4 Case Studies **40**
4.1 Evaluation Criteria . . . 40
4.2 Case Study: Single-board Computer – BeagleBone Black . . . 42
4.2.1 Nominal Setup . . . 42
4.2.2 Simulated Hardware Attacks . . . 42
4.2.3 Simulated Firmware Attacks . . . 43
4.2.4 Experiment Setup . . . 44
4.2.5 Results . . . 44
4.3 Case Study: Flight Controller – Speedybee F405 Wing . . . 52
4.3.1 Nominal Setup . . . 52
4.3.2 Simulated Hardware Attacks . . . 52
4.3.3 Simulated Firmware Attacks . . . 53
4.3.4 Experiment Setup . . . 53
4.3.5 Results . . . 53
4.4 Case Study: Flight Controller – CubePilot Cube Orange+ . . . 59
4.4.1 Nominal Setup . . . 59
4.4.2 Simulated Firmware Attacks . . . 59
4.4.3 Experiment Setup . . . 60
4.4.4 Results . . . 60
4.5 Parameter Sensitivity Analysis . . . 62
4.5.1 Piecewise Aggregate Approximation (PAA) Downsampling Rate . . . 62
4.5.2 Median Filter Kernel Size . . . 62
4.5.3 Sliding Window Size . . . 63
4.5.4 Window Overlap Ratio . . . 64
4.6 Real World Considerations . . . 65
4.7 Detection Limits of SWAD . . . 65
4.7.1 Aggregated Deviation . . . 66
4.7.2 Theoretical Duration and Amplitude Deviation . . . 66
4.7.3 False Positive and False Negative Tradeoff . . . 72

5 Discussion **75**

5.1 Boot Process Duration Determination 75

5.2 Distance Metric Selection 75

6 Conclusion **77**

References **79**

List of Figures

3.1 Example of a boot power trace. The x-axis represents time (ms) and the y-axis represents the digitizer's power amplitude readings. The pre-boot phase (before 4000 ms) shows ambient noise, while the subsequent pattern reflects the system's boot process. 14

3.2 Comparison of power traces from a nominal Speedybee F405 Wings (SFW) boot process and an attacked system. The nominal traces exhibit consistent transitions, while the anomalous trace shows a noticeable delay, indicating the impact of the Time-of-Check to Time-of-Use (TOCTOU) attack on the boot power signature. 16

3.3 Overview of the SWAD method. 17

3.4 Illustration of the setup of the power capture system using PCH. 19

3.5 A example of a Teledyne ADQ32-PCIe digitizer [74]. 19

3.6 Example power trace collected from a BeagleBone Black (BBB) during the boot process, sampled at 100 samples per ms. 20

3.7 Comparison of a power trace from a BBB before and after PAA downsampling. The downsampled trace retains dominant patterns while reducing high-frequency noise 22

3.8 Comparison of a power trace from a BBB before and after median filtering. The processed trace effectively reduces transient noise spikes while preserving key patterns. 25

3.9 Comparison of traces before and after alignment, trimming and amplitude normalization, zoomed into the first 2,000 ms for clearer visual inspection. These traces are captured from the BBB. The processed traces exhibit improved alignment. 27

3.10 Comparison of a power trace from a BeagleBone Black before and after boot start detection. The post-processed trace begins at the point of boot initiation and spans the expected boot duration, forming the *anchor boot trace*. 29

3.11 An illustration of three processed nominal traces from BBB, shown in the full window and zoomed-in views of Window 1 and Window 4, each with a window size of 825 ms. 36

3.12 An illustration of the average standard deviation of training set traces for each window. 37

4.1 An example of a BBB used in the case studies [5]. 42

4.2 An illustration comparing a nominal trace, an anomalous trace, and the reference trace for the SD card replacement scenario in case studies on BBB, shown in the full window and a zoomed-in view. 47

4.3 An illustration comparing a nominal trace, an anomalous trace, and the reference trace for the BBB replacement scenario in case studies on BBB, shown in the full window and a zoomed-in view. 48

4.4 An illustration comparing a nominal trace, four anomalous traces, and the reference trace for the peripheral additions scenario in case studies on BBB, shown in the full window and a zoomed-in view. 49

4.5 An illustration comparing a nominal trace, three anomalous traces, and the reference trace for the firmware version change scenario in case studies on BBB, shown in the full window and a zoomed-in view. 50

4.6 An illustration comparing a nominal trace, an anomalous trace, and the reference trace for the bootloader version change scenario in case studies on BBB, shown in the full window and a zoomed-in view. 51

4.7 An example of a SFW used in the case studies [75]. 52

4.8 An illustration comparing a nominal trace, three anomalous traces, and the reference trace for the SFW replacement scenario in case studies on SFW, shown in the full window and a zoomed-in view. 56

4.9 An illustration comparing a nominal trace, two anomalous traces, and the reference trace for the firmware version change scenario in case studies on SFW, shown in the full window and a zoomed-in view. 57

4.10 An illustration comparing a nominal trace, two anomalous traces, and the reference trace for the firmware modification scenario in case studies on SFW, shown in the full window and a zoomed-in view. 58

4.11 An example of a CubePilot Cube Orange+ (CPCO) used in the case studies [69]. 59

4.12 An illustration comparing a nominal trace, three anomalous traces, and the reference trace for the firmware version change scenario in case studies on CPCO, shown in the full window and a zoomed-in view. 61

4.13 Distribution of anomaly scores for test set traces in the BBB firmware version change scenario in Window 5. Each point represents the anomaly score of a test trace, with the red line indicating the anomaly threshold. 67

4.14 An illustration of the range within which sample points are considered normal, with points outside the range deemed significantly abnormal and included in L_a, alongside a nominal trace and an anomalous trace for comparison. 69

4.15 An illustration of the residual power envelope for BBB that determines the maximum allowable duration and corresponding average deviation before a trace is flagged as anomalous. 70

4.16 An illustration of the residual power envelope for BBB, where each point corresponds to a test set trace in the BBB firmware version change scenario in Window 5. Points above the envelope are considered anomalous, while points below are considered nominal. 71

4.17 An illustration of three different anomaly thresholds, along with the distribution of anomaly scores for test set traces in the BBB firmware version change scenario in Window 5. 73

4.18 An illustration of three power envelopes, along with the distribution of test set traces in the BBB firmware version change scenario in Window 5. . . . 74

List of Tables

4.1 Experiment setup for BBB: Number of nominal and anomalous traces collected for each attack scenario, along with the sizes of the training and test sets. . . . 44

4.2 Summary of anomaly detection performance for BBB. . . . 45

4.3 Experiment setup for SFW: Number of nominal and anomalous traces collected for each attack scenario, along with the sizes of the training and test sets. . . . 54

4.4 Summary of anomaly detection performance for SFW. . . . 54

4.5 Experiment setup for CPCO: Number of nominal and anomalous traces collected for each attack scenario, along with the sizes of the training and test sets. . . . 60

4.6 Summary of anomaly detection performance for CPCO. . . . 60

4.7 Performance of Sliding Window Anomaly Detection (SWAD) across different PAA downsampling rates. . . . 63

4.8 Performance of SWAD across different median filter kernel size. . . . 63

4.9 Performance of SWAD across different sliding window sizes (as a percentage of the power trace length). . . . 64

4.10 Performance of SWAD across different window overlap ratio. . . . 65

List of Abbreviations

BBB BeagleBone Black x–xiii, 20, 22, 25, 27, 36, 40, 42–51, 53, 60, 62, 65–68, 70, 71, 73, 74, 77

BPV Boot Process Verifier 2, 11, 35

CPCO CubePilot Cube Orange+ xii, xiii, 40, 59–61, 77

PAA Piecewise Aggregate Approximation viii, x, xiii, 17, 21–25, 44, 53, 62, 63

PCH Power Capture Hardware 18

SFW Speedybee F405 Wings x–xiii, 15, 16, 40, 52–54, 56–59, 77

SWAD Sliding Window Anomaly Detection xiii, 2, 8, 11, 12, 17, 25, 38–40, 42–46, 52–55, 59, 60, 62–66, 68, 77, 78

TOCTOU Time-of-Check to Time-of-Use x, 6, 15, 16

Chapter 1

Introduction

Embedded systems are widely used in safety-critical domains such as automotive, military, and industrial control [81], and it is crucial to ensure their security. While there exist conventional security measures for embedded systems, such as code signing for firmware protection and ARM TrustZone for hardware security, their effectiveness can be compromised. For instance, any mishandling of the cryptographic keys can undermine the protection provided by code signing [24]. Similarly, TrustZone-based hardware security mechanisms have also been shown to be vulnerable to attacks, demonstrating that even built-in hardware security is prone to exploitation [26]. Furthermore, TrustZone-based approaches often introduce power and computational overhead, along with high implementation costs, limiting their practicality for many embedded systems [40]. Additionally, the limited observable outputs of embedded systems [62] make integrity verification challenging. These factors leave embedded systems susceptible to various attacks, including firmware tampering, hardware trojans, and unauthorized modifications [77, 62].

To address these vulnerabilities, we propose leveraging side-channel information, specifically power traces, to detect anomalies in the hardware and firmware of embedded systems. Side-channel information provides a non-intrusive way to gain insights into the internal state of a system. Power traces, in particular, are well-suited for this purpose because of their correlation with system activity and can provide insights into hardware and firmware behaviour without requiring direct access to internal components [50, 70]. In this study, we focus on power consumption during the boot phase, as this segment of the power traces captures both hardware and firmware activities and exhibits consistent, repeatable patterns [23]. This consistency makes boot-phase power traces a reliable basis for anomaly detection.

Our proposed power-based side-channel methods complement traditional security mechanisms by addressing their limitations. We will demonstrate that these methods are applicable across diverse platforms and capable of detecting a broad range of attacks. Furthermore, we will show that power-based approaches are more cost-effective than traditional hardware security measures, requiring no additional power or computational overhead from the systems, making them suitable for deployment in embedded systems.

Our work is inspired by the Boot Process Verifier (BPV) approach proposed by Grisel-Davy et al. [24], which demonstrated the potential to detect firmware anomalies by monitoring a device's power consumption during its boot process. Building upon the BPV's framework, this thesis introduces SWAD, a method designed to detect both hardware and firmware anomalies in embedded systems. SWAD constructs a reference profile of nominal boot-time power consumption behaviour, where deviations from this profile may indicate firmware tampering or hardware compromise.

Unlike the full-window comparison used in BPV, SWAD performs localized analysis of boot power traces using sliding windows. It computes an anomaly score threshold for each window, enabling the method to adapt to varying levels of local variance and adjust its sensitivity to deviations. Additionally, this localized approach also allows SWAD to detect short-lived but critical anomalies that may be overlooked by full-window comparisons. We evaluated SWAD on single-board computers and flight controllers, demonstrating its effectiveness in detecting a variety of hardware and firmware attacks. SWAD achieved overall F1 scores of 98%, 96%, and 85% across the three systems used in our case studies, whereas the benchmark method based on full-window comparison, as used in BPV, achieved overall F1 scores of 92%, 50%, 51%.

The organization of this thesis is as follows: Chapter 2 provides the necessary background information. Chapter 3 formulates the problem and details the SWAD methodology. Chapter 4 presents the case studies and analyzes the experimental results. Finally, Chapter 6 concludes the thesis by highlighting the main findings and contributions, as well as discussing the limitations and potential directions for future work.

Chapter 2

Background

This chapter discusses the essential background knowledge related to our study. We begin with an overview of embedded systems, their core components and their characteristics. Next, we introduce the system integrity verification methods for hardware and firmware. Given that embedded systems often lack such verification methods, we review those applied to more advanced systems and discuss their limitations when applied to embedded systems. We then provide an overview of common real-world attacks on hardware and firmware. Following this, we explain the concept of side-channel information and its relevance to security analysis, highlighting how power consumption data emerges as a particularly compelling form of side-channel information for our study.

2.1 Embedded Systems

Embedded systems are computer systems designed for specific applications, comprising three fundamental components: hardware, firmware, and software. Each layer plays a distinct role in fulfilling the tasks of embedded systems. Although firmware is a specialized form of software, we distinguish between the two by defining "firmware" as the software responsible for low-level control, while "software" refers to programs that manage higher-level commands. Below, we provide further details for each component:

- Hardware: The hardware forms the physical foundation of an embedded system, including electronic components such as microprocessors, memory, sensors, and actuators. Its design—ranging from integrated circuits to printed circuit boards—directly influences system performance, power consumption, and overall functionality [79].

- Firmware: Firmware is tightly integrated with the hardware and is typically stored in non-volatile memory. It is responsible for the low-level control and initialization of hardware components. Due to storage constraints and its essential role in system boot-up and configuration, firmware is usually compact and optimized for performance and security [16].

- Software: The software layer encompasses higher-level functionalities beyond basic hardware control. It is responsible for implementing complex operations and user interfaces tailored to the specific applications [13].

Together, the firmware and software provide commands while the hardware executes the functionalities, creating integrated systems that perform dedicated tasks.

Embedded systems are widely employed in various applications, such as industrial automation, automotive control units, and medical devices [81]. Compared to general-purpose computers, embedded systems are typically characterized by the following features [66]:

- Power Efficiency: These systems are optimized for low power consumption, making them ideal for power-constrained applications. This efficiency also helps minimize heat generation, often eliminating the need for moving parts such as fans [36].

- Single-Purpose Design: Embedded systems are engineered to perform a specific function throughout their operational lifetime. They lack redundant computational power, as they are not intended to perform additional tasks. Furthermore, they usually have limited observable outputs, focusing on executing specific tasks rather than supporting various user interactions.

- Cost-Effectiveness: Embedded systems are often more affordable due to their single-purpose functionality and the absence of expansion slots for peripherals, which reduces complexity and manufacturing costs. This cost-effectiveness makes them suitable for mass production and deployment in cost-sensitive applications [83].

2.2 Integrity Verification for Hardware and Firmware

Attacks can target any component of embedded systems, and in this study, we focus on safeguarding embedded systems from hardware and firmware attacks. Due to resource constraints, embedded systems often lack such protections. In contrast, general-purpose systems, such as personal computers (PC), typically employ more sophisticated protection

methods as they have more resources for implementing security measures. This section reviews existing general hardware and firmware integrity verification methods and discusses the challenges and limitations of applying these methods to embedded systems.

2.2.1 Hardware Integrity Verification

Hardware integrity verification traditionally depends on specialized security mechanisms designed for specific systems, providing functions such as secure key storage and platform authentication. In general-purpose computing, industry standards like Trusted Platform Module (TPM) and Intel Software Guard Extensions (SGX) serve this role [48, 28]. However, embedded systems lack widely applicable security solutions due to their diverse hardware architectures. For instance, the ARM TrustZone security feature is exclusive to ARM-based processors like Cortex-A and Cortex-M [84, 44], while systems using other architectures, such as MIPS32 M5150 [47], do not support it [78]. Additionally, these security mechanisms are typically embedded at the hardware level, such as on a PC motherboard, with configurations that vary across systems, reducing their portability and adaptability.

In addition, the vast diversity of embedded systems hardware architectures make designing and manufacturing tailored solutions for each architecture costly. Moreover, these components require extra power and computational resources. Many embedded systems operate under constraints such as limited power and processing capacity, making it challenging to integrate resource-intensive solutions [40, 84].

Furthermore, even the tailored security solutions can still have vulnerabilities. A notable example is the CLKSCREW attack [76, 58], which exploits dynamic voltage and frequency scaling mechanisms to bypass ARM TrustZone protections and gain unauthorized access to secure areas.

Therefore, there is a pressing need for effective, adaptable and resource-efficient security solutions that can ensure hardware integrity across diverse platforms without imposing significant overhead or necessitating extensive customization.

2.2.2 Firmware Integrity Verification

Firmware integrity verification techniques, such as Secure Boot, use cryptographic code signing to ensure that only authorized firmware is executed, preventing the execution of unauthorized or malicious code [24]. However, these methods rely on the secure management of cryptographic keys, which are often distributed among multiple organizations or

vendors. Any mishandling or compromise of these keys poses a significant threat to system integrity [24].

Additionally, vulnerabilities in Secure Boot implementations can be exploited to bypass these protections. For instance, the BlackLotus UEFI bootkit takes advantage of such weaknesses to disable Secure Boot, effectively undermining firmware integrity [49]. Attackers can also exploit race condition-based techniques, such as TOCTOU attacks, which manipulate the delay between verification and execution, allowing them to modify firmware after it has been verified but before it is loaded. A notable example of this method was used to bypass Secure Boot in Cisco systems, demonstrating its real-world feasibility [33, 82].

These limitations of existing firmware integrity verification methods underscore the need for more effective and robust safeguards.

2.3 Common Attacks

Understanding the types of attacks embedded systems may face is essential to protect them from hardware and firmware integrity compromises. In this section, we provide an overview of documented real-world attacks that exploit vulnerabilities in hardware and firmware.

Common Attacks on Hardware

Hardware vulnerabilities in embedded systems often arise from flaws in physical components or manufacturing processes [52]. In this study, we discuss attacks that involve physical modifications, where attackers exploit these weaknesses by altering or replacing hardware components, thereby compromising system integrity [51]. Common physical modifications include:

1. Component Replacement Attacks: Attackers replace system components—including storage devices or entire systems—with counterfeit or malicious versions to introduce covert functionalities. For example, hardware Trojans embedded in integrated circuits can facilitate unauthorized access or data leakage [77].

2. Peripheral Injection Attacks: Unauthorized peripherals may be introduced into a system to manipulate signal flows or extract sensitive data. A notable example is the USB Rubber Ducky [19], a device resembling a standard USB flash drive but

functioning as a keystroke injection tool. Users may overlook its presence since it appears to be an ordinary USB device. However, once connected, it acts as a virtual keyboard, executing pre-programmed keystrokes to control the system, execute commands, or leak data.

3. Signal Injection Attacks: Attackers can inject malicious signals into a system to alter its behaviour or extract information. Electromagnetic signal injection attacks, for instance, can exploit vulnerabilities in differential signalling, leading to unauthorized control or data breaches [32].

Common Attacks on Firmware

Vulnerabilities in firmware can arise from various sources, such as poor development practices, flawed update mechanisms, and deliberate tampering during manufacturing. The increasing complexity of firmware update mechanisms and the lack of adequate security practices lead to increased risks associated with firmware security [24]. Common firmware attack techniques include:

1. Firmware Modification Attacks: Attackers exploit weaknesses in firmware update processes to install malicious firmware. For instance, insecure implementations of Platform Key in UEFI systems can allow attackers with compromised private keys to sign and deploy malicious firmware that appears legitimate [57].

2. Firmware Version Change Attacks: Attackers may manipulate firmware update mechanisms to force a downgrade or upgrade to an insecure version, effectively reintroducing previously patched vulnerabilities, as illustrated in [54].

3. Bootloader Attacks: The bootloader, which is responsible for initializing the operating system, can be another target for attack. By exploiting weaknesses in bootloader authentication mechanisms, attackers can bypass security checks and load malicious firmware. For example, one study describes a bootloader vulnerability where using custom arguments allows attackers to bypass authentication, granting access to sensitive system resources such as password hashes [56].

4. Firmware Configuration Manipulation: Attackers can modify firmware configurations to disable critical security settings or weaken integrity checks. Such manipulations can persist across reboots and enable the execution of unauthorized software, leading to integrity breaches [55].

These hardware and firmware attacks highlight the risks associated with embedded systems that lack robust safeguards. In Chapter 4, we will conduct case studies by simulating some of these attacks to evaluate the effectiveness of our proposed SWAD method in defending against them.

2.4 Side-Channel Information

In this section, we provide an overview of side-channel information in cybersecurity, examining its role from both adversarial and defensive perspectives. We then focus more specifically on power side channels, discussing their characteristics and relevance in greater detail.

2.4.1 Overview of Side-Channel Information

Side-channel information refers to measurable signals or phenomena arising from the physical implementation of systems, which can reflect the systems' internal states or behaviors [63]. Common sources of side-channel information include:

- Power Consumption: Variations in power consumptions can disclose information such as the types of operations performed, their intensity and duration. Specific tasks like encryption, complex computations, or idle states exhibit distinct power usage patterns, enabling detailed profiling of internal processes and activities [38].
- Timing Information: Measurements of operation durations or response times can reveal internal computational characteristics. For example, variations in execution time may indicate the use of fixed Diffie-Hellman exponents, RSA private key structures, or other cryptographic parameters [37]. These timing variations reflect data-dependent behaviors that offer insight into the underlying computations.
- Electromagnetic Emissions: Electromagnetic signals emitted by electronic circuits can reveal information about internal system activity, including instruction execution, data transfers, memory accesses, and cryptographic computations, along with their timing and duration [1].
- Acoustic Signals: Mechanical noises generated by hardware components, such as fans, hard drives, or relays, can reflect the operational load, activity intensity, and even operational sequences within a system. Beyond these mechanical sources, high-pitched

noises emitted by electronic components can reveal detailed information about the software's internal operations, including sensitive security-related computations. For instance, different RSA keys have been shown to produce distinguishable acoustic patterns, and full RSA keys have been extracted using sound captured from a mobile phone or a microphone several meters away during decryption tasks [21].

2.4.2 Side-Channel Information in Cybersecurity

Traditionally, side-channel information has been viewed as a security concern, as it can indirectly reveal sensitive internal states. Rather than attacking cryptographic algorithms directly, adversaries exploit observable side effects of system execution to bypass in-system protections [24]. For example, Kocher [37] demonstrated timing attacks that exploit variations in the duration of cryptographic operations to recover secret keys from systems using RSA, Diffie-Hellman, and other algorithms, highlighting how computationally inexpensive measurements could pose significant risks to widely used cryptosystems. Similarly, Genkin *et al.* [21] introduced an acoustic cryptanalysis attack capable of extracting complete 4096-bit RSA keys from laptop computers by analyzing high-frequency sounds emitted during RSA decryption. Both studies underscore how side-channel information can significantly weaken the security of cryptographic systems.

Side-channel attacks can pose significant risks to security-critical systems, prompting extensive research into countermeasures. Many existing work focuses on securing cryptographic implementations by developing techniques such as masking, shielding, and noise insertion to minimize information leakage [50]. For instance, Ding et al. [14] proposed a statistical model for higher-order Differential Power Analysis (DPA) attacks on masked devices, which evaluates how factors like algorithmic confusion, signal-to-noise ratio, and masking influence leakage. These efforts aim to reduce unauthorized information exposure and strengthen system resilience against increasingly sophisticated side-channel threats.

Conversely, side-channel information can also serve defensive purposes. When employed proactively, side-channel analysis can provide valuable insights into system states, facilitating the detection and analysis of anomalous behaviors [24]. For embedded systems, defensive mechanisms based on side-channel information are particularly advantageous because they are non-intrusive and require minimal or no modifications to existing system designs.

2.4.3 Power Side-Channel Information

Power consumption has a direct link to system operations, allowing adversaries to infer internal states based on physical processes and circuit activity [43]. Its effectiveness as a means of extracting sensitive information has been demonstrated across a range of environments. For example, Lipp *et al.* showed that software-based power analysis can recover cryptographic keys on general-purpose computing systems, compromising system security [42]. Building on this foundation, Conti *et al.* applied fine-grained energy profiling to uniquely identify laptop users, achieving over 80% accuracy within 3.5 hours of monitoring power consumption across user groups [11]. Further, Michalevsky *et al.* found that even aggregate power measurements from mobile devices can inadvertently reveal precise location information, highlighting the wide applicability of power side-channel techniques across platforms [46].

Beyond its malicious applications, power side-channel analysis has also emerged as a promising defensive tool across a range of hardware platforms. Nguyen [65] demonstrated its potential for detecting malware in conventional computing systems, while Jiménez [30] leveraged power profiles to uncover unauthorized activities on laptops. Extending these ideas to smaller-scale platforms, Han [27] showed that similar techniques are effective in microcontroller-based systems, demonstrating their applicability on resource-constrained embedded devices. These studies highlight that power consumption data can be leveraged to enhance system security.

Power consumption analysis is particularly noteworthy in the context of hardware and firmware integrity verification for embedded systems, as it offers several compelling advantages:

1. Cost-Effectiveness: While the tools for collecting power consumption data are specialized, their design can be generic, enabling application across different embedded systems. For instance, a generic tool can be engineered to collect power consumption data for various embedded systems, such as single-board computers, drones and medical devices. Furthermore, since power consumption data can be collected through external measurements at the power supply level, the measurement tool can be reused on multiple devices. This reusability also improves cost-effectiveness.

2. Effectiveness and Wide Range of Applications: Power consumption levels have a direct correlation with system activity, and when used defensively, can help detect anomalous behaviour indicative of integrity breaches across a wide range of systems. For example, [39] demonstrates that power traces can identify specific instructions

a microcontroller executes during sensor data processing. In addition, Clark et al. proposed a malware detector based on power consumption for medical devices [10], demonstrating the feasibility of using power analysis for security purposes [24].

3. Comprehensive Attack Detection: Since power consumption data reflect both hardware operations and firmware executions, it can capture a broad spectrum of potential attacks, providing a more holistic security assessment.

4. Security: Forging or modifying the power profile in an embedded system is challenging. This inherent difficulty adds a layer of security to power-based integrity verification methods [22].

These features make power trace verification a valuable addition to existing hardware and firmware integrity verification methods. The work of Grisel-Davy et al. inspires this study [24], who demonstrated the feasibility of using boot-time power traces for firmware verification. The authors proposed the BPV method to identify tampered firmware based on boot power traces. Building upon their work, this thesis introduces a method called SWAD, which targets anomaly detection in both hardware and firmware, aiming to improve detection accuracy and defend against a broader range of attacks. Similar to BPV, we focus on the boot phase of the power consumption signals, as this segment primarily captures hardware and firmware activities. In contrast, post-boot power traces also capture software activities, which exhibit more variability and complicate the isolation of hardware and firmware effects. Furthermore, power traces from multiple boot-ups demonstrate consistent and repeatable patterns, as observed by [23]. This reliability makes them a reliable basis for anomaly detection in hardware and firmware.

Chapter 3

Methodology

In this chapter, we formulate the problem and present our proposed SWAD method for anomaly detection in embedded systems. The SWAD pipeline includes data collection, preprocessing, training, and testing.

We begin with a guiding example and a high-level overview of the approach. We then detail each component of the method. First, we describe the process of collecting power traces. Next, we outline our preprocessing strategy, which focuses on dimensionality reduction and noise suppression to improve data quality. We follow this with trace alignment and amplitude normalization, which ensure consistency across traces for meaningful comparison. Subsequently, we explain how the training and test datasets are constructed. In the training phase, we compute anomaly scores for each trace and define a threshold to distinguish nominal from anomalous behavior. In the testing phase, we classify new, unlabeled traces based on this threshold.

The chapter concludes with a discussion of the advantages of the proposed approach.

3.1 Problem Statement and Assumptions

As discussed in Chapter 2, boot power traces can be used to analyze the internal states of embedded systems. We define a nominal system as one operating in its expected state, free from hardware or firmware modifications. By collecting a set of power traces during the boot sequence of such a system, we hypothesize that these traces form a consistent and identifiable signature of nominal behavior. Modifications to the system are expected to introduce measurable deviations in the boot power trace, enabling anomaly detection.

A boot power trace is defined as a real-valued time series representing the system's power consumption over time during the boot process. Each data point corresponds to an instantaneous power reading captured at regular intervals. In this context, a real-valued time series refers to a sequence of real numbers sampled at successive time steps.

We formalize the anomaly detection problem as follows:

Problem Statement 1 (Anomaly Detection with Boot Power Traces) Given a set of N nominal power traces $V = \{v_1, v_2, \ldots, v_N\}$, where each v_i is a real-valued time series of size L and duration D, representing power consumption during the boot sequence of a nominal system, and a new boot power trace u of the same size and duration, determine whether u originates from the nominal system or represents an anomaly.

We define the size and duration of a power trace as:

- Size (L): The number of samples in the trace.
- Duration (D): The total time span of the trace, measured in milliseconds (ms).

Figure 3.1 illustrates an example boot power trace. The x-axis represents time in milliseconds, and the y-axis shows the amplitude measured by the power monitoring system. Each point reflects the instantaneous power consumption of the device. Prior to approximately 4000 ms, the system is in the pre-boot phase, where the signal primarily captures ambient noise. After this point, the amplitude increases and a structured pattern emerges, reflecting the system's dynamic activity during boot-up. This segment is what we define as the boot power trace used for anomaly detection in our work.

We establish the following assumptions for this research:

1. Initial Power Spike: We assume that boot power traces exhibit an initial power spike at system startup, marking the beginning of the boot process. This assumption enables precise identification of the boot power trace's starting point, which is essential for aligning multiple traces during analysis.
2. Boot Process Duration: We assume that the duration of the boot process for the target system is known a priori. This assumption allows us to capture complete boot power traces. In practice, prior knowledge of the boot duration can be established through preliminary testing of multiple boot cycles. More details on this process are given in Section 5.1.

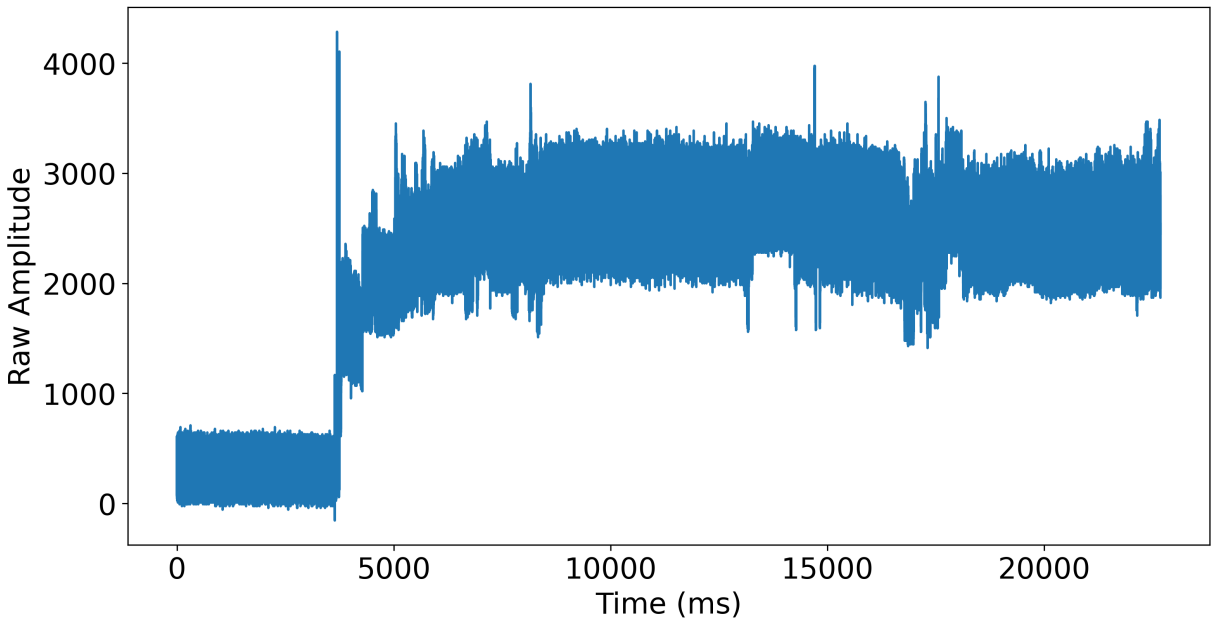


Figure 3.1: Example of a boot power trace. The x-axis represents time (ms) and the y-axis represents the digitizer's power amplitude readings. The pre-boot phase (before 4000 ms) shows ambient noise, while the subsequent pattern reflects the system's boot process.

3. Finite Boot Behaviour: We assume that the nominal device's boot process follows a consistent pattern, making its boot power traces largely predictable with minimal variation across multiple traces. This assumption allows us to effectively model the system's power consumption patterns.

In addition, this research operates under the following practical constraints:

1. Limited Training Data: In practice, the nominal data available for training is limited. While collecting more boot traces prior to deployment is possible, it can be time-consuming and resource-intensive, particularly for systems with physical setup requirements or slow reboot cycles. For instance, as shown in Chapter 4, our training set consists of only 10 nominal boot power traces. Assuming this constraint, in our case studies (Chapter 4), we limit the size of our training set to only 10 traces.

2. Unbounded Anomaly Space: We formulate our problem as a classical anomaly detection task, where only nominal data is available for training and anomalies are undefined and potentially unbounded. Unlike conventional classification problems with predefined classes, we assume access only to nominal power traces and no prior knowledge of anomalous behavior.

3.2 Guiding Example

To illustrate the effectiveness of boot power trace analysis for anomaly detection, consider a hypothetical case involving the flight controller SFW, which has a boot process duration of 14 seconds. Initially, we collect 10 power traces during the system's boot sequence under nominal conditions to establish a baseline signature of its expected behaviour. These traces are preprocessed using the methods described in Sections 3.5 and 3.6. Subsequently, we simulate a TOCTOU attack based on techniques described in [82, 59]. In this attack, an attacker exploits the temporal gap between system verification and operation by introducing modifications during the boot process. After executing this simulated attack on the SFW flight controller, we collect a power trace from the compromised system.

Figure 3.2 presents a comparison between the 10 nominal traces and the one anomalous trace. The nominal traces exhibit consistent patterns, with level transitions occurring at similar time points, whereas the trace from the attacked system displays an evident delay in these transitions. This deviation demonstrates that the TOCTOU attack alters the boot power signature, and by identifying this change, we can detect the attack.

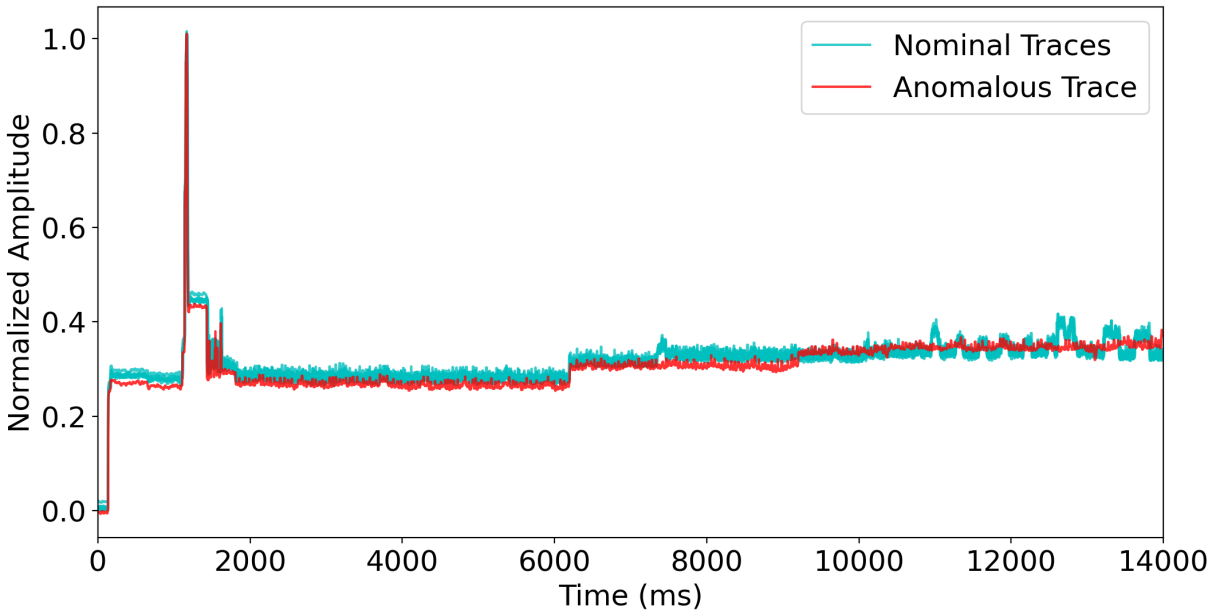


Figure 3.2: Comparison of power traces from a nominal SFW boot process and an attacked system. The nominal traces exhibit consistent transitions, while the anomalous trace shows a noticeable delay, indicating the impact of the TOCTOU attack on the boot power signature.

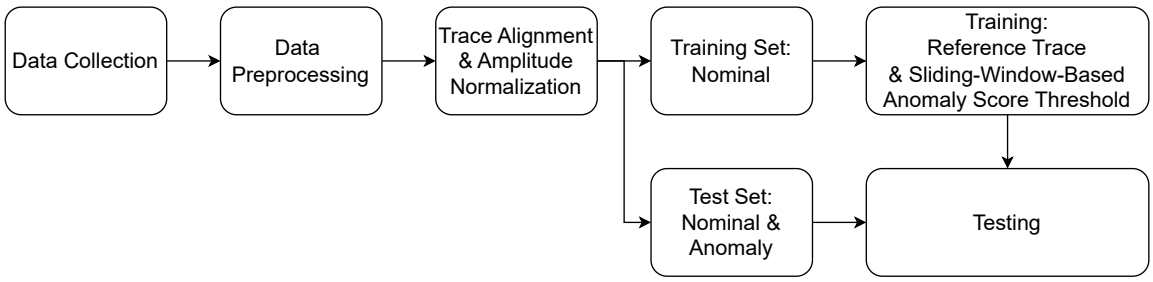


Figure 3.3: Overview of the SWAD method.

3.3 Overview of Approach

In this section, we provide a high-level overview of the proposed SWAD method, with Figure 3.3 illustrating its key components and workflow.

First, we use a power capture system to collect raw power traces from the idle state (before system power-on) to the end of the boot process. Next, the raw traces undergo a two-step preprocessing phase. Initially, PAA is applied to compress and denoise the traces; subsequently, a median filter is employed to suppress impulse noise further.

Following preprocessing, we perform trace alignment and amplitude normalization. Temporal alignment is achieved by synchronizing the traces using the initial power spike as a common starting point, thereby enabling a point-by-point comparison across traces. The traces are then trimmed to the same size and duration, corresponding to the duration of the boot process. Since our analysis focuses on the relative patterns of power consumption rather than the absolute amplitude values, we apply normalization on traces to simplify subsequent analysis.

Once we collect and process a set of nominal and anomalous traces, we partition the dataset into a training set and a test set. We construct a reference trace from the training set and segment it using sliding windows. Within each window, we establish an anomaly score threshold based on the variations among the nominal traces relative to the reference trace. Then, given a test trace, we compute an anomaly score for each window. We flag the trace as anomalous if its anomaly score exceeds the threshold in any window. We will delve into details for each of these components in the following sections.

3.4 Data Collection

In this section, we describe the method used to acquire power traces that capture the boot process. This process can be used to collect power trace from both nominal and anomalous systems.

To begin, we set up a power capture system in accordance with the methodology outlined in [61]. This setup utilizes a Power Capture Hardware (PCH) to measure the power consumption of the embedded device. Rather than connecting the embedded system directly to the power outlet, the measurement system is inserted in series between the power source and the device, in the following configuration:

$$PowerOutlet \rightarrow PCH \rightarrow EmbeddedSystem.$$

Figure 3.4 illustrates this configuration. In this thesis, we adopt the measurement setup described in [61] to collect power traces. The process involves placing a resistor in series with the system's power input and capturing the voltage drop across the resistor over time. These measurements are then used to generate power traces, which serve as the raw data for subsequent analysis. The specific PCH employed for this study is a Teledyne ADQ32-PCIe digitizer (see Figure 3.5), which is interfaced with a dedicated PC to facilitate high-speed data acquisition at a sampling rate of 100 samples per ms.

For each session, to capture a complete power trace of the boot process, we use a script to start the digitizer and then boot the target system. Let D^* represent the expected duration of the boot process. To account for the brief delay between activating the digitizer and initiating the system boot, we include a 3,000 ms buffer, resulting in a total recorded duration of

$$D = D^* + 3000\,\text{ms}.$$

Our analysis indicates that this buffer is sufficient to capture the entire boot event, and the resulting trace encompasses an initial idle segment followed by the actual boot process. We will describe the method for extracting the boot process segment in Section 3.6. Figure 3.6 illustrates an example of a collected power trace that includes the boot process. Note that the absolute level of the digitized value shown in Figure 3.6 does not convey essential information, and only the relative patterns are meaningful. We will apply normalization in Section 3.6 to facilitate analysis.

Note that the sampling rate of 100 samples per ms is determined by the maximum capability of the capture card. This reflects a trade-off between sampling frequency and the total duration that can be recorded. To maximize the amount of information captured

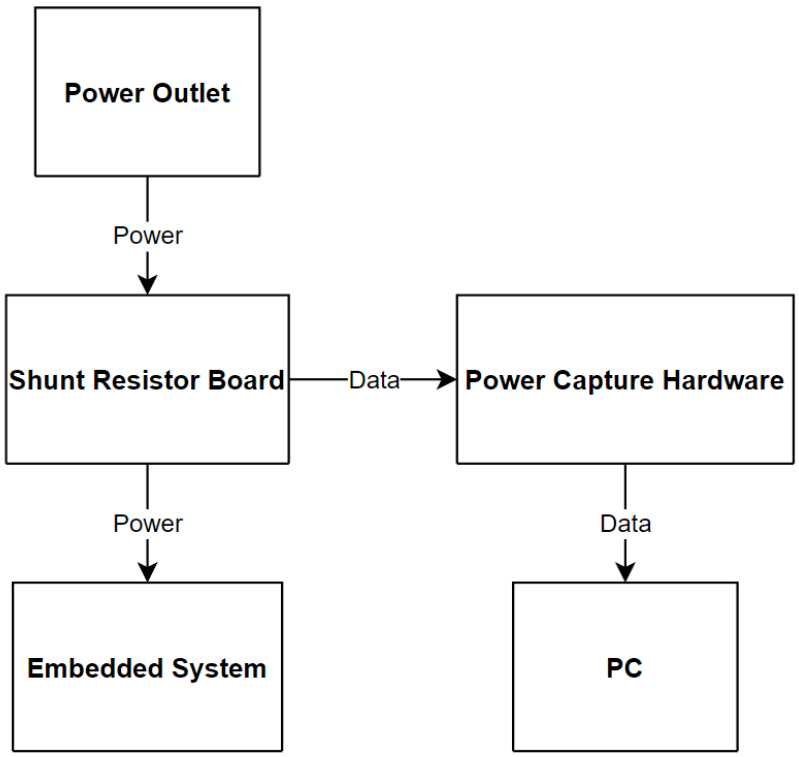


Figure 3.4: Illustration of the setup of the power capture system using PCH.



Figure 3.5: A example of a Teledyne ADQ32-PCIe digitizer [74].

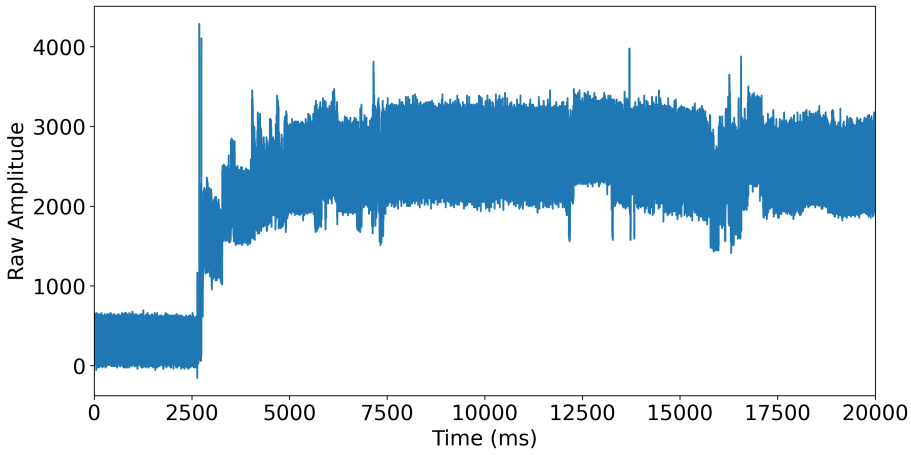


Figure 3.6: Example power trace collected from a BBB during the boot process, sampled at 100 samples per ms.

within the boot process, we select the highest frequency that still permits complete coverage of the entire boot sequence. However, this high-frequency setting also introduces considerable noise due to factors such as electrical interference and inherent sensor noise [67]. This is observable in Figure 3.6, where the raw signal exhibits frequent fluctuations. To address this, we apply noise reduction techniques during the preprocessing stage, as described in Section 3.5, to enhance pattern clarity.

After acquiring the power trace with duration D, we turn off the digitizer and transfer the power traces to a dedicated storage server to ensure that the data are securely archived and readily accessible for subsequent analysis. Using this method we collect a set of power traces from a nominal embedded system, referred to as S_{nominal}, and another set from a modified embedded system, referred to as S_{modified}. These traces will be used in subsequent sections for training and testing.

3.5 Data Preprocessing

In this section, we describe our preprocessing approach for the collected raw power traces to extract dominant patterns and facilitate analysis.

The raw power traces acquired are sampled at a rate of 100 samples per ms, resulting in very large trace sizes. For example, a 20-second trace contains 2 million samples. While this high-resolution data captures fine-grained details, it also introduces noise, which can

obscure critical patterns and complicate trace comparison [67]. To mitigate these issues, we apply a two-stage preprocessing pipeline: (1) trace compression using PAA to denoise and reduce the size while retaining temporal duration, and (2) further noise suppression via a median filter. These preprocessing steps are applied to both the nominal traces in S_{nominal} and the modified traces in S_{modified}. Detailed explanations for each stage of the pipeline are provided in the following sections.

3.5.1 Power Trace Compression via Piecewise Aggregate Approximation

In this step, our objective is to denoise and reduce the size of the power traces (i.e., the number of samples) without altering their overall duration. This allows us to reduce computational and memory requirements and facilitate the analysis of dominant patterns.

We employ PAA [35] for this purpose. Given a downsampling rate of n, we partition a trace of size L into $\frac{L}{n}$ segments of equal size, where each segment contains n samples. The aggregated value for the i-th segment, denoted by $\bar{q}_i$, is computed as follows:

$$\bar{q}_i = \frac{1}{n} \sum_{j=(i-1)\cdot n+1}^{i\cdot n} q_j, \tag{3.1}$$

where q_j is the value of the j-th sample in the original trace. The sequence of aggregated values forms the compressed trace, whose size is $L_{compressed} = \frac{L}{n}$. Importantly, while the size of the trace is reduced, the duration remains unchanged. For example, a raw power trace with a duration of 20 seconds and $L = 2{,}000{,}000$ samples, when compressed with $n = 100$, will still span 20 seconds but will contain only $L_{compressed} = 20{,}000$ samples. In this case, the effective sampling rate becomes 1 sample per ms. This approach effectively condenses the original trace into a series of aggregated values that capture the dominant patterns while discarding much of the high-frequency noise. Figure 3.7 illustrates the power trace before and after PAA, demonstrating that PAA retains the essential patterns while significantly reducing noise.

This reduced representation is vital for our application because it enables efficient comparisons and analyses of both the nominal traces in S_{nominal} and the modified traces in S_{modified} without being encumbered by excessive data volume or pervasive noise.

There are numerous alternative techniques for dimensionality reduction and denoising [31, 80, 73]. Below, we review four commonly used approaches and evaluate them relative to PAA, discussing their advantages and limitations when applied to power traces.

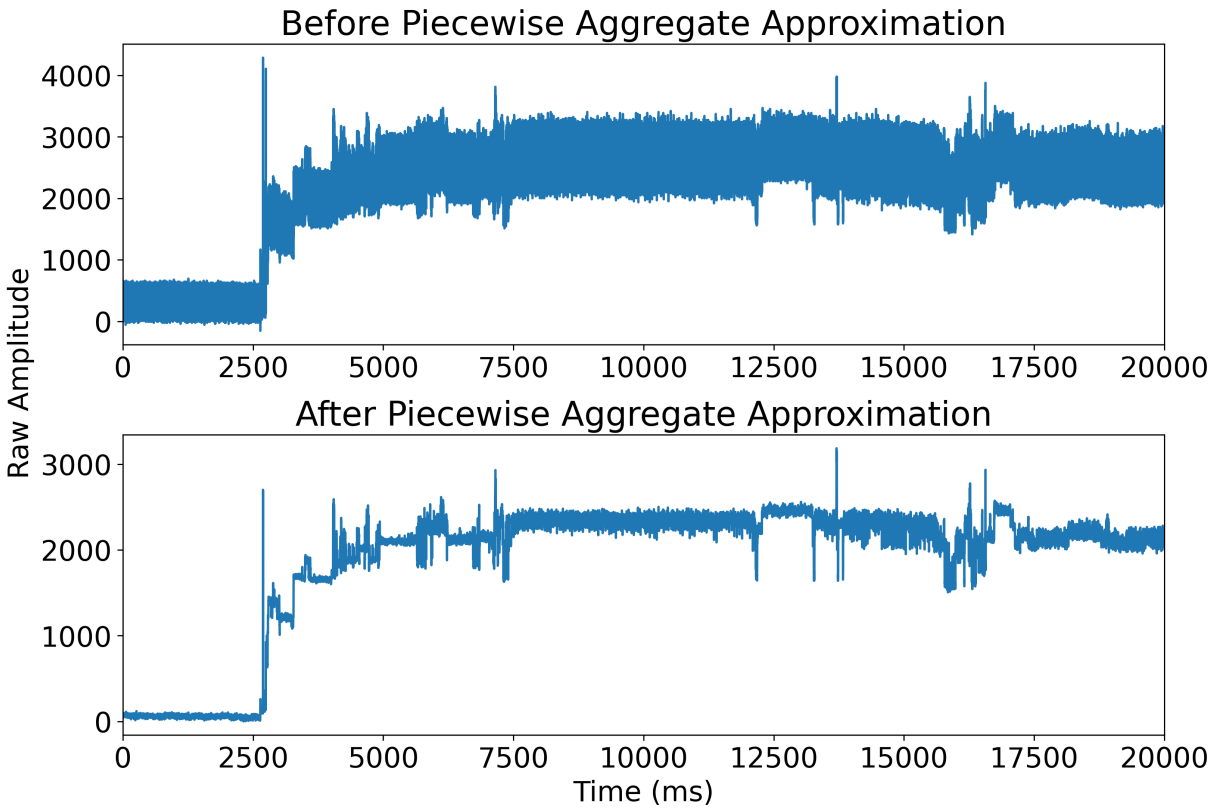


Figure 3.7: Comparison of a power trace from a BBB before and after PAA downsampling. The downsampled trace retains dominant patterns while reducing high-frequency noise

- **Uniform Downsampling**: This method reduces the size of power traces while maintaining the original duration by selecting every k-th sample from the signal [31]. It is computationally efficient and straightforward to implement. However, because it simply discards the samples between the selected points, important pattern details may be lost. Additionally, uniform downsampling does not inherently address noise suppression, leaving the resulting trace vulnerable to the effects of noise.

- **Symbolic Aggregate Approximation**: This method transforms time-series data into a symbolic representation by discretizing amplitude values into a finite alphabet. This transformation not only reduces the dimensionality of the data but also assists in reducing noise by smoothing small fluctuations. Despite these benefits, the discretization process may lead to a loss of detailed amplitude information, and subtle differ-

ences in amplitude—which might be important for detecting anomalies—could be completely lost or misrepresented in the symbolic domain. In addition, this method assumes the time series follows a Normal distribution [41], which does not necessarily hold for power traces.

- **Principal Component Analysis**: This method reduces dimensionality by projecting high-dimensional data onto a new set of orthogonal axes—principal components—that capture the maximum variance in the data [45]. While PCA can isolate and retain the most significant features by discarding noise-associated components, it does not preserve the duration and temporal ordering of the data. As a result, detecting anomalies that occur at specific moments during the boot process becomes challenging.

- **Wavelet Transform**: The wavelet transform decomposes a signal into time-frequency components, effectively capturing both local temporal and spectral characteristics [25]. This property makes it particularly effective for denoising non-stationary signals such as boot power traces. However, the wavelet transform does not inherently reduce the data size; and additional dimensionality reduction techniques are required to achieve a more compact representation. Furthermore, wavelet-based methods can be computationally intensive and demand careful selection of wavelet bases along with appropriate decomposition levels, complicating their deployment in resource-constrained environments.

Compared to these alternatives, PAA is particularly attractive for power traces for the following reasons:

- **Noise Reduction and Pattern Preservation**: By averaging over segments, PAA effectively smooths out high-frequency noise while retaining macro-level patterns—such as voltage regulator activation or memory initialization—that characterize the boot sequence [35].

- **Interpretability and Temporal Integrity**: The downsampled data produced by PAA preserves the original trace's duration, amplitude levels, and temporal order [35]. This retention of time information is critical for detecting anomalies occurring at specific moments within the boot process. For instance, if one process starts a few seconds later than its normal start time, the timing discrepancy can be clearly identified, enabling precise pinpointing of the anomaly within the time span.

- **Computational Efficiency**: Owing to its simple averaging mechanism, PAA is computationally lightweight, making it well-suited for resource-constrained environments [35].

Thus, PAA is an effective and practical solution for compressing power traces in our application.

3.5.2 Impulse Noise Suppression via Median Filtering

The second stage of our preprocessing pipeline is median filtering. Although PAA is effective at reducing high-frequency noise, transient artifacts such as impulse noise due to sporadic electrical interference can still persist. Impulse noise appears as sudden, short-duration spikes in the power trace that may distort key patterns and impede accurate analysis, and a median filter can effectively address this challenge [2]. A median filter of kernel size s, which we require to be an odd natural number, replaces each sample value with the median of the s samples in its local neighbourhood—comprising the sample itself and its $\frac{s-1}{2}$ immediate neighbours on either side.

Formally, let $\bar{q}_i$ denote the i-th sample in a PAA-compressed trace of size $L_{compressed}$. The median-filtered value $\tilde{q}_i$ is then given by:

$$\tilde{q}_i = \text{median}\left(\bar{q}_{i-\frac{s-1}{2}}, \bar{q}_{i-\frac{s-1}{2}+1}, \ldots, \bar{q}_i, \ldots, \bar{q}_{i+\frac{s-1}{2}-1}, \bar{q}_{i+\frac{s-1}{2}}\right) \tag{3.2}$$

for $i \in \{1 + \frac{s-1}{2}, \ldots, L_{compressed} - \frac{s-1}{2}\}$.

For samples with indices

$$i \in \left\{1, \ldots, \frac{s-1}{2}\right\} \cup \left\{L_{compressed} - \frac{s-1}{2} + 1, \ldots, L_{compressed}\right\},$$

where a full neighbourhood is not available, we compute the median using all the available neighbouring samples. This ensures that every sample in the trace is processed appropriately, including the boundaries.

This approach effectively balances the suppression of impulse noise with the preservation of the underlying patterns in the power trace. It is notably more robust to outliers than techniques such as mean filtering, which computes the average of samples within each kernel-a method that can be disproportionately influenced by extreme values [2]. In contrast, the median filter resists outlier effects, ensuring that sudden spikes do not distort

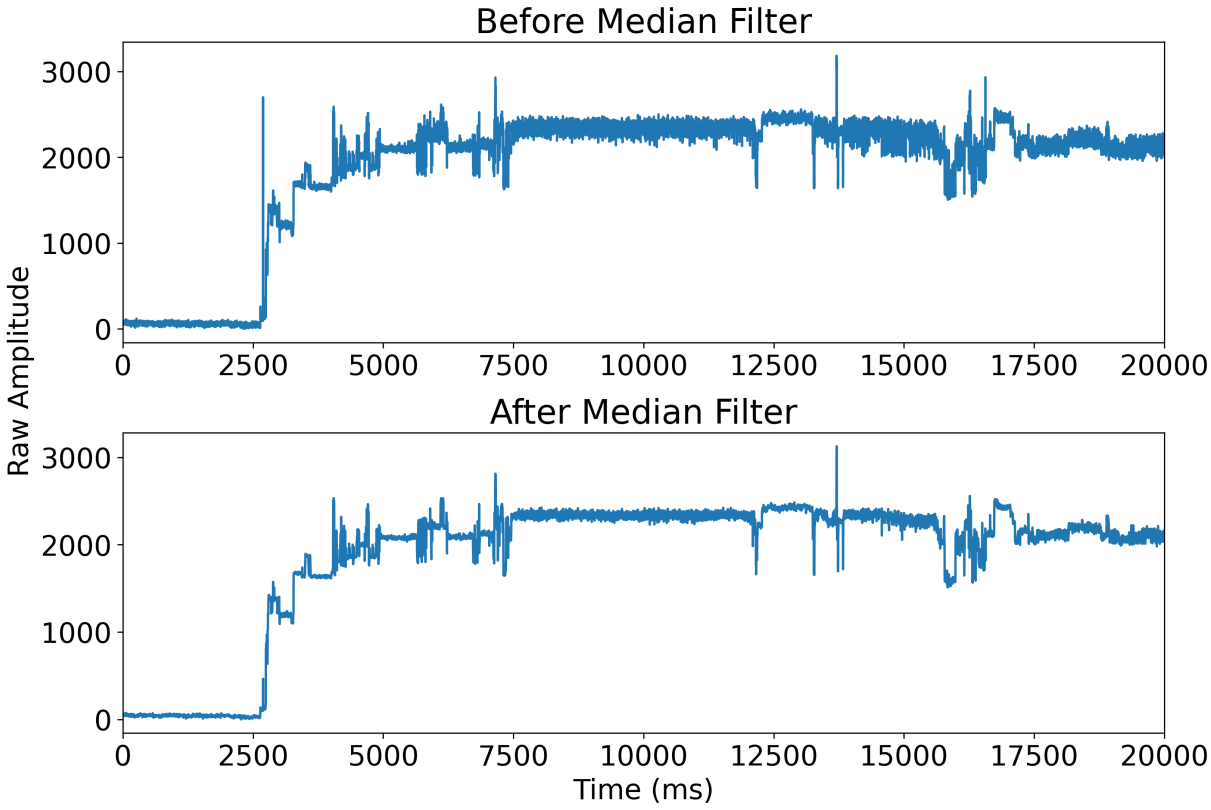


Figure 3.8: Comparison of a power trace from a BBB before and after median filtering. The processed trace effectively reduces transient noise spikes while preserving key patterns.

adjacent data. Figure 3.8 illustrates this effect, showing a power trace before and after median filtering. Notably, the spike around 4000ms is removed, while key structural patterns in the trace are retained.

By applying our preprocessing pipeline, which combines PAA and median filtering, we effectively reduce both the noise and size of the power traces while preserving the essential underlying patterns. This refined representation facilitates more accurate and reliable anomaly detection in the subsequent steps of our SWAD method.

3.6 Trace Alignment and Amplitude Normalization

As described in Section 3.4, the collected power traces comprise an idle segment (representing the pre-boot state) followed by the actual boot process. However, due to variations in the durations of these idle segments, the temporal position of the boot process start differs across traces. We perform temporal alignment to synchronize the boot segments across all traces to facilitate trace comparisons. In addition, recognizing that the relative amplitude scales are more critical than the absolute values, we apply amplitude normalization.

This section details our approach in five interconnected phases: (1) anchor trace establishment, (2) boot segment identification, (3) temporal alignment, (4) trace trimming, and (5) amplitude normalization. The complete workflow is formalized in Algorithm 1, and each component is supported by empirical validation. Figure 3.9 presents a comparison of traces before and after alignment, demonstrating improved synchronization and consistency.

3.6.1 Anchor Trace Establishment

The alignment process begins with selecting an anchor trace, which serves as the reference point for identifying the boot starting point and aligning all other traces accordingly. Although any trace from the nominal or anomalous sets could theoretically serve as the anchor, we choose an anchor from the nominal set for practical reasons. In actual deployments, we only have access to nominal system power traces for developing the anomaly detection method. Therefore, we randomly select a trace from the pre-processed nominal set as the *anchor trace*.

Aligning the remaining traces with this *anchor trace* ensures meaningful synchronized comparisons while avoiding the computational complexity associated with alternative alignment methods such as pairwise alignment.

3.6.2 Boot Start Detection

The start of the boot segment is marked by a sharp increase in power consumption from the pre-boot idle state to elevated levels as the device initializes. We detect this starting point via a two-step threshold-based method:

- Threshold Parameterization: An empirically determined threshold, T_{bootup}, is set to represent the minimum power amplitude that signifies boot initiation. This threshold

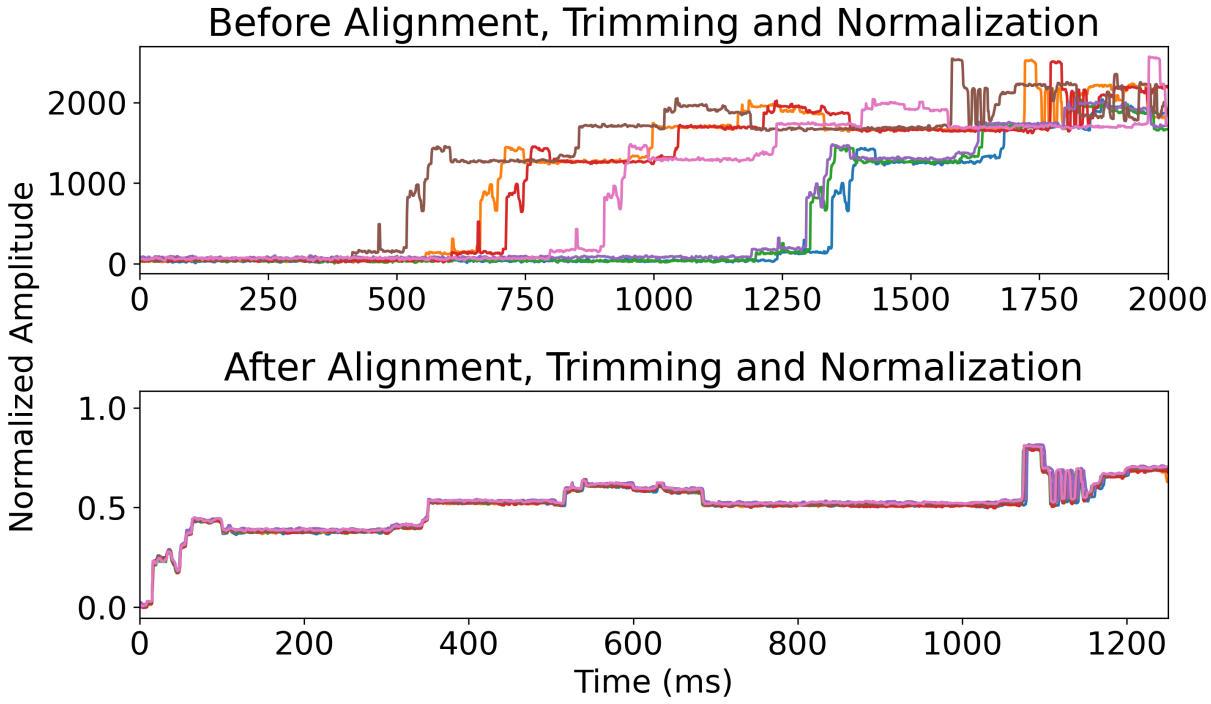


Figure 3.9: Comparison of traces before and after alignment, trimming and amplitude normalization, zoomed into the first 2,000 ms for clearer visual inspection. These traces are captured from the BBB. The processed traces exhibit improved alignment.

is derived from analyzing multiple boot sequences to balance between noise immunity and detection sensitivity.

- Boot Start Localization: The algorithm scans the *anchor trace* sequentially until it encounters the first sample with an amplitude exceeding T_{bootup}. This sample, which we denote its index as i_{boot}, is considered as the start of the boot segment.

Once the boot start in the *anchor trace* is identified, we extract the subsequent samples spanning a duration of D^*, which is the expected duration of the boot process. We refer to the resulting trace as the *anchor boot trace*. Since the power trace compression results in an effective sampling rate of 1 sample per ms, the size of the extracted trace is given by $L^* = D^* \cdot (1\,\text{sample/ms})$.

This threshold mechanism effectively filters out low-amplitude pre-boot fluctuations while reliably capturing the distinct power surge that indicates boot initiation, allowing for the extraction of the *anchor boot trace*. Figure 3.10 illustrates a sample trace before and after boot start detection and boot sequence extraction.

3.6.3 Temporal Alignment

Next, we illustrate our approach of aligning all traces to the *anchor boot trace*, which is used as the temporal reference. The process is carried out as follows:

- Anchor Segment Extraction: The first $\lfloor \frac{L^*}{20} \rfloor$ samples of the extracted *anchor boot trace* are used to create the *anchor segment*. This 5%-sized segment is purposefully selected for trace alignment for the following reasons:
  - Early boot operations are generally stable and less affected by subsequent hardware or firmware modifications, thereby bolstering the reliability of the alignment.
  - The initial phase of the boot process displays a distinctive power surge that remains highly consistent across different traces, serving as a robust feature for alignment. In contrast, reducing the segment size may increase variability across traces, complicating alignment.
  - Aligning based on the initial segment preserves temporal delay anomalies, as illustrated in Figure 4.2. In contrast, alignment using a later segment could obscure these time delay features, which are critical for anomaly detection.

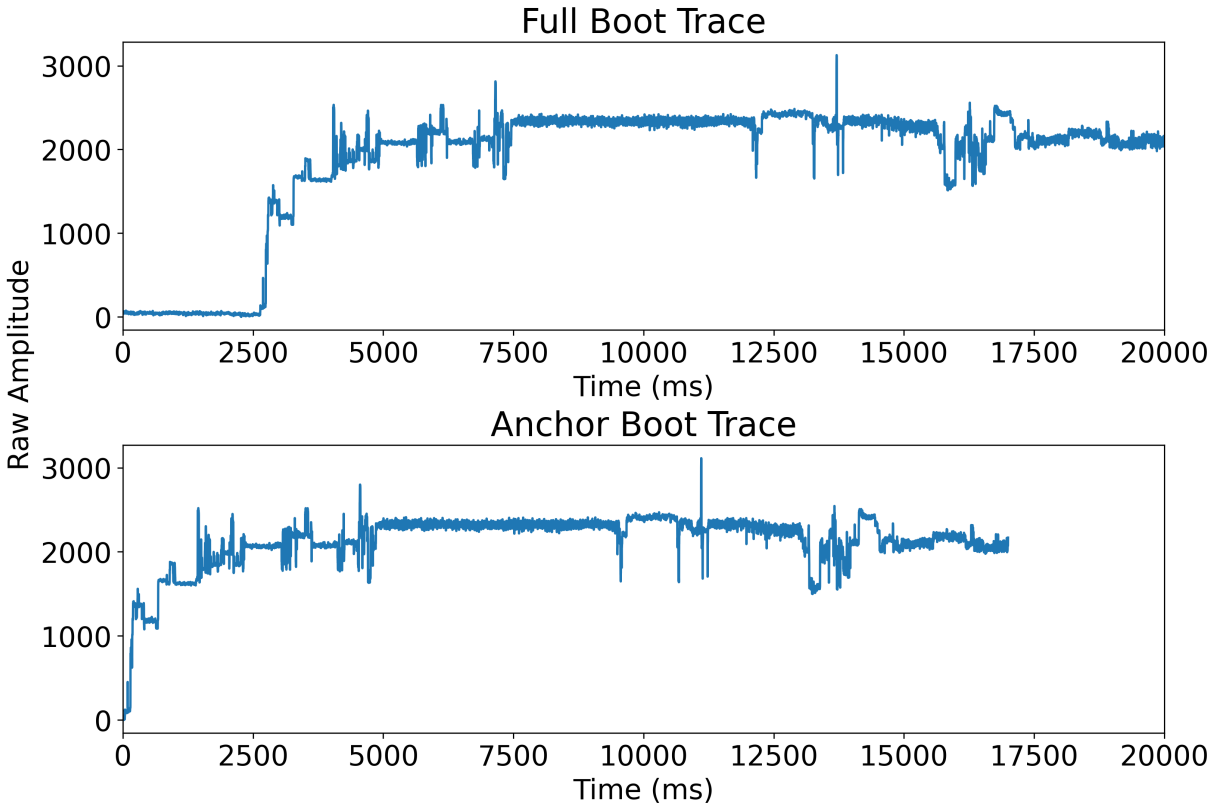


Figure 3.10: Comparison of a power trace from a BeagleBone Black before and after boot start detection. The post-processed trace begins at the point of boot initiation and spans the expected boot duration, forming the *anchor boot trace*.

- Sliding Window Distance: For each unaligned trace, we slide a window of size $\lfloor \frac{L^*}{20} \rfloor$ along the entire trace, positioning the window's starting point at each sample. At each sample index i, we compute the Manhattan distance d_i between the window segment extracted from the unaligned trace and the *anchor segment* as:

$$d_i = \text{Manhattan Distance}\Big(\big(q_i, q_{i+1}, \ldots, q_{i+\lfloor \frac{L^*}{20} \rfloor - 1}\big),\ \textit{anchor segment}\Big),$$

  where q represents the unaligned trace and q_i denotes its i^{th} sample.

- Optimal Alignment Point: The sample index i at which the corresponding window segment yields the minimum Manhattan distance d_i is selected as the optimal alignment point, denoted by i_{best}.

- Boot Power Trace Extraction: Using i_{best} as the reference, we extract the boot power trace from the unaligned trace as

$$(q_{i_{\text{best}}}, q_{i_{\text{best}}+1}, \ldots, q_{i_{\text{best}}+L^*}).$$

  This results in a boot power trace that is aligned with the anchor boot trace and has a size of L^* (corresponding to a duration of D^*).

We apply this alignment process to all traces, ensuring the extracted boot power traces are aligned at the initial boot surge.

An alternative alignment strategy is to independently apply the threshold-based boot start extraction to each trace and treat the resulting segments as aligned. However, this method can lead to suboptimal results due to noise and subtle variations in power consumption during boot initiation. For example, suppose we set $T_{bootup} = 500$ to detect boot start; one trace might first exceed this threshold with an amplitude of 600, while another might do so at an amplitude of $1,000$. In such cases, aligning the traces solely based on the threshold crossing may result in misalignment, as it might be preferable to synchronize based on a point where the amplitude in the lower-valued trace is closer to $1,000$ rather than merely the first occurrence above 500. The anchor trace-based alignment approach effectively mitigates these issues by determining the optimal alignment point utilizing information from an extended segment that captures the initial boot surge rather than relying on a single threshold-exceeding sample.

3.6.4 Amplitude Normalization

The digitizer acquired the raw data in arbitrary units due to the lack of absolute calibration for signal amplitude. Since our pattern-centric algorithm prioritizes relative power level patterns over absolute values, scaling the amplitude levels will not lead to losing essential details of the power traces. Here, we apply min-max normalization to transform the amplitude values to a scale that is easier to analyze. Importantly, we use the minimum and maximum values derived from the *anchor boot trace* to transform all traces, which ensures that relative amplitude differences are preserved across the traces. Using trace-specific minima and maxima could distort these relative levels. Specifically, for a boot power trace q, the normalized trace $q_{\text{normalized}}$ is computed as:

$$q_{\text{normalized}} = \frac{q - \min(q_{\text{anchor}})}{\max(q_{\text{anchor}}) - \min(q_{\text{anchor}})}.$$

Algorithm 1 Trace Alignment and Amplitude Normalization

Input: Unaligned power traces *unaligned_traces* (after PAA compression and median filtering)
Output: Aligned and normalized power traces *aligned_boot_traces*
aligned_traces_set ← [ ]
anchor_trace ← *random_select*(*unaligned_traces*) ▷ Anchor Trace Establishment
for $i \leftarrow 0$ **to** len(*anchor_trace*) **do**
    if *anchor_trace*$[i] \geq T_{bootup}$ **then** ▷ Boot Start Detection
        anchor_boot_trace ← *anchor_trace*$[i : i + L^*]$
        break
Initialize *scaler* ← MinMaxScaler()
anchor_trace ← *scaler*.fit(*anchor_boot_trace*)
anchor_segment ← *anchor_boot_trace*$[0 : \lfloor \text{len}(\textit{anchor_boot_trace})/20 \rfloor]$ ▷ Temporal Alignment with the first 5% as reference
for all *trace* ∈ *unaligned_traces* **do**
    lowest_distance ← ∞, *best_index* ← 0
    for $i \leftarrow 0$ **to** len(*trace*) **do**
        segment ← *trace*$[i : i + \text{len}(\textit{anchor_segment})]$
        current_dist ← distance(*segment*, *anchor_segment*)
        if *current_dist* < *lowest_distance* **then**
            lowest_distance ← *current_dist*
            best_index ← *i*
    aligned_trace ← *trace*$[\textit{best_index} : \textit{best_index} + L^*]$
    aligned_trace ← *scaler*.transform(*aligned_trace*) ▷ Amplitude Normalization
    Append *aligned_trace* to *aligned_traces*
return *aligned_traces*

3.7 Construction of Training and Test Datasets

After preprocessing the traces, we construct the training and test datasets. First, we create the training set S_{training} by randomly sampling without replacement from S_{nominal}, ensuring that S_{training} is a proper subset of S_{nominal}, i.e., $S_{\text{training}} \subsetneq S_{\text{nominal}}$. Next, we form the test set S_{test} by combining the remaining nominal traces with the set of modified traces, expressed as $S_{\text{test}} = (S_{\text{nominal}} \setminus S_{\text{training}}) \cup S_{\text{modified}}$. This configuration mirrors our anomaly detection scenario, in which only nominal traces are used during training. Once the model is trained, it is evaluated on test set traces to assess its ability to differentiate between

nominal and anomalous traces. For notation convenience in later sections, we define:

$$N = |S_{\text{training}}|, \quad M = |S_{\text{test}}|.$$

3.8 Training Phase

In the training phase, we utilize the training dataset to understand the characteristics of nominal boot power trace behaviour and establish a basis for effective anomaly detection. To accomplish this, we employ a sliding window approach comprising four key steps: (1) reference trace construction, (2) sliding window segmentation, (3) anomaly score calculation, and (4) threshold construction. These steps are detailed in the following subsections.

3.8.1 Reference Trace Construction

In this step, we construct a reference trace that captures the power consumption behavior during a nominal boot process, serving as a baseline for identifying deviations in new observations. This strategy is commonly employed in distance-based anomaly detection methods [72]. Specifically, the reference trace is generated by computing the median of values at each sample index across all traces in S_{training}. Given $S_{\text{training}} = \{v_1, v_2, \ldots, v_N\}$, where each trace v_i has a size of L^*, the reference trace v_r is constructed with samples defined as:

$$v_{r,j} = \text{median}(v_{1,j}, v_{2,j}, \ldots, v_{N,j}),$$

for $j \in \{1, \ldots, L^*\}$.

We choose median aggregation due to its robustness against outliers, ensuring that transient anomalies or measurement artifacts in individual traces do not disproportionately affect the reference trace. This approach yields a temporal consensus that accurately reflects the central tendency of nominal boot power traces within the training dataset.

3.8.2 Sliding Window Segmentation

With the established reference trace, we then segment it to be enable localized analysis of the boot power trace and enhancing detection sensitivity. We divide both the reference trace and training traces into overlapping windows with a sliding window approach. Each

window has a fixed size $W \leq L^*$ (except for the last window, which may be smaller), and overlaps $\alpha\%$ with its previous window, where $\alpha \in (0, 100)$. The first window starts at the initial sample of the boot power trace, and the kth window starting at the $\lceil (k-1) \cdot (1-\alpha) \cdot W \rceil$sample, resulting in a total of $K = \lceil \frac{L^*}{(1-\alpha) \cdot W} - 1 \rceil$ windows.

3.8.3 Anomaly Score Calculation

For each window, we calculate an anomaly score for each trace in the training set by measuring its deviation from the reference trace. The training set anomaly scores helps gauge the nominal level of deviations. The Manhattan distance is employed for this calculation; a detailed discussion of this design choice is provided in Section 5.2. Specifically, for window $k \in \{1, \ldots, K\}$ and trace $i \in \{1, \ldots, N\}$, the anomaly score a_i^k is calculated as:

$$a_i^k = \sum_{j=k \cdot W}^{(k+1) \cdot W - 1} |v_{i,j} - v_{r,j}|, \tag{3.3}$$

where $v_{i,j}$ represents the value of the j-th sample in trace v_i. This process yields a set of anomaly scores for each window across all training set traces.

3.8.4 Threshold Construction

In the final step, we utilize the anomaly scores from the training set traces to establish a threshold for each window, distinguishing nominal behaviour from potential deviations. Let A^k represent the set of anomaly scores for window k. The anomaly detection threshold is defined as:

$$\text{Threshold}^k = \mu(A^k) + 5 \cdot \sigma(A^k), \tag{3.4}$$

where $\mu(A^k)$ and $\sigma(A^k)$ denote the mean and standard deviation of the anomaly scores in the set A^k. According to Chebyshev's inequality, this threshold ensures that at least 96% of the nominal segments' anomaly scores for window k fall within this range, providing a robust basis for anomaly detection [71].

3.8.5 Overall Training Algorithm

The training phase is summarized as follows:

1. Construct the reference trace v_r using median aggregation.
2. Partition the reference and training traces into overlapping windows of size W.
3. Compute anomaly scores for all traing set traces within each window using the Manhattan distance.
4. Determine each window's anomaly threshold based on the training traces anomaly scores.

These steps collectively enable a comprehensive characterization of normal boot power consumption, providing a baseline against which deviations in test traces can be effectively identified.

It is important to note that when $W = L^*$, this sliding window approach simplifies to a full-window approach, as adopted in the BPV method [24]. However, the full-window approach does not effectively account for varying levels of local variance in traces. Additionally, aggregating deviations over the entire trace may obscure short-lived but critical anomalies, as prolonged small deviations can dominate and mask brief, significant ones. Our proposed sliding-window method, which compares trace segments to corresponding segments in v_r, is capable of capturing transient irregularities in specific phases of power consumption and adapts to variance fluctuations. Figure 3.11 and Figure 3.12 illustrate the varying level of standard deviations across windows. Our proposed approach ensures that the model captures both global and localized patterns in power consumption. We will use the full-window approach as a benchmark method to highlight the improvements brought by our sliding-window approach in Chapter 4.

3.9 Testing Phase

In the testing phase, we establish a decision criterion to determine whether a boot power trace from the test dataset is anomalous, thereby indicating the presence of hardware or firmware anomalies in the system. For a given boot power trace $u \in S_{\text{test}}$, we compute its anomaly scores for each window and compare these scores against the thresholds set during the training phase.

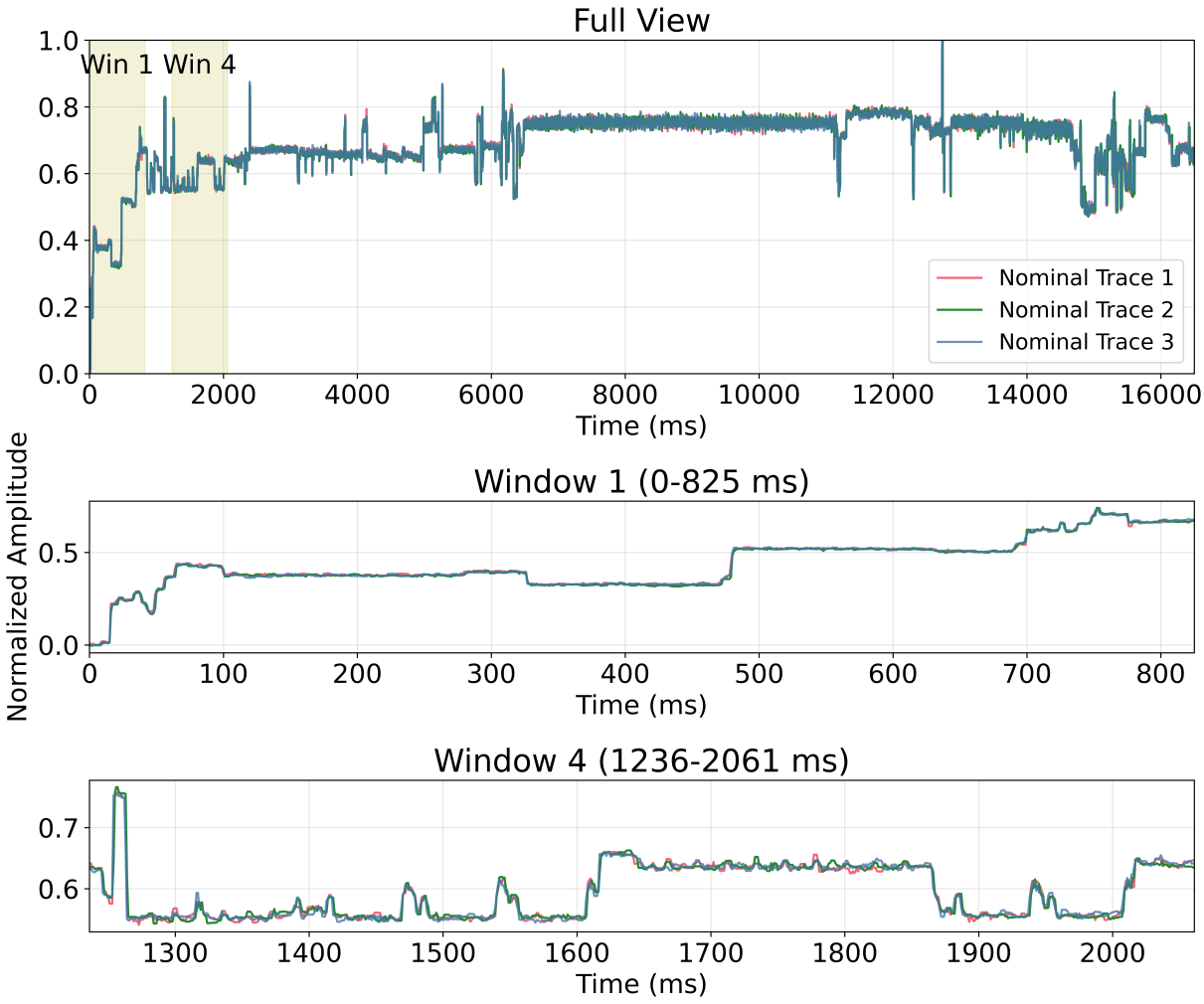


Figure 3.11: An illustration of three processed nominal traces from BBB, shown in the full window and zoomed-in views of Window 1 and Window 4, each with a window size of 825 ms.

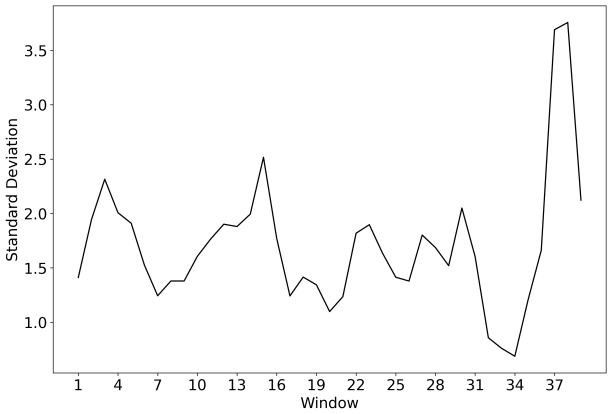


Figure 3.12: An illustration of the average standard deviation of training set traces for each window.

To determine whether the entire trace u is anomalous, we employ an OR criterion inspired by the logical OR gate: u is flagged as anomalous if it exceeds the threshold in any window. Formally, with K windows of size W, the classification for u is determined by:

$$\text{Anomalous}(u) = \bigvee_{k=1}^{K} \left(\sum_{j=k}^{k+W-1} |u_j - \mathbf{v}_{\mathbf{r}j}| > \text{Threshold}^k \right)$$

This approach improves the sensitivity of anomaly detection, especially for localized anomalies, by marking the entire trace as anomalous if any single window signals an anomaly. This feature is particularly valuable for safety-critical embedded systems, where maintaining system integrity is crucial and any anomalies can pose significant risks.

There are alternative approaches to establishing the decision criterion within our sliding window framework. For instance, one could classify anomalies based on the number of windows where the threshold is breached, such as flagging a trace as anomalous only if its anomaly scores exceed their corresponding thresholds in the majority of windows. While this method reduces false positives by requiring multiple violations, it sacrifices the ability to detect short-term, localized deviations. This trade-off conflicts with the primary goal of our sliding window approach. Thus, the OR criterion is the preferred method for our purposes.

3.10 Advantages of SWAD

Having presented the entire pipeline for the SWAD method, we now discuss four key advantages offered by this framework:

- **Adaptive Sensitivity and Localized Detection**: By conducting analysis within smaller windows, SWAD can capture varying levels of local variance in traces and adjust its sensitivity to deviations accordingly. For instance, in windows where even nominal traces exhibit high variance in power consumption patterns—indicative of high noise levels—our detection method becomes less sensitive, thereby reducing the likelihood of falsely flagging a nominal trace as anomalous and decreasing false positive rates. Conversely, in windows where nominal power traces show low variance, even a small deviation in an anomalous trace relative to other windows can be effectively detected, thus reducing false negative rates. Additionally, SWAD is adept at identifying short-lived but critical anomalies that might be overlooked in full-window comparisons. This capability is particularly valuable for detecting transient or localized deviations, which is a desirable feature for anomaly detection in safety-critical embedded systems.

- **Data and Computational Efficiency**: SWAD theoretically only requires two traces in the training set to work, and in Section 4, we will see that with 10 traces in the training set, the algorithm performs effectively, which is a significant advantage in real-world applications where data availability may be limited. This capability ensures that SWAD remains applicable and efficient even under data constraints. In addition, the distance-based approach incurs lower computational costs compared to more complex methods, such as those employing deep learning models. This efficiency makes it suitable for implementation on devices with computational constraints, enhancing its practicality.

- **Interpretability**: Alternative approaches, such as using Principal Component Analysis for dimensionality reduction or employing deep learning models for classification, often transform data into complex embedded spaces, making interpretation challenging. Additionally, the decision-making processes in deep learning models are non-linear and lack explainability. In contrast, SWAD allows users to visualize the reference trace and directly compare it with the observed trace. The sliding window approach further aids users in identifying anomalies by highlighting specific windows that signal deviations. With domain knowledge, users can potentially uncover the root causes of anomalies. This transparency fosters trust in the system and facilitates the resolution of detected anomalies.

- **Maintainability and Flexibility**: The simplicity of the SWAD framework allows for easy maintenance and modification without requiring extensive specialized knowledge in fields such as machine learning or data science. This is particularly advantageous in sensitive environments where external support is restricted due to security or confidentiality concerns. Additionally, the framework includes tunable parameters that enable users to adapt the algorithm to specific use cases and operational requirements, such as adjusting window sizes or threshold parameters. This flexibility ensures that SWAD can be tailored to different systems and anomaly detection scenarios, making it applicable across diverse domains.

These advantages position SWAD as an effective and practical solution for detecting hardware and firmware anomalies in embedded systems. Its interpretability fosters trust, making it particularly suitable for environments where transparency is crucial. Additionally, the framework's simplicity, maintainability, and flexibility empower users to actively engage in system monitoring and maintenance, allowing them to tailor the solution to their specific needs and operational contexts.

Chapter 4

Case Studies

In this chapter, we present case studies in which we apply the SWAD method to detect hardware and firmware anomalies in three embedded systems: the BBB, SFW, and CPCO. The BBB is a single-board computer, while the SFW and CPCO are flight controllers. For each embedded system, we simulate a range of real-world hardware and firmware anomalies, providing a diverse set of scenarios to evaluate the effectiveness of our proposed SWAD method. We use the full-window approach discussed in Section 3.8.5 as a baseline for comparison. We begin this chapter by outlining the evaluation criteria, followed by detailed case studies for each embedded system. Next, we assess the sensitivity of the selected parameter settings and discuss relevant real-world considerations. Finally, we conclude the chapter with an analysis of SWAD's detection limits, providing insight into the extent to which the model can reliably identify deviations from nominal profiles.

4.1 Evaluation Criteria

In this study, we use the F1 score to evaluate the performance of the SWAD method. To explain this metric, we first introduce the following concepts:

- True Positives (TP): The number of correctly identified anomalies.
- False Positives (FP): The number of instances incorrectly identified as anomalies.
- True Negatives (TN): The number of correctly identified nominal instances.

- False Negatives (FN): The number of anomalies that were not detected.
- Precision: Precision measures the proportion of correctly identified anomalies among all detected anomalies. It is defined as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

  High precision indicates that the algorithm generates few false positives, which is crucial in applications where false alarms are costly.
- Recall: Recall measures the proportion of actual anomalies that are detected. It is defined as:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

  High recall ensures that the algorithm misses a few anomalies, which is crucial in safety-critical systems.

With the above building blocks, The F1 score is defined as:

$$\text{F1 score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.1}$$

The F1 score balances precision and recall and accounts for both false positives and false negatives [8]. This balancing makes it particularly suitable for anomaly detection, where both precision and recall are important.

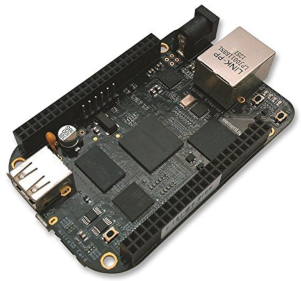


Figure 4.1: An example of a BBB used in the case studies [5].

4.2 Case Study: Single-board Computer – BeagleBone Black

In this section, we apply the SWAD method to detect hardware and firmware anomalies in the BBB (Figure 4.1), a commonly used single-board computer. Single-board computers such as BBB are characterized by their high processing power, flexible peripheral expansion, and extensive firmware support, making them widely used in robotics, IoT gateways, industrial control, and research laboratories [20]. This flexibility increases the potential attack surface, making BBB an ideal case study object for evaluating the effectiveness of SWAD in real-world scenarios.

4.2.1 Nominal Setup

The nominal setup for the BBB serves as the baseline for detecting anomalies introduced by hardware and firmware attacks. This setup includes:

- SD Card: The BBB uses a 16GB Patriot SD card for booting the operating system.
- Firmware Version: The BBB runs am57xx-debian-11.7, a stable and widely used firmware version for this device [6].

4.2.2 Simulated Hardware Attacks

The BBB is susceptible to hardware attacks that exploit its flexibility and peripheral support. Drawing on documented real-world hardware attacks as discussed in Section 2.2, we simulate the following scenarios where hardware components are replaced or added:

- SD Card Replacement: We simulate a hardware replacement attack by substituting the original SD card with another 16GB Patriot SD card. In real-world scenarios, such replacements can be challenging to detect, as the SD cards look identical. However, subtle variations between the original and replacement cards may arise from sources such as manufacturing differences or age, potentially leading to discrepancies in their boot power traces. This test evaluates whether SWAD can detect these subtle variations.

- BBB Replacement: We simulate another hardware replacement attack by replacing the original BBB with another identically configured BBB. This test assesses whether SWAD can detect subtle variations between these seemingly identical devices.

- Peripheral Additions: We simulate four hardware addition attacks by attaching peripherals—specifically, a monitor, a USB WiFi adapter, a hard disk, and a phone—to the BBB, one at a time. Due to their power demands, these additional hardware components may alter the boot power consumption behaviour. This test determines if SWAD can detect such additions.

4.2.3 Simulated Firmware Attacks

Based on documented real-world firmware attacks we discussed in Section 2.3, we simulate scenarios where the firmware running on the BBB is modified:

- **Firmware Version Change**: We upgrade or downgrade the original firmware from am57xx-debian-11.7 to different versions, including am57xx-debian-11.8, am57xx-debian-11.6, and am57xx-debian-10, one at a time. These different versions may result in varied boot power consumption behaviours due to factors such as differing initialization routines, bug fixes and optimizations, and security updates. This test assesses whether SWAD can detect such changes in firmware versions.

- **Bootloader Version Change**: Bootloader is a critical component of the boot process, responsible for loading the operating system. We upgrade the bootloader to version U-boot v2022.04-bbb.io-am335x-am57xx [4]. This upgrade may lead to differences in boot power traces for similar reasons as firmware version changes, and we assess if SWAD can detect this bootloader change.

4.2.4 Experiment Setup

We collect boot power traces from the nominal setup and various attack scenario setups, perform data processing, and construct the training and testing datasets following the methodologies outlined in Chapter 3. Specifically, the collected power traces have a duration of $D = 19,500\,\text{ms}$. After processing with PAA downsampling rate of $n = 100$ and a median filter with kernel size $s = 5$, the extracted boot power traces have a duration of $D^* = 16,500\,\text{ms}$, corresponding to a size of $L^* = 16,500$ samples. We construct sliding windows from the boot power traces, each with a size of $W = 825$ samples and overlapping $\alpha = 50\%$ of the previous window, resulting in $K = 39$ windows. We collect 30 nominal traces for each scenario and randomly select 10 of these to form the training set. We then collect 20 anomalous traces for each scenario (or each specific setup within a scenario) and combine them with the remaining 20 nominal traces to create the test set. This ensures class balance within the test set, with 20 traces per setup. Table 4.1 summarizes the number of traces collected for each scenario and the sizes of the training and test sets.

Table 4.1: Experiment setup for BBB: Number of nominal and anomalous traces collected for each attack scenario, along with the sizes of the training and test sets.

Scenario	**Nbr. of Nominal Traces**	**Nbr. of Anomalous Traces**	**Training Set Size**	**Test Set Size**
SD card replacement	30	20	10	40
BBB replacement	30	20	10	40
Peripheral additions	30	80 (20 per setup)	10	100
Firmware version change	30	60 (20 per setup)	10	80
Bootloader attacks	30	20	10	40

4.2.5 Results

We apply SWAD to each attack scenario to detect anomalous traces within each scenario's test set. The F1 score is calculated for each scenario, using the full-window approach as a baseline for comparison. We summarized the result in Table 4.2. The SWAD method achieves a high F1 score across all attack scenarios, with an overall F1 score of 98%. Notably, in the peripheral additions scenario, where the full-window approach struggles, SWAD significantly outperforms it.

Table 4.2: Summary of anomaly detection performance for BBB.

Device	Scenario	F1 score (SWAD)	F1 score (full-window)
BBB	SD card replacement	**100%**	**100%**
	BBB replacement	**100%**	**100%**
	Peripherals additions	**95%**	78%
	Firmware version change	**100%**	**100%**
	Bootloader version change	**100%**	**100%**
	Overall [1]	**98%**	**92%**

Analysis of Attack Scenarios

In this part, we provide a detailed discussion of SWAD's performance in each attack scenario.

- SD Card Replacement: Both methods achieve a 100% F1 score in this scenario. Figure 4.2 illustrates an example of a nominal trace, an anomalous trace, and the reference trace, showing that replacing the SD card introduces a delay at the start of the boot process. This delay could be attributed to differences in SD card controller initialization times or loading speeds. The delay persists throughout the entire duration, resulting in high anomaly scores for both methods, as they are calculated by aggregating point-wise distances with the reference trace. Thus, both methods can detect this change.

- BBB Replacement: Both methods achieve a 100% F1 score in this scenario. Figure 4.3 presents an example of a nominal trace, an anomalous trace, and the reference trace, revealing noticeable deviations across many segments. These variations could arise from differences in manufacturing, component aging, or usage history of the two BBB systems. Both methods successfully detect this change.

- Peripheral Additions: In this scenario, SWAD achieves a 95% F1 score, whereas the full-window approach reaches only 78%. Upon investigation, we find that the four different peripheral additions lead to two main types of pattern changes in the boot power traces. The first type is characterized by a spike in power consumption near the end of the boot sequence, observed in the cases of disk, phone, and WiFi stick additions, as illustrated in the top plot of Figure 4.4. This spike may occur because

[1] Note that this overall F1 score is calculated by first counting all the TP, FP, and FN in each attack scenario, and then applying the F1 score formula in Equation 4.1.

the peripheral does not significantly draw power during the early boot stages and only begins drawing power at a later stage, causing a noticeable change near the end. The second type is characterized by a level shift throughout the entire duration, observed in the case of monitor addition, as shown in the bottom plot of Figure 4.4. A possible explanation for the observed decrease in power consumption when connected to a monitor is that it may result from system optimizations, such as reduced GPU/CPU load or power management adjustments to accommodate the display. We find that the misclassified cases under the full-window approach mainly arise from the first type of pattern change, particularly from the phone addition case. From the plot, we can see that this deviation from the reference trace is short-lived and localized. When using the full-window approach, this deviation gets averaged out, and because the phone addition case's deviation is relatively small in amplitude compared to disk and WiFi stick additions, it is prone to be masked by long-lived but minor deviations. The sliding window approach is more sensitive to such localized changes, as the segment containing this power spike can easily detect this deviation. This makes SWAD a superior tool for detecting such anomalies.

- Firmware Version Change: Both methods achieve a 100% F1 score in this scenario. A visual inspection of Figure 4.5 reveals that the example traces for each firmware version change setup exhibit substantial differences across many segments of the boot power trace. These differences could be caused by changes in kernel configurations, drivers, or system services, which alter the system's boot behaviour. Since numerous segments have relatively large deviations, both methods can effectively detect them.

- Bootloader Version Change: Both methods achieve a 100% F1 score in this scenario. The example of a nominal trace, an anomalous trace, and the reference trace shown in Figure 4.6 indicate that modifying the bootloader results in a delay in the early part of the boot process, and noticeable amplitude deviations in several segments. Potential causes for these changes include alterations in the bootloader code execution order, differences in hardware initialization sequences, or modified timing in loading subsequent stages of the boot process. Similar to the SD card replacement and firmware version change scenarios, both methods can easily detect the delay and large amplitude deviations caused by bootloader changes.

In summary, SWAD proves to be an effective anomaly detection method for protecting the BBB against various hardware and firmware attacks. It is particularly valuable in scenarios where attacks result in short-lived deviations in the boot power trace, which are challenging to detect using the alternative full-window method.

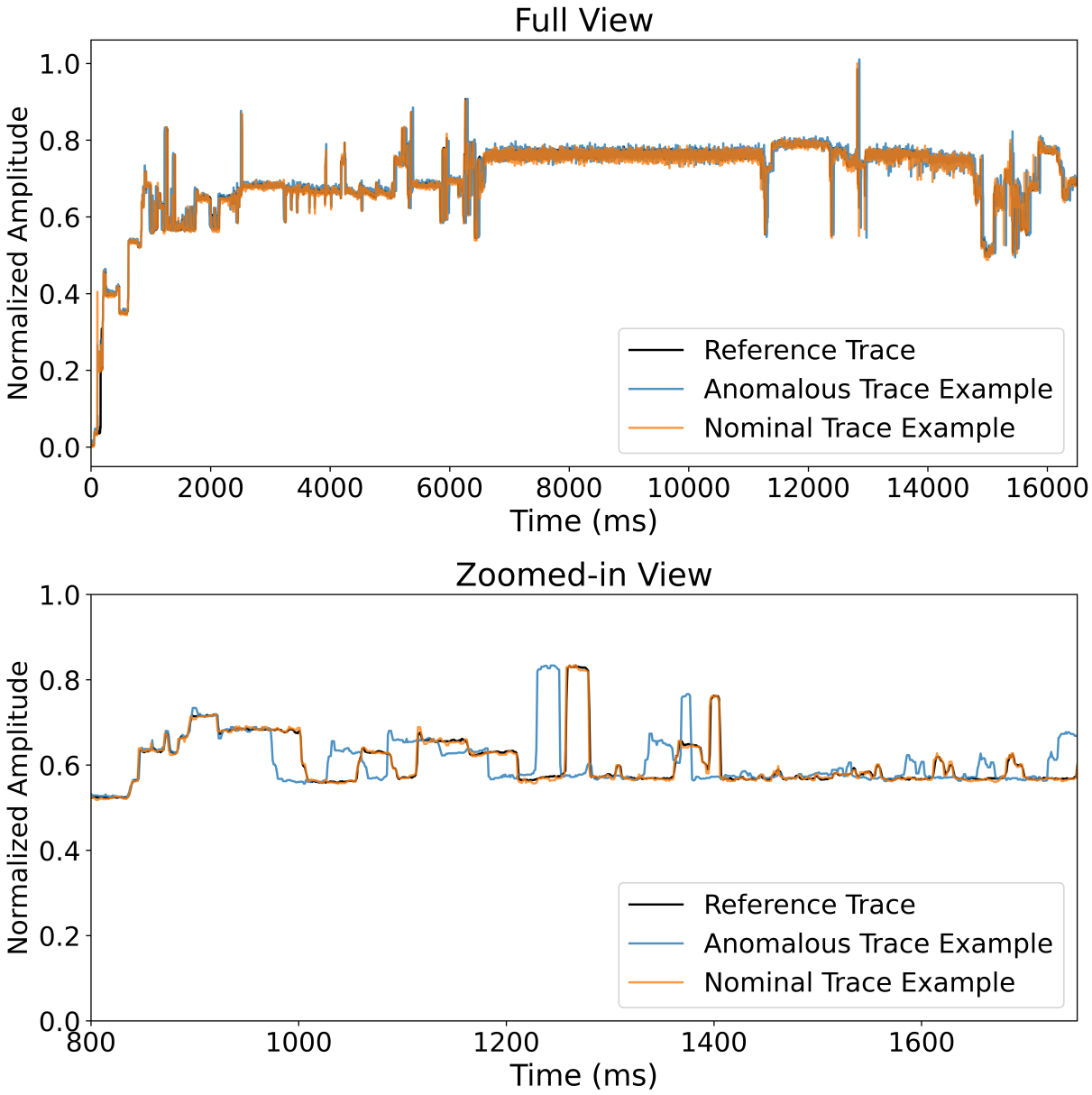


Figure 4.2: An illustration comparing a nominal trace, an anomalous trace, and the reference trace for the SD card replacement scenario in case studies on BBB, shown in the full window and a zoomed-in view.

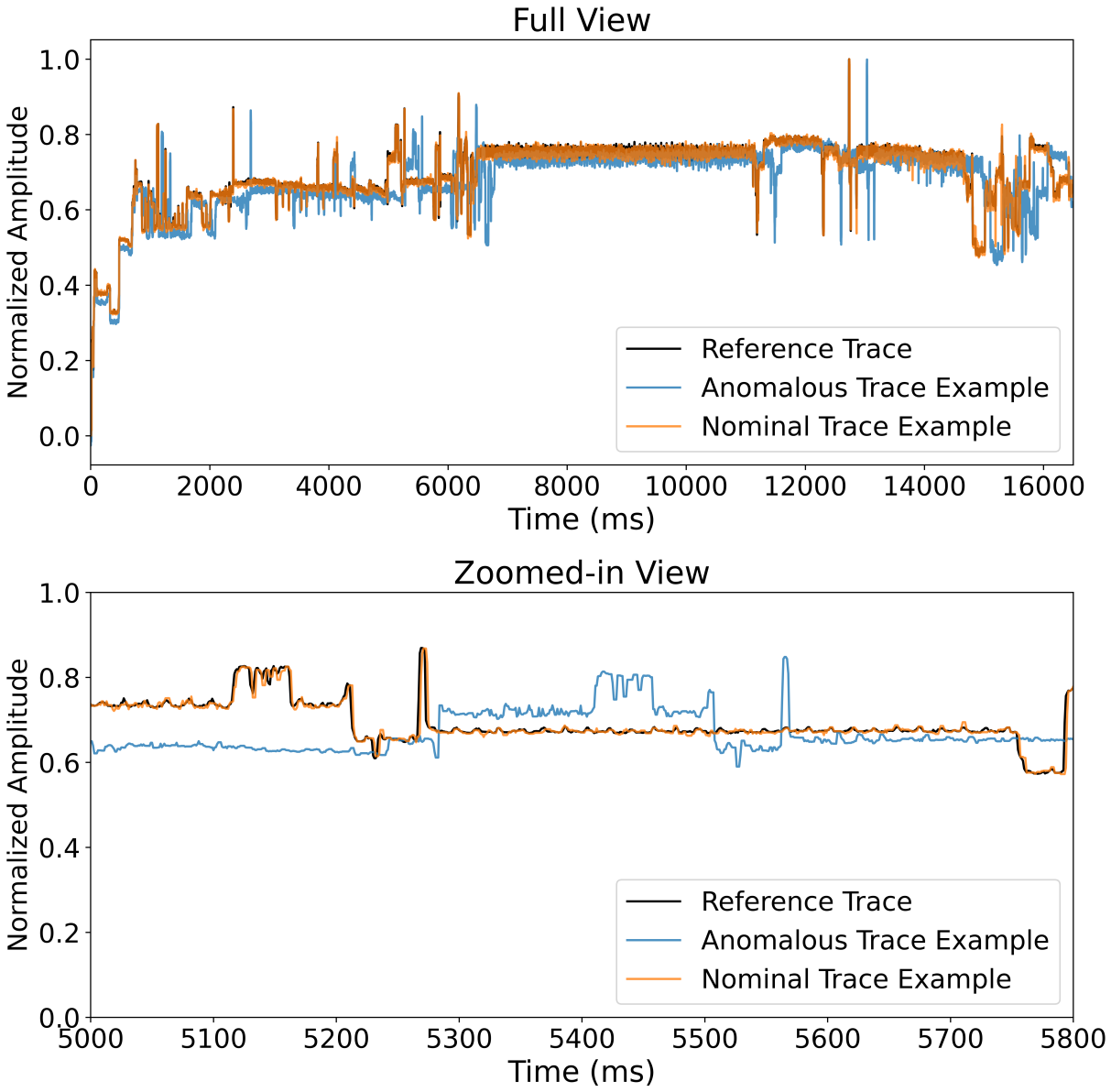


Figure 4.3: An illustration comparing a nominal trace, an anomalous trace, and the reference trace for the BBB replacement scenario in case studies on BBB, shown in the full window and a zoomed-in view.

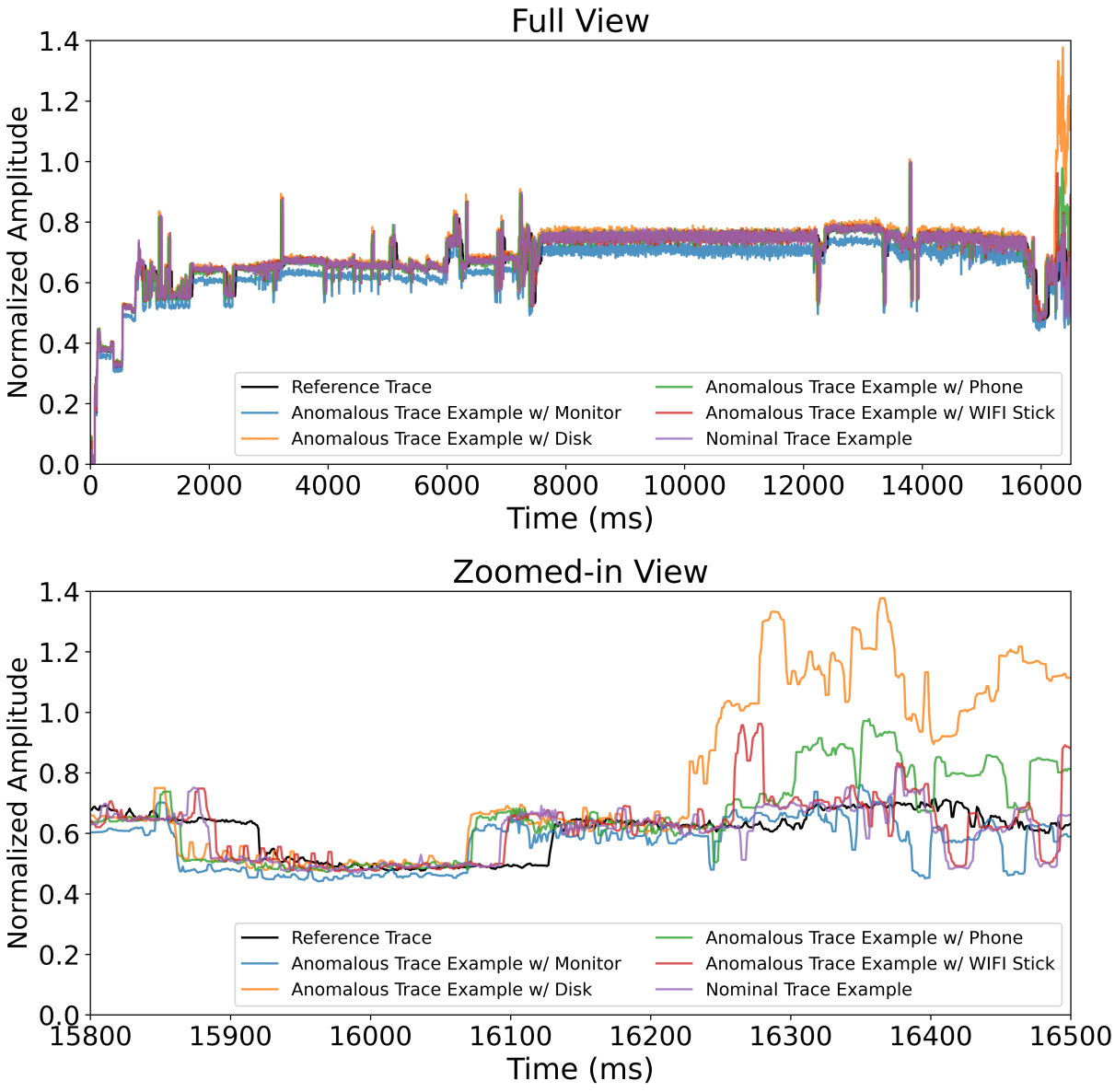


Figure 4.4: An illustration comparing a nominal trace, four anomalous traces, and the reference trace for the peripheral additions scenario in case studies on BBB, shown in the full window and a zoomed-in view.

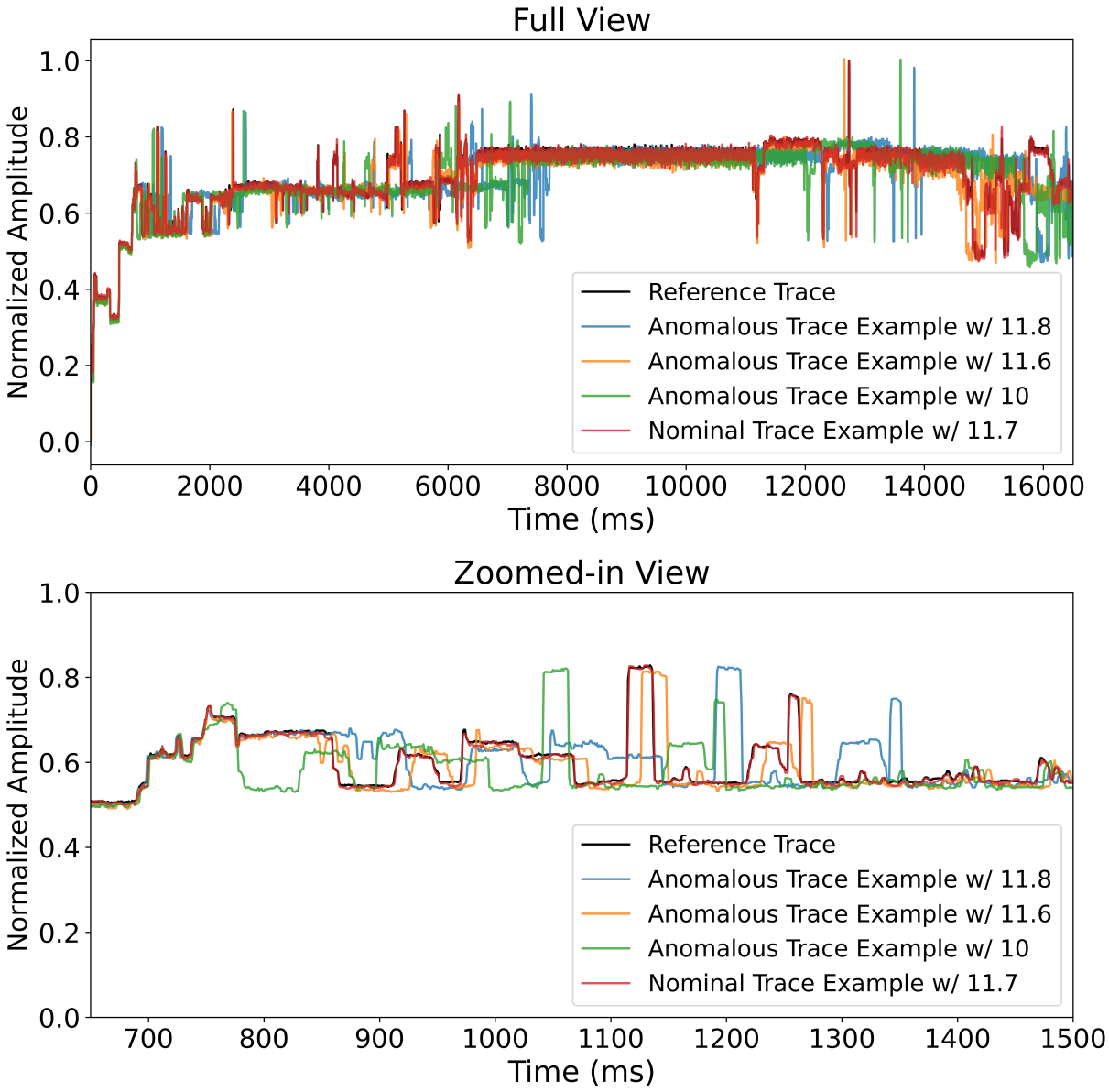


Figure 4.5: An illustration comparing a nominal trace, three anomalous traces, and the reference trace for the firmware version change scenario in case studies on BBB, shown in the full window and a zoomed-in view.

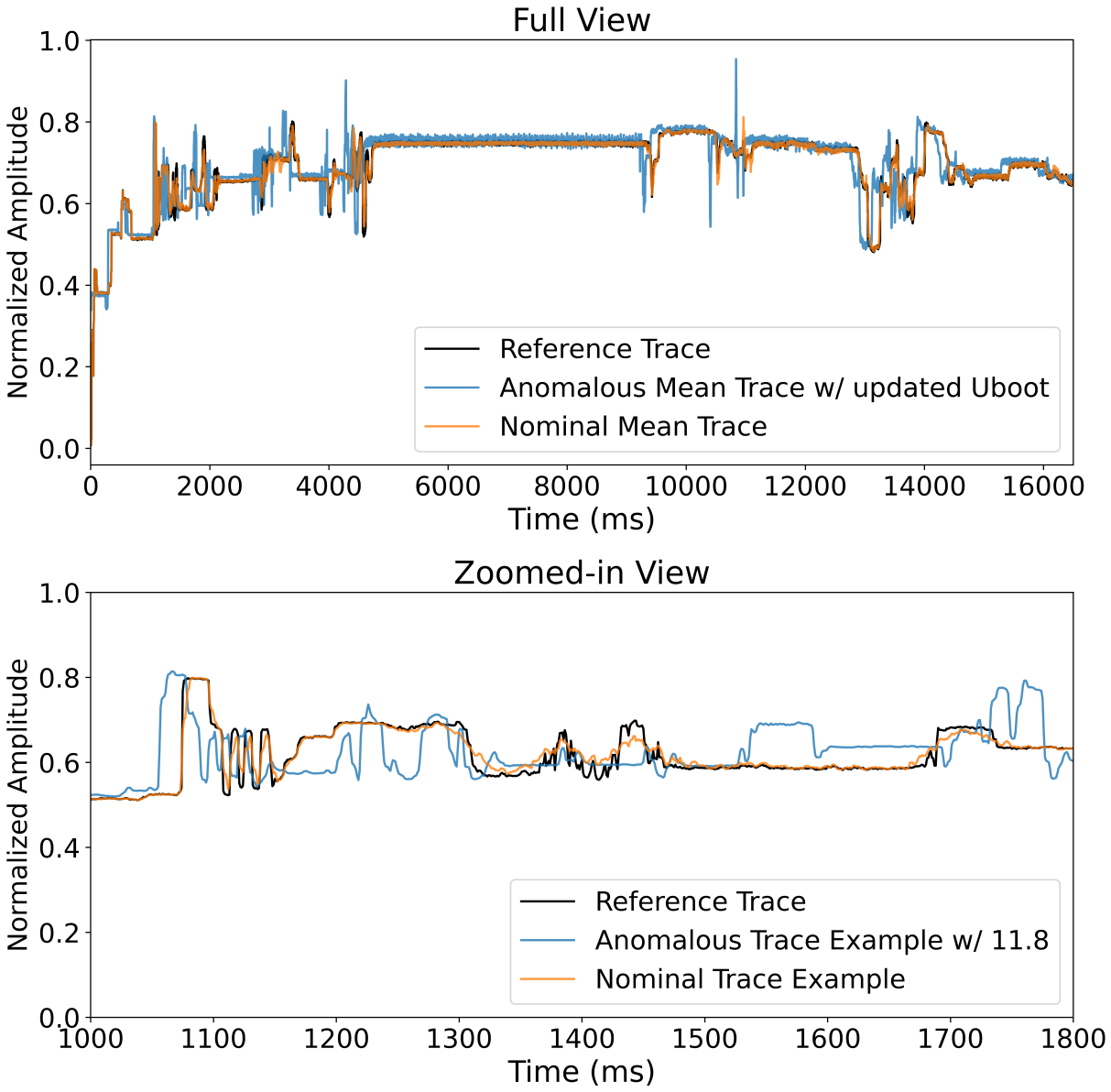


Figure 4.6: An illustration comparing a nominal trace, an anomalous trace, and the reference trace for the bootloader version change scenario in case studies on BBB, shown in the full window and a zoomed-in view.

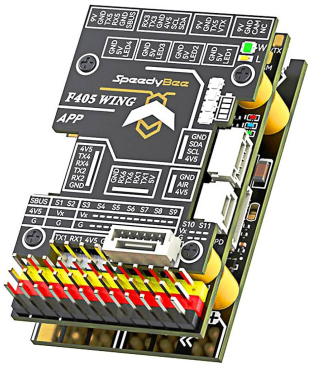


Figure 4.7: An example of a SFW used in the case studies [75].

4.3 Case Study: Flight Controller – Speedybee F405 Wing

This section examines the application of SWAD for detecting hardware and firmware anomalies in the SFW (Figure 4.7), a widely used flight controller in unmanned aerial vehicles (UAVs), commonly known as drones. Flight controllers such as the SFW manage critical real-time tasks, including sensor fusion, motor control, and flight stabilization [17]. Ensuring system integrity is crucial for its operational safety, as modifications or replacements can introduce unintended behaviour or security vulnerabilities.

4.3.1 Nominal Setup

The nominal setup for the SFW serves as the baseline for detecting anomalies introduced by hardware and firmware attacks. This setup includes:

- Firmware Version: The SFW operates on ArduPilot Plane V4.5.5, a stable and widely-used firmware version [3].

4.3.2 Simulated Hardware Attacks

For the SFW, we simulate hardware replacement attacks based on documented real-world hardware attacks as discussed in Section 4.2.2. Specifically, we replace the original SFW with three distinct units, named SFW02, SFW03, and SFW04, each running the same

firmware (ArduPilot Plane V4.5.5) and with identical configuration. Similar to the BBB, while these units appear identical, they may not necessarily exhibit the same power behaviour during the boot phase due to factors such as manufacturing differences and age. This experiment evaluates whether SWAD can detect these variations.

4.3.3 Simulated Firmware Attacks

Based on documented real-world firmware attacks we discussed in Section 2.3, we simulate scenarios where the firmware running on the SFW is modified:

- Firmware Version Change: We replace the original firmware with different versions, including ArduPilot Plane V4.5.4 and ArduPilot Plane V4.4.0. These versions might cause differences in boot power consumption behaviour due to factors such as differing initialization routines, bug fixes and optimizations, and security updates. This test evaluates whether SWAD can identify changes from different firmware versions.
- Firmware Modification: We manually alter the firmware by introducing controlled anomalies one at a time, including buffer overflows and intentional time delays, which could lead to different boot power behaviour. This scenario assesses whether SWAD can identify these modifications.

4.3.4 Experiment Setup

For the SFW, we collect power traces with a duration of $D = 17,000\,\text{ms}$. After processing with PAA downsampling rate of $n = 100$ and a median filter with kernel size $s = 5$, the extracted boot power traces have a duration of $D^* = 14,000\,\text{ms}$, corresponding to a size of $L^* = 14,000$ samples. From these boot power traces, we construct sliding windows with a size of $W = 700$ samples and overlapping $\alpha = 50\%$ of the previous window, resulting in $K = 39$ windows. The remainder of the experiment setup follows the same procedure outlined for the BBB in Section 4.2.4. Table 4.3 summarizes the number of traces collected for each attack scenario, and the sizes of the training and test sets.

4.3.5 Results

In each attack scenario, we utilize SWAD to identify anomalous traces within the test set and calculate the F1 score. We then compare SWAD's performance with the full-window

Table 4.3: Experiment setup for SFW: Number of nominal and anomalous traces collected for each attack scenario, along with the sizes of the training and test sets.

Scenario	Nbr. of Nominal Traces	Nbr. of Anomalous Traces	Training Set Size	Test Set Size
SFW replacement	30	60 (20 per setup)	10	80
Firmware version change	30	40 (20 per setup)	10	60
Firmware modification	30	40 (20 per setup)	10	60

approach. Table 4.2 summarizes the results. Overall, the SWAD method consistently achieves a high F1 score across all attack scenarios, with an overall F1 score of 96%, significantly outperforming the full-window approach in all attack scenarios.

Table 4.4: Summary of anomaly detection performance for SFW.

Device	Scenario	F1 score (SWAD)	F1 score (full-window)
SFW	SFW replacement	**95%**	68%
	Firmware version change	**100%**	14%
	Firmware modification	**94%**	48%
	Overall[2]	**96%**	50%

Analysis of Attack Scenarios

This section offers an in-depth analysis of SWAD's performance across the various attack scenarios.

- SFW Replacement: In this scenario, the SWAD method surpasses the full-window approach, achieving an F1 score of 95%. Figure 4.8 provides a visualization of a nominal trace, an anomalous trace for each anomaly setup, and the reference trace. We observe that the replacement with SFW04 introduces frequent spikes compared to the reference trace, while replacements with SFW02 and SFW03 cause noticeable localized deviation in earlier trace segments. The full-window method fails to detect replacements with SFW02 and SFW03, as other minor, long-lasting deviations overshadow the localized deviations.

[2]Note that this overall F1 score is calculated by first counting all the TP, FP, and FN in each attack scenario, and then applying the F1 score formula in Equation 4.1.

- Firmware Version Change: In this scenario, SWAD achieves an F1 score of 100%, whereas the full-window method only reached 14%. Figure 4.9 illustrates a nominal trace, an anomalous trace for each anomaly setup, and the reference trace, showing that switching to firmware version 4.5.4 introduces a level shift in the second half of the trace while switching to firmware version 4.4.0 results in relatively short-lived pattern deviations. These deviations are effectively captured by SWAD. However, the full-window approach misclassifies most anomalous cases because the nominal traces already exhibit relatively large variances, overshadowing the more localized deviations.

- Firmware Modification: In this scenario, SWAD achieves an F1 score of 94%, significantly outperforming the full-window approach, which has an F1 score of 48%. By examining an example of a nominal trace, an anomalous trace, and the reference trace in Figure 4.10, we observe that the firmware modifications introduce noticeable pattern changes in the boot power traces. SWAD effectively capture these changes, while the full-window approach struggles with the time delay injection change due to the relatively short-lived nature of the deviation.

Overall, these test cases demonstrate that SWAD is an effective method for detecting anomalies caused by hardware and firmware modifications. It offers significant performance improvements over the full-window method due to its ability to detect localized variations.

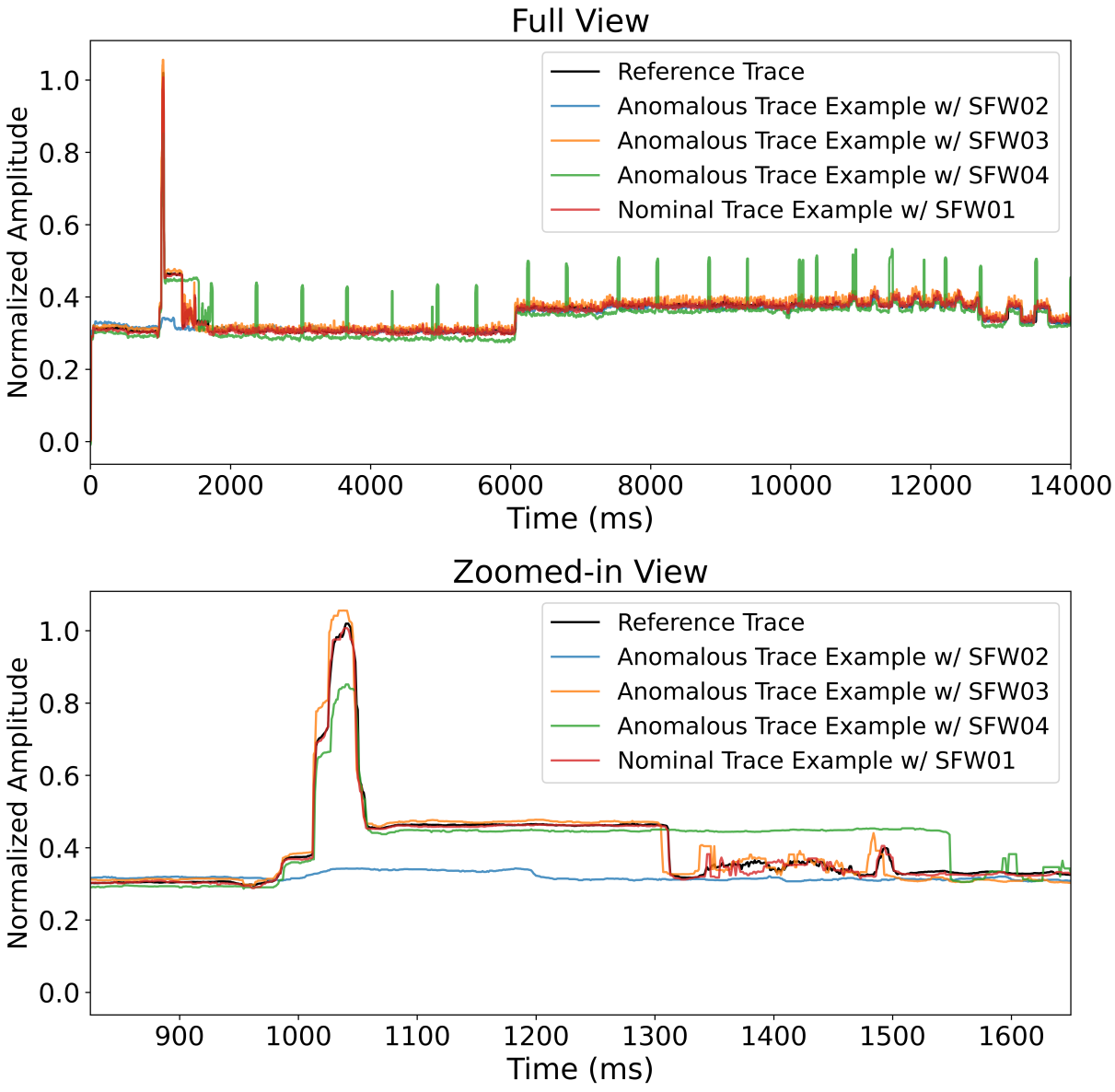


Figure 4.8: An illustration comparing a nominal trace, three anomalous traces, and the reference trace for the SFW replacement scenario in case studies on SFW, shown in the full window and a zoomed-in view.

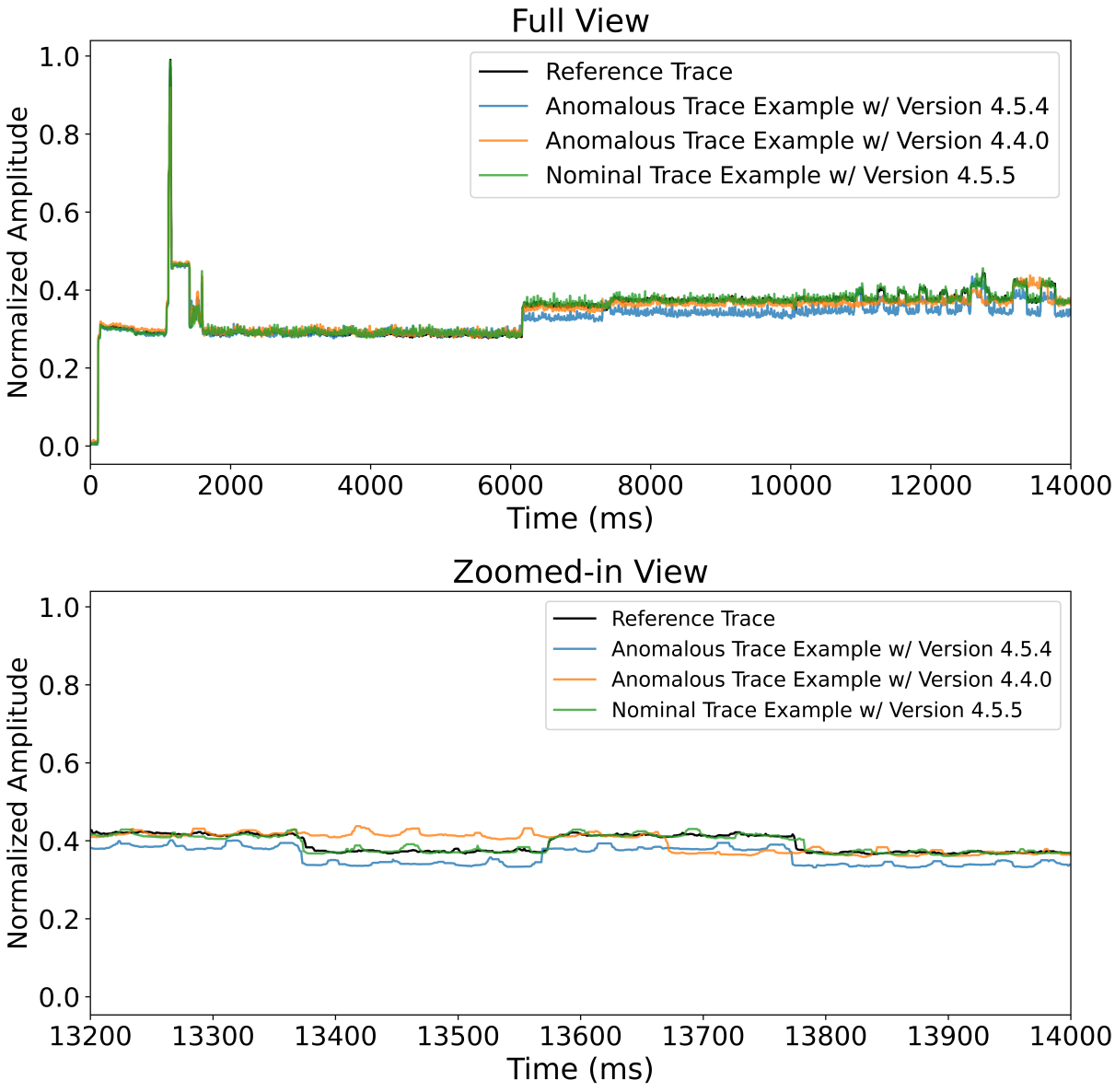


Figure 4.9: An illustration comparing a nominal trace, two anomalous traces, and the reference trace for the firmware version change scenario in case studies on SFW, shown in the full window and a zoomed-in view.

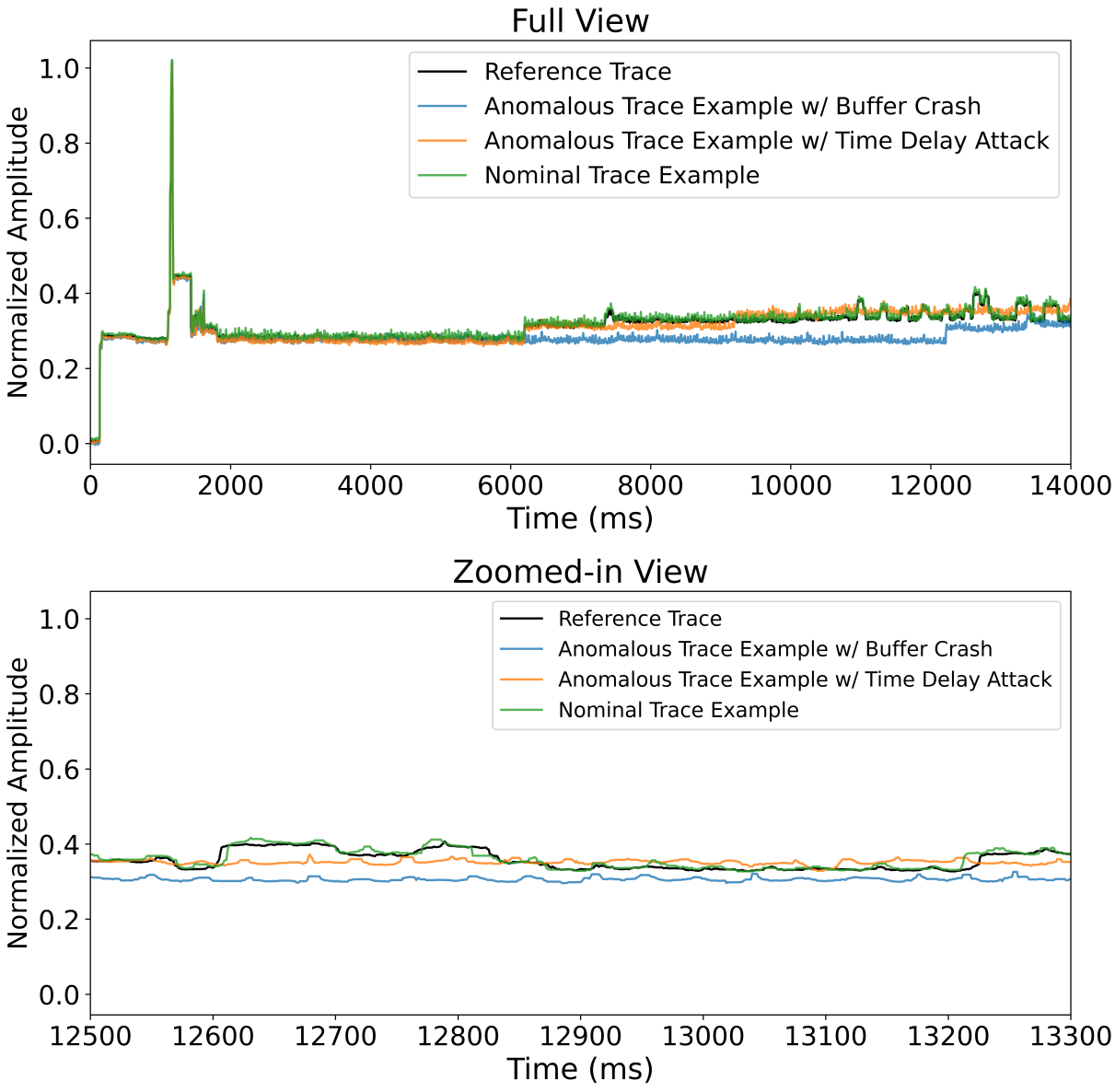


Figure 4.10: An illustration comparing a nominal trace, two anomalous traces, and the reference trace for the firmware modification scenario in case studies on SFW, shown in the full window and a zoomed-in view.



Figure 4.11: An example of a CPCO used in the case studies [69].

4.4 Case Study: Flight Controller – CubePilot Cube Orange+

This section examines the application of SWAD for detecting firmware anomalies in the CPCO (Figure 4.11). Similar to the SFW, the CPCO is a drone flight controller. The experiment conducted on the CPCO serves as a complement to the case study on the SFW.

4.4.1 Nominal Setup

The nominal setup for the CPCO serves as the baseline for detecting anomalies introduced by firmware attacks. This setup includes:

- Firmware Version: The CPCO runs ArduPilot Plane V4.5.5, a stable and widely-used firmware version [3].

4.4.2 Simulated Firmware Attacks

We simulate scenarios where the firmware running on the CPCO is modified. These scenarios are based on documented real-world firmware attacks discussed in Section 2.3.

- Firmware Version Change: We replace the original firmware with different versions, including ArduPilot Plane V4.5.4, ArduPilot Plane V4.4.4, and ArduPilot Plane

V4.4.0. These version changes may result in different boot power consumption behaviours due to factors such as differing initialization routines, bug fixes and optimizations, and security updates. This scenario evaluates whether SWAD can effectively detect such changes.

4.4.3 Experiment Setup

For the CPCO, we use the same experiment setup as BBB as in Section 4.2.4. We summarize the number of traces collected for each attack scenario, and the sizes of the training and test sets in Table 4.5.

Table 4.5: Experiment setup for CPCO: Number of nominal and anomalous traces collected for each attack scenario, along with the sizes of the training and test sets.

Scenario	Nbr. of Nominal Traces	Nbr. of Anomalous Traces	Training Set Size	Test Set Size
Firmware version change	30	60 (20 per setup)	10	80

4.4.4 Results

For the firmware change attack scenario, we apply SWAD to detect anomalous systems and evaluate their performance using the F1 score. SWAD achieves an 85% F1 score, significantly outperforming the full-window approach, which only reaches 51%. As shown in Figure 4.12, the differences in power traces between firmware versions V4.4.4, V4.5.4, and the reference trace are subtle and primarily localized, causing the full-window approach to struggle in detecting these changes. Firmware version V4.4.0 exhibits much more substantial pattern changes compared to the reference trace, and both methods ca detect this change.

Table 4.6: Summary of anomaly detection performance for CPCO.

Device	Scenario	F1 score (SWAD)	F1 score (full-window)
CPCO	Firmware version change	**85%**	51%

Overall, this attack scenario demonstrates the capability of SWAD to detect firmware anomalies in the CPCO, particularly highlighting its effectiveness in identifying localized deviations.

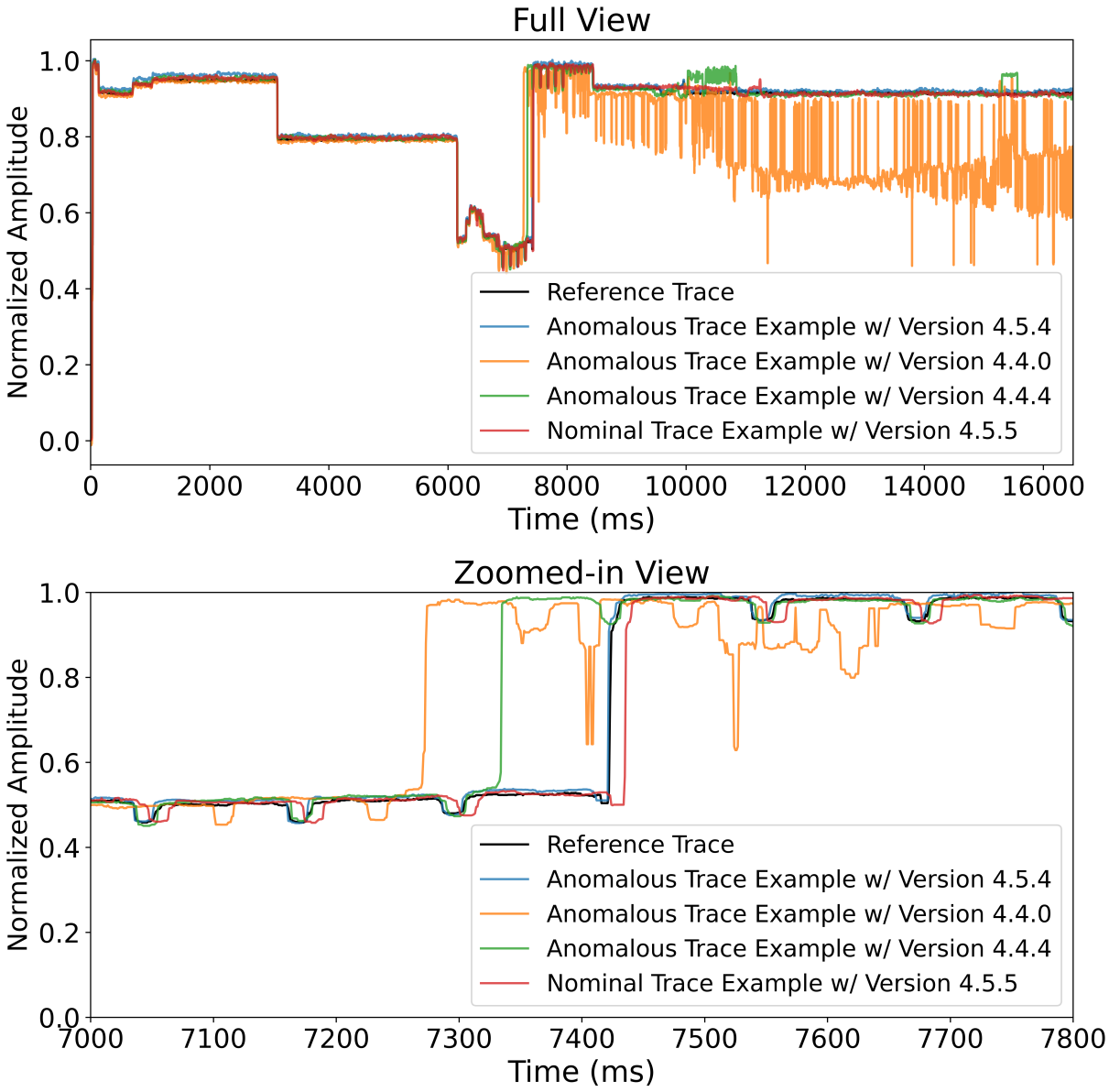


Figure 4.12: An illustration comparing a nominal trace, three anomalous traces, and the reference trace for the firmware version change scenario in case studies on CPCO, shown in the full window and a zoomed-in view.

4.5 Parameter Sensitivity Analysis

The performance of SWAD depends on several tunable parameters, including the PAA downsampling rate n, median filter kernel size s, sliding window size W, and window overlap ratio α. In our case studies, we adopt a default configuration with $n = 100$, $s = 5$, $W = 5\%$ of the boot power trace length, and $\alpha = 50\%$. To evaluate the robustness and sensitivity of each parameter, we conduct a series of experiments using the BBB firmware attack scenario as a representative case. In each experiment, one parameter is varied while the others are held fixed at their default values.

Our results show that SWAD is generally robust to a range of parameter values. Performance remains stable across a wide range of downsampling rates and median filter kernel sizes, with degradation only occurring under extreme smoothing. Similarly, sliding window size and overlap ratio influence detection accuracy, but the method performs well as long as the window size is not excessively large and a minimal overlap is maintained. These findings confirm that the selected parameter settings in our case studies are effective, though they can be further fine-tuned for specific system characteristics in practical deployments.

4.5.1 PAA Downsampling Rate

Table 4.7 summarizes the F1 scores of SWAD across different PAA downsampling rates n, tested for $n \in \{10, 25, 50, 100, 200, 400, 800, 1000\}$. Each value of n determines how many original samples are aggregated into a single downsampled point. Larger downsampling rates improve noise reduction and computational efficiency but risk discarding important signal details. As shown in the table, SWAD is generally robust to the choice of n, maintaining a 100% F1 score up to $n = 100$. However, performance begins to degrade at higher downsampling rates due to excessive smoothing, which reduces signal details. We select $n = 100$ in our case studies as it provides an effective balance between computational efficiency, noise reduction, and detection accuracy.

4.5.2 Median Filter Kernel Size

Table 4.8 shows the sensitivity of SWAD to different median filter kernel sizes s, where $s \in \{1, 3, 5, 7, 9, 21, 51, 101, 251, 501, 1001\}$. The kernel size determines the level of smoothing applied to the trace: larger values provide stronger noise reduction but risk removing important signal features. Only odd values are used to preserve symmetry around the center sample, which is standard for median filtering [2].

Table 4.7: Performance of SWAD across different PAA downsampling rates.

Downsampling Rate	F1 score
10	100%
25	100%
50	100%
100	100%
200	98.73%
400	93.33%
800	93.33%
1000	90.41%

As with the PAA downsampling rate, we observe that performance remains stable for small kernel sizes, but begins to degrade as s increases beyond 51. Excessive smoothing at large kernel sizes leads to loss of essential signal details, thereby affecting detection accuracy. We select $s = 5$ in our case studies as it offers a balance between noise suppression and detection accuracy.

Table 4.8: Performance of SWAD across different median filter kernel size.

Median Filter Kernel Size	F1 score
1	100%
3	100%
5	100%
7	100%
9	100%
21	100%
51	98.73%
101	96.10%
251	87.32%
501	90.41%
1000	88.88%

4.5.3 Sliding Window Size

We evaluate the impact of the sliding window size W on performance in Table 4.9. Since different systems have varying boot power trace lengths, we express W as a percentage of

the trace length. Specifically, we test $W \in \{1\%, 3\%, 5\%, 10\%, 20\%, 25\%, 50\%, 75\%, 100\%\}$ of the boot power trace.

Smaller window sizes can capture localized variations more effectively but may be sensitive to noise, while larger windows are more robust to noise yet less responsive to local anomalies. Our analysis reveals that detection accuracy begins to degrade when the window size exceeds 5%. Therefore, we select $W = 5\%$ in our case studies to balance between sensitivity and robustness.

Table 4.9: Performance of SWAD across different sliding window sizes (as a percentage of the power trace length).

Sliding Window Size (% of Boot Power Trace Size)	**F1 score**
1%	100%
3%	100%
5%	100%
10%	80%
20%	71.43%
25%	68.29%
50%	68.29%
75%	0%
100%	13.79%

4.5.4 Window Overlap Ratio

Table 4.10 summarizes the F1 scores of SWAD across different window overlap ratios $\alpha \in \{1\%, 3\%, 5\%, 10\%, 25\%, 50\%, 75\%, 99\%\}$, where α indicates the percentage of sample points each sliding window shares with the previous one. Higher overlap can improve sensitivity to localized anomalies but increases computational cost.

As shown in the table, SWAD maintains strong performance across a wide range of overlap ratios. The method is particularly robust even at relatively low values of α, with only a performance drop at 3%. This robustness is partly due to our default sliding window size of 5% of the power trace length, which is already small enough to capture localized anomalies effectively. In our case studies, we use a default overlap ratio of 50%, which provides a good balance between sensitivity and computational efficiency.

Table 4.10: Performance of SWAD across different window overlap ratio.

Window Overlap Ratio	F1 score
1%	100%
3%	74.42%
5%	100%
10%	100%
25%	100%
50%	100%
75%	100%
99%	100%

4.6 Real World Considerations

Our experiments simulate controlled attack scenarios where only one component is modified while others remain unchanged. However, real-world attackers are unlikely to restrict themselves to a single modification. For example, in the BBB replacement scenario, an attacker might introduce firmware changes rather than keeping the firmware unchanged. Such combined modifications would result in more significant variations in boot power trace, making them easier to detect. Therefore, our proposed SWAD method is evaluated on a more conservative basis and remains applicable and effective for identifying anomalies in practical settings.

4.7 Detection Limits of SWAD

In this section, we analyze the detection limits of SWAD to determine the extent of deviation it can reliably identify. We assess the algorithm's sensitivity from two key perspectives:

1. **Aggregated Deviation:** SWAD detects anomalies by computing an anomaly score for each window of a trace, aggregating the pointwise deviations from the reference trace. Although this score does not separately quantify the duration or magnitude of deviations, it is directly used to determine if an anomaly exists. This makes it a practical and accurate reflection of the algorithm's actual detection limits. To further illustrate these limits, we will employ a separation plot for visualization.

2. **Theoretical Duration and Amplitude Deviation:** A more intuitive approach is to analyze detection limits regarding the duration and magnitude of deviations

required for SWAD to flag an anomaly. However, real-world power traces exhibit inherent random noise at each time point, causing deviations throughout the entire trace with varying magnitudes. To provide a more precise analysis, we theoretically evaluate the scenario in which a given power trace deviates from the reference trace for various lengths of brief segments. Specifically, we determine the maximum average deviation magnitude that can persist over these short segments before being flagged by SWAD. We will employ a residual power envelope plot to visualize the detection limits.

Together, these perspectives offer insights into the detection limits of SWAD. We explore them in detail in the following sections, using the BBB firmware version change attack scenario as an example.

4.7.1 Aggregated Deviation

Our SWAD algorithm flags a trace as anomalous if its anomaly score, calculated as the aggregated absolute pointwise deviation from the reference trace, exceeds a threshold in any window. Thus, the detection limit of the algorithm is determined by the threshold value for each window. Figure 4.13 illustrates this concept by showing the distribution of the anomaly scores in Window 5 from the BBB firmware version change attack scenario. Each data point represents the anomaly score of a single trace from the test set, while the vertical red dotted line indicates the detection threshold. Traces with scores to the left of this line are considered nominal, whereas those to the right are flagged as anomalies. In this plot, we present aggregated anomaly scores for nominal system traces and for anomalous traces from four different firmware versions. We observe a large gap between the scores of nominal and anomalous traces, with the latter greatly exceeding the detection threshold. This demonstrates that this window effectively separates nominal and anomalous behaviour.

4.7.2 Theoretical Duration and Amplitude Deviation

While the aggregated deviation provides a direct measure of SWAD's detection limits, we further examine the duration and magnitude of deviations that lead to flagged anomalies. Consider a scenario in which a given trace deviates from the reference trace within a given window. We define two key metrics:

- Anomaly Duration (L_a): The number of samples over which the given trace deviates from the reference trace.

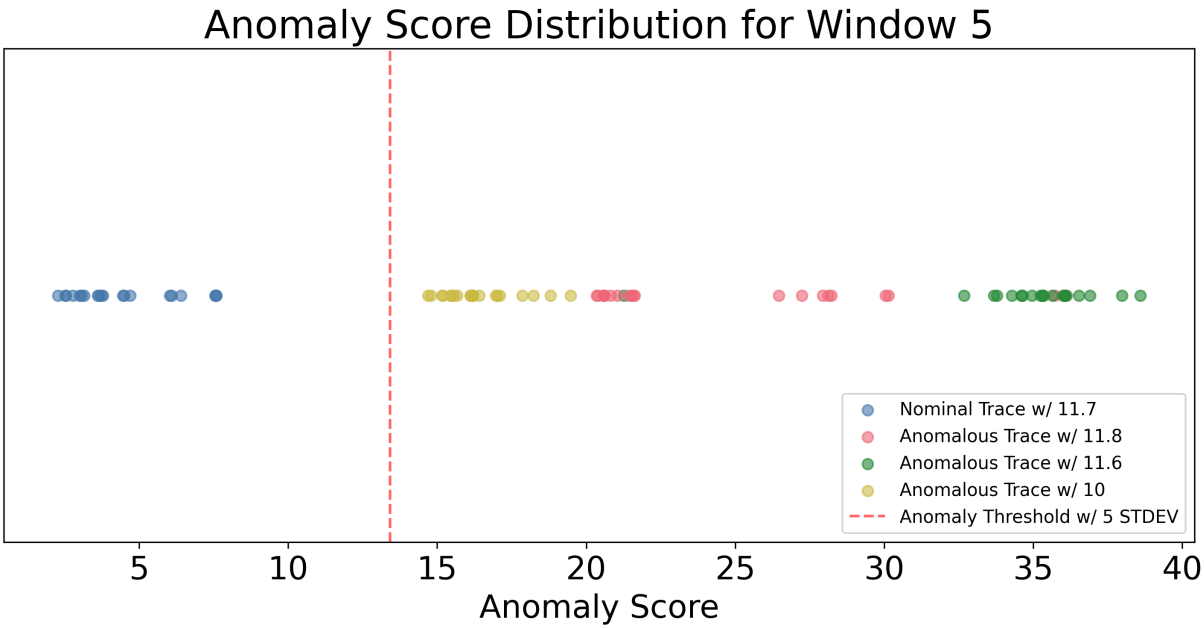


Figure 4.13: Distribution of anomaly scores for test set traces in the BBB firmware version change scenario in Window 5. Each point represents the anomaly score of a test trace, with the red line indicating the anomaly threshold.

- Anomaly Magnitude (D): The average deviation of the given trace from the reference trace over the anomaly duration L_a.

Given the anomaly threshold Threshold^k for window k, the boundary average deviation for a given anomaly duration is defined as:

$$D_{\text{boundary}} = \frac{\text{Threshold}^k}{L_a}. \tag{4.2}$$

Using Window 5 from the BBB firmware version change attack scenario as an example, we plot the boundary average deviation across different anomaly durations, forming what we refer to as the residual power envelope. This envelope determines the maximum allowable duration and corresponding average deviation before a trace is flagged as anomalous; points above the envelope are identified as anomalous, while those below are considered nominal.

In practice, power traces contain inherent random noise; consequently, both nominal and anomalous traces exhibit an anomaly duration close to or equal to the full window length. In such cases, plotting the boundary average deviation in the residual power envelope essentially reduces to the anomaly distribution score plot shown in Figure 4.13—with the difference that Figure 4.13 depicts the aggregated deviation from the reference trace, whereas the residual power envelope plot displays the average deviation.

To enhance the effectiveness and interpretability of the residual power envelope analysis, we modify the definition of L_a by counting only those samples that are considered significantly abnormal. Specifically, for each sample index, we compute the Manhattan distance of the training set traces from the reference trace and calculate the standard deviation of these distances. A sample in a test trace is deemed significantly abnormal if its distance from the reference trace exceeds the 90th percentile of the nominal traces' distances. As illustrated in Figure 4.14, the blue shaded region represents the range where a sample is not considered significantly abnormal and not counted into L_a. This approach allows us to focus solely on relatively large deviations to make the power envelope analysis more meaningful.

Figure 4.15 demonstrates the residual power envelope for the scenario we demonstrate in Figure 4.14. Figure 4.16 presents the resulting residual power envelope with the distribution of all the traces in the test set of BBB firmware version change scenario in Window 5. Each data point represents a test trace. Traces located near the upper-right corner indicate substantial deviation from the reference trace, with longer anomaly durations and higher anomaly magnitudes. In contrast, traces near the lower-left corner exhibit minimal deviation, with fewer anomalous points and lower average scores. Thus, the residual power envelope effectively characterizes the detection limits of SWAD, with traces closer to the envelope being more prone to misclassifications.

There are similar concepts to the residual power envelope in the literature, often referred to as the detection envelope, sensitivity envelope, or residual threshold envelope [68]. Similar to our use case, the envelope concept is commonly employed to:

- Quantify the range of residual values that are considered normal.
- Establish detection thresholds that balance sensitivity and specificity.
- Understand how subtle changes in system behaviour may evade detection.

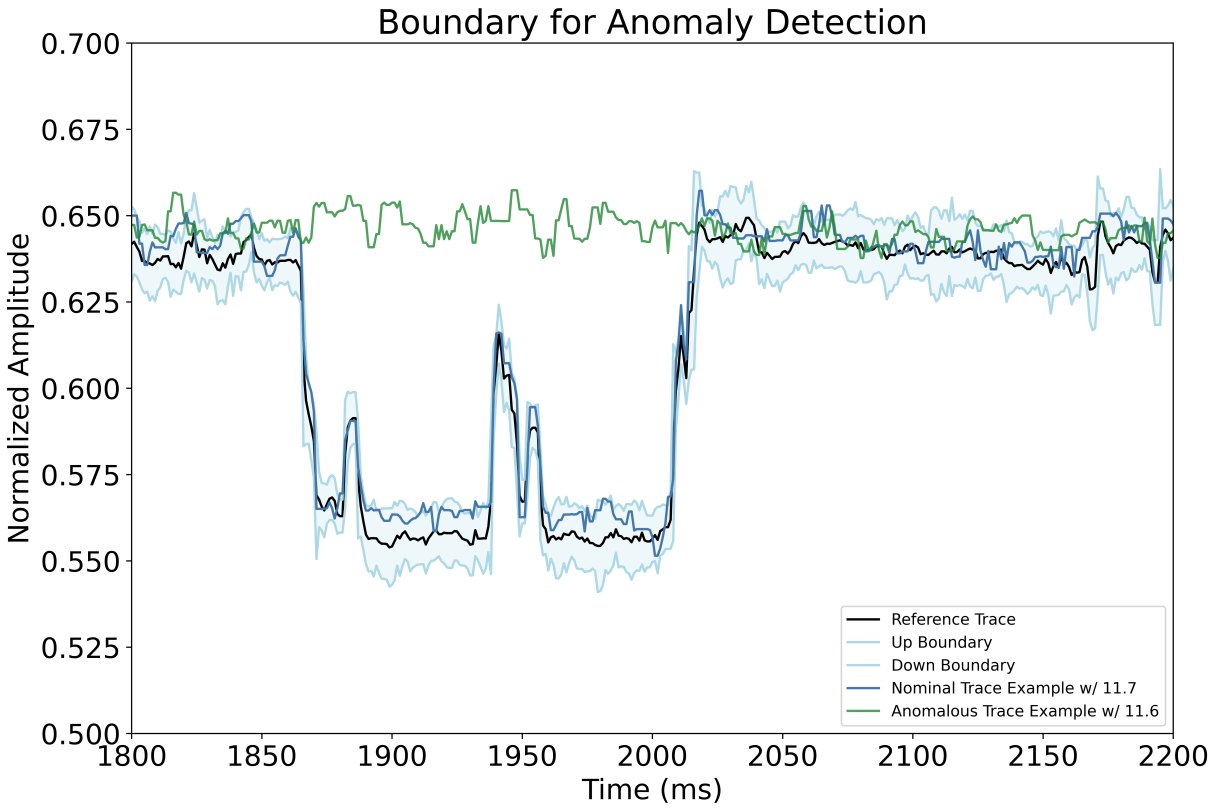


Figure 4.14: An illustration of the range within which sample points are considered normal, with points outside the range deemed significantly abnormal and included in L_a, alongside a nominal trace and an anomalous trace for comparison.

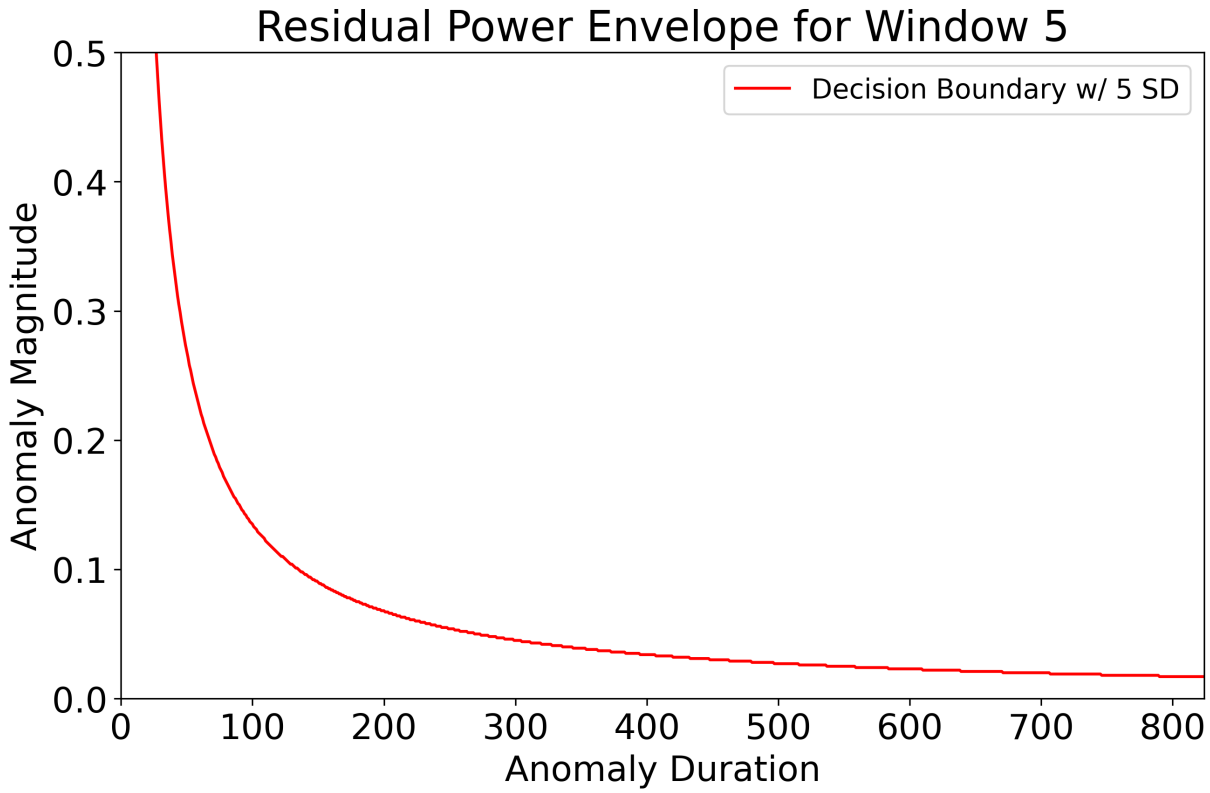


Figure 4.15: An illustration of the residual power envelope for BBB that determines the maximum allowable duration and corresponding average deviation before a trace is flagged as anomalous.

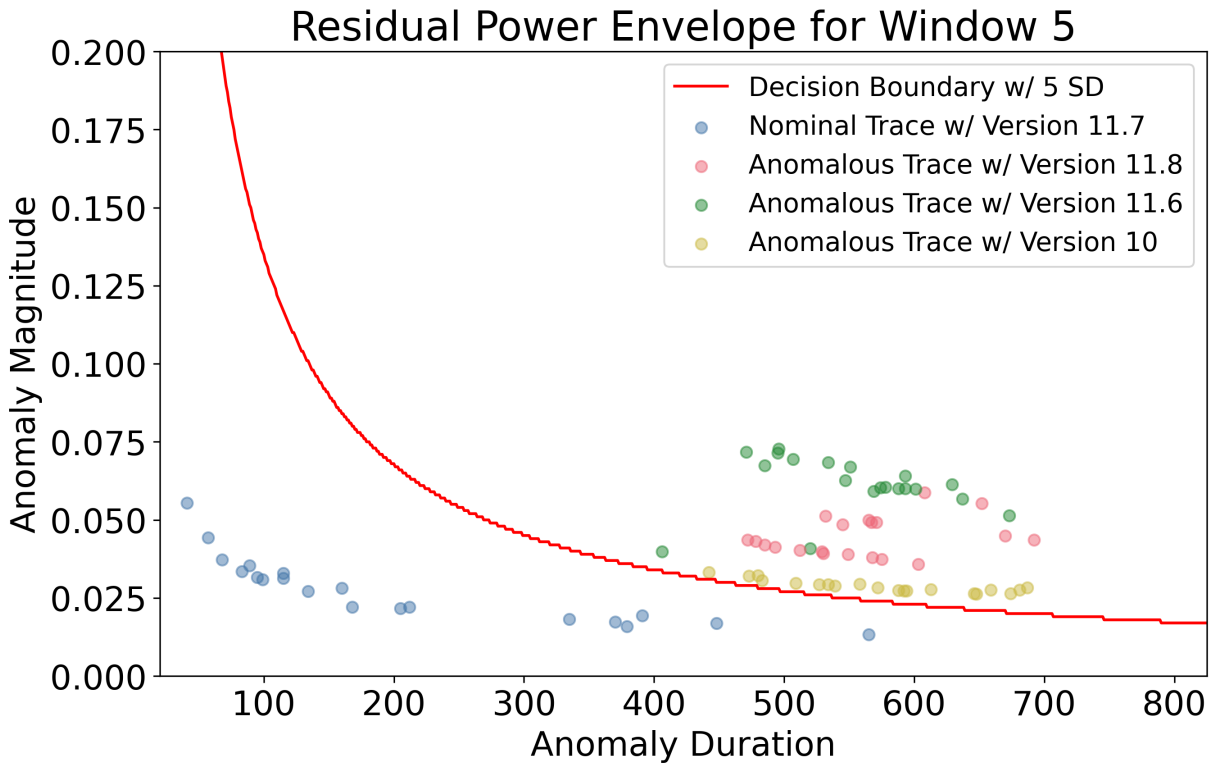


Figure 4.16: An illustration of the residual power envelope for BBB, where each point corresponds to a test set trace in the BBB firmware version change scenario in Window 5. Points above the envelope are considered anomalous, while points below are considered nominal.

4.7.3 False Positive and False Negative Tradeoff

Next, we discuss the tradeoff between false positive and false negative misclassifications by leveraging the anomaly score distribution and residual power envelope plots. The anomaly threshold used to distinguish between nominal and anomalous traces in window k is computed as in Equation 3.4. This threshold is adjustable via the number of standard deviations: a higher multiplier yields a wider threshold, reducing the likelihood of flagging nominal traces as anomalous at the risk of missing some anomalies, while a lower multiplier narrows the threshold, increasing sensitivity but with the risk of false alarms.

To illustrate this tradeoff, we define two alternative thresholds:

$$\text{Threshold}^k_{\text{narrow}} = \mu(A^k) + 1 \cdot \sigma(A^k), \tag{4.3}$$

$$\text{Threshold}^k_{\text{wide}} = \mu(A^k) + 9 \cdot \sigma(A^k), \tag{4.4}$$

where $\mu(A^k)$ and $\sigma(A^k)$ are the mean and standard deviation of the anomaly scores in window k, respectively.

Using BBB firmware version change attack scenario with Window 5 as an example. Figure 4.17 and Figure 4.18 illustrate the impact of these thresholds on the anomaly score distribution and the residual power envelope. With $\text{Threshold}^k_{\text{narrow}}$, several nominal traces are misclassified as anomalous. In contrast, $\text{Threshold}^k_{\text{wide}}$ is insufficiently sensitive, resulting in some anomalous traces—such as those corresponding to firmware version 4.4.4—not being detected. These results underscore the inherent tradeoff between false positives and false negatives. In practice, users can adjust the threshold according to their specific needs, using these plots as a guide to balance false positives and false negatives.

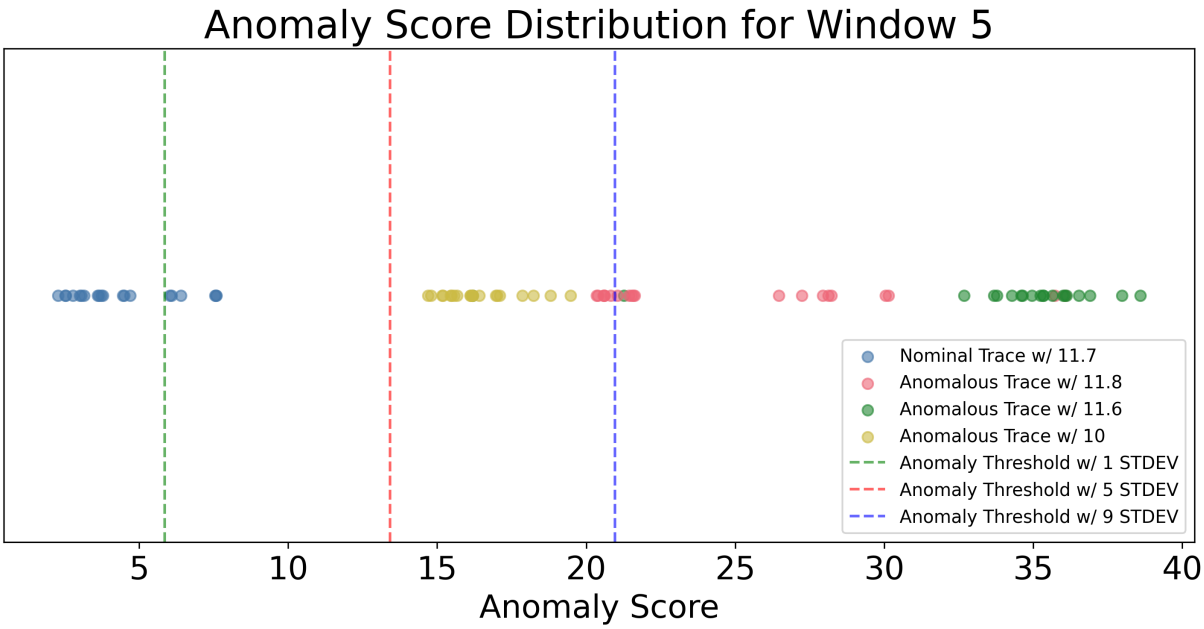


Figure 4.17: An illustration of three different anomaly thresholds, along with the distribution of anomaly scores for test set traces in the BBB firmware version change scenario in Window 5.

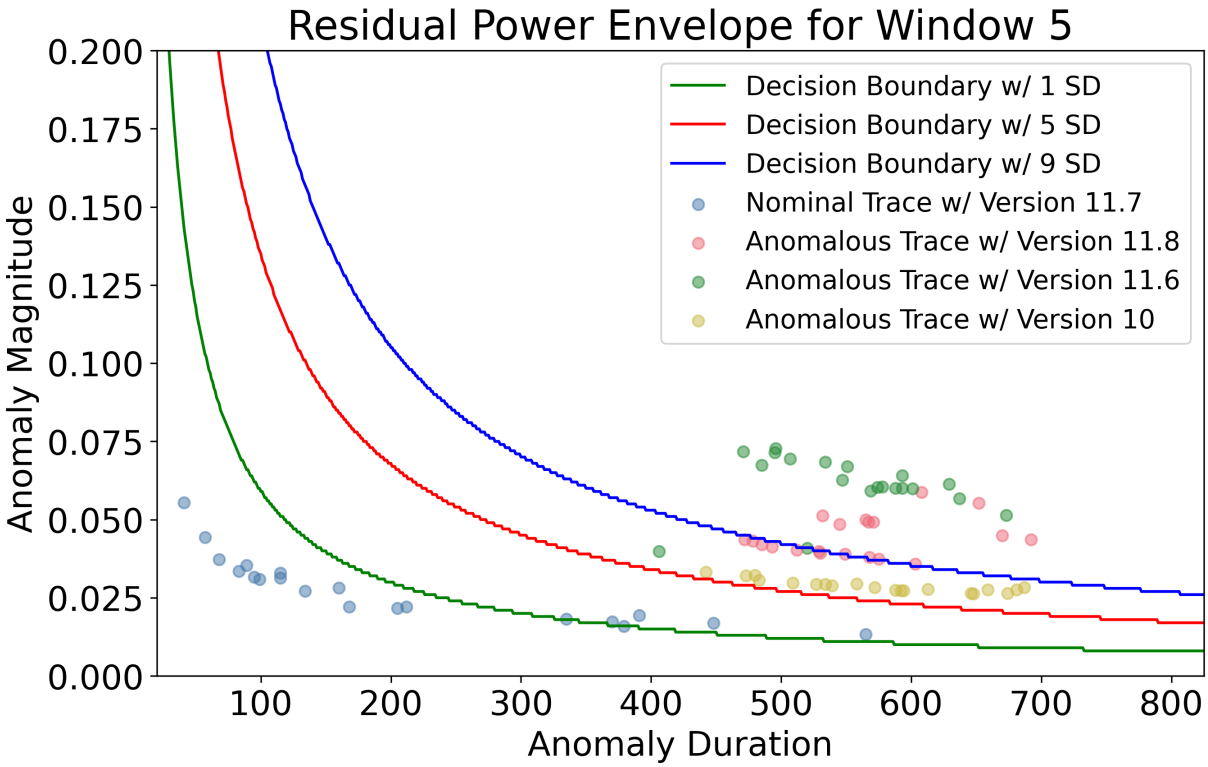


Figure 4.18: An illustration of three power envelopes, along with the distribution of test set traces in the BBB firmware version change scenario in Window 5.

Chapter 5

Discussion

5.1 Boot Process Duration Determination

In the assumption stated in Section 3.1, we assume that the duration of the boot process is known a priori. In practice, however, this duration must be empirically determined based on observable criteria.

In our research, we employed two complementary methods to estimate the end of the boot process. The first method involved visual observation. By monitoring the system's output, such as the display, console messages, or LED status indicators, we identified the point at which the system transitioned to a stable, ready state.

The second method was based on trace-based analysis. We collected extended-duration power traces over multiple boot cycles and analyzed the variance across these traces. When the power behavior became highly variable or unstable across traces, which we interpreted as an indication of post-boot activities, we considered the preceding point as the end of the boot process. This approach allowed us to determine a consistent cutoff for capturing the boot phase while excluding unrelated runtime fluctuations.

5.2 Distance Metric Selection

In our anomaly detection framework, we use the Manhattan distance to calculate the anomaly score of each traces. This choice is motivated by several practical considerations.

First, Manhattan distance is less sensitive to large individual deviations compared to the Euclidean distance, which amplifies outliers due to the squared error term. Since power traces can contain occasional spikes or transient noise, the Manhattan distance offers a more stable and robust measure. It is also particularly effective at capturing small but long-lasting deviations, which may be diluted with Euclidean distance.

Second, Manhattan distance provides a more interpretable anomaly score. Because it directly sums absolute differences in power at each time point, the resulting value has a clear physical meaning—it represents the total deviation in power usage across the window. This contrasts with Euclidean distance, whose squared terms can distort scale and obscure interpretation.

Therefore, we choose Manhattan distance for its ability to balance detection sensitivity, robustness to noise, and the interpretability of results in the context of power-based anomaly detection.

Chapter 6

Conclusion

This study proposed using boot power traces to detect hardware and firmware anomalies in embedded systems. Our approach identifies potential security breaches by comparing unlabeled traces against nominal profiles. We introduced SWAD, a novel sliding window anomaly detection method, which improves upon a full-window approach by capturing transient irregularities and adapting to local variations within power traces.

We evaluated SWAD across a range of simulated real-world hardware and firmware attack scenarios, using the full-window approach as a benchmark. SWAD achieved F1 scores of 98%, 96%, and 85% for attacks on BBB, SFW, and CPCO, respectively, outperforming the full-window approach, which yielded 92%, 50%, and 51% in the same scenarios. These results confirm the practical viability of SWAD across diverse platforms and attack types. In addition, this power side-channel method is cost-efficient, as it relies on a generic power monitoring device that can be applied across different systems without requiring direct access to internal components. Moreover, the reference trace-based approach used in SWAD provides advantages in explainability, simplicity, and robustness, making SWAD well-suited for real-world deployment.

Despite these compelling features, SWAD has certain limitations. While the default anomaly detection threshold defined in Section 3.4 is generally effective across the analyzed attack scenarios, it may not be optimal for all datasets or operating conditions. Fine-tuning is often necessary to balance false positives and false negatives, especially with noisy or highly variable power traces. Additionally, parameters such as downsampling rate, median-filter kernel size, sliding window size and window overlap ratio influence performance and should be carefully tuned in practical deployments. Addressing these challenges through adaptive parameter tuning can be an area of future research. Moreover,

the threshold currently used to identify the start of the boot process in Section 3.6.2 is determined empirically and could be improved through automation to reduce reliance on manual analysis.

Finally, our study evaluated SWAD on three embedded systems. Future work can extend this research by exploring a broader range of embedded platforms and attack scenarios to validate the method's effectiveness further.

References

[1] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM Side—Channel(s). In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 29–45, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[2] Obed Appiah, Ezekiel Mensah Martey, and Ebenezer Quayson. Effect of Window's Shape on Median Filtering. In *2019 IEEE AFRICON*, pages 1–8, 2019.

[3] ArduPilot Firmware. SpeedyBee F405 Wing Firmware Versions, 2025. Accessed: 2025-03-17.

[4] BeagleBoard.org. BeagleBone Black U-Boot Repository, 2022. Accessed: 2025-03-17.

[5] BeagleBoard.org. BeagleBone Black Board Image, 2023. Accessed: 2025-03-17.

[6] BeagleBoard.org. BeagleBone Black Firmware Version, 2023. Accessed: 2025-03-17.

[7] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly Detection: A Survey. *ACM Comput. Surv.*, 41(3), July 2009.

[8] Peter Christen, David J. Hand, and Nishadi Kirielle. A Review of the F-Measure: Its History, Properties, Criticism, and Alternatives. *ACM Comput. Surv.*, 56(3), October 2023.

[9] Shane Clark, Hossen Mustafa, Benjamin Ransford, Jacob Sorber, Kevin Fu, and Wenyuan Xu. Current Events: Identifying Webpages by Tapping the Electrical Outlet. pages 700–717, 09 2013.

[10] Shane S Clark, Benjamin Ransford, Amir Rahmati, Shane Guineau, Jacob Sorber, Wenyuan Xu, and Kevin Fu. WattsUpDoc: Power Side Channels to Nonintrusively

Discover Untargeted Malware on Embedded Medical Devices. In *2013 USENIX Workshop on Health Information Technologies (HealthTech 13)*, 2013.

[11] Mauro Conti, Michele Nati, Enrico Rotundo, and Riccardo Spolaor. Mind The Plug! Laptop-User Recognition Through Power Consumption. pages 37–44, 05 2016.

[12] Andrew A. Cook, Goksel Misirli, and Zhong Fan. Anomaly Detection for IoT Time-Series Data: A Survey. *IEEE Internet of Things Journal*, 7(7):6481–6494, 2020.

[13] Chris DeProfio. Firmware vs Embedded Software - What's the difference? *AndPlus Blog*, September 2019.

[14] A. Adam Ding, Liwei Zhang, Yunsi Fei, and Pei Luo. A Statistical Model for Higher Order DPA on Masked Devices. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems – CHES 2014*, pages 147–169, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[15] Murray Dunne and Sebastian Fischmeister. Powertrace-based Fuzzing of CAN Connected Hardware. In *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 239–244, 2022.

[16] Václav Dvorák. Embedded Firmware Development with Multi-way Branching. In *Third International Conference on Systems (icons 2008)*, pages 317–322, 2008.

[17] Emad Ebeid, Martin Skriver, and Jie Jin. A Survey on Open-Source Flight Control Platforms of Unmanned Aerial Vehicle. In *2017 Euromicro Conference on Digital System Design (DSD)*, pages 396–402, 2017.

[18] Adam Entous and Michael Schwirtz. The C.I.A.'s Blind Spot About Ukraine. *The New York Times*, February 2024. Accessed: 2025-03-08.

[19] Corin Faife. The new USB Rubber Ducky is more dangerous than ever. *The Verge*, 2022. Accessed: 2025-03-08.

[20] Pavlo Galkin, Lydmila Golovkina, and Igor Klyuchnyk. Analysis of Single-Board Computers for IoT and IIoT Solutions in Embedded Control Systems. In *2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC ST)*, pages 297–302, 2018.

[21] Daniel Genkin, Adi Shamir, and Eran Tromer. Acoustic Cryptanalysis. *J. Cryptol.*, 30(2):392–443, April 2017.

[22] Carlos R. Aguayo Gonzalez and Jeffrey H. Reed. Power fingerprinting in SDR & CR integrity assessment. In *MILCOM 2009 - 2009 IEEE Military Communications Conference*, pages 1–7, 2009.

[23] Arthur Grisel-Davy, Amrita Milan Bhogayata, Srijan Pabbi, Apurva Narayan, and Sebastian Fischmeister. Work-in-Progress: Boot Sequence Integrity Verification with Power Analysis. In *2022 International Conference on Embedded Software (EMSOFT)*, pages 3–4, 2022.

[24] Arthur Grisel-Davy and Sebastian Fischmeister. Independent Boot Process Verification using Side-Channel Power Analysis. In *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security Companion (QRS-C)*, pages 196–207, 2023.

[25] Tiantian Guo, Tongpo Zhang, Enggee Lim, Miguel López-Benítez, Fei Ma, and Limin Yu. A Review of Wavelet Analysis and Its Applications: Challenges and Opportunities. *IEEE Access*, 10:58869–58903, 2022.

[26] Gareth Halfacree. Aedan Cullen Cracks the Raspberry Pi RP2350's Security Subsystem—Wide Open, 2025. Accessed: March 31, 2025.

[27] Yi Han, Matthew Chan, Zahra Aref, Nils Ole Tippenhauer, and Saman Zonouz. Hiding in Plain Sight? On the Efficacy of Power Side Channel-Based Control Flow Monitoring. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 661–678, Boston, MA, August 2022. USENIX Association.

[28] Intel Corporation. Intel® Software Guard Extensions (Intel® SGX), 2024. Accessed: 2025-02-14.

[29] Nisha Jacob, Johann Heyszl, Andreas Zankl, Carsten Rolfes, and Georg Sigl. How to Break Secure Boot on FPGA SoCs through Malicious Hardware. Cryptology ePrint Archive, Paper 2017/625, 2017.

[30] Jarilyn M. Hernández Jiménez, Jeffrey A. Nichols, Katerina Goseva-Popstojanova, Stacy Prowell, and Robert A. Bridges. Malware Detection on General-Purpose Computers Using Power Consumption Monitoring: A Proof of Concept and Case Study, 2017.

[31] Chen Jin, Ryutaro Tanno, Thomy Mertzanidou, Eleftheria Panagiotaki, and Daniel C. Alexander. Learning to Downsample for Segmentation of Ultra-High Resolution Images. *CoRR*, abs/2109.11071, 2021.

[32] Rong Jin and Kai Zeng. Physical layer key agreement under signal injection attacks. In *2015 IEEE Conference on Communications and Network Security (CNS)*, pages 254–262, 2015.

[33] Jatin Kataria, Rick Housley, Joseph Pantoga, and Ang Cui. Defeating Cisco Trust Anchor: A Case-Study of Recent Advancements in Direct FPGA Bitstream Manipulation. In *13th USENIX Workshop on Offensive Technologies (WOOT 19)*, Santa Clara, CA, August 2019. USENIX Association.

[34] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. *Knowledge and information systems*, 3(3):263–286, 2001.

[35] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. *Knowledge and Information Systems*, 3, 01 2002.

[36] Dinesh Kithany. Challenges and Innovations in Medical Power Supply Design. *WAWT Tech*, August 2024. Accessed: 2025-03-17.

[37] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 104–113, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[38] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, page 388–397, Berlin, Heidelberg, 1999. Springer-Verlag.

[39] Deepak Krishnankutty, Zheng Li, Ryan Robucci, Nilanjan Banerjee, and Chintan Patel. Instruction Sequence Identification and Disassembly Using Power Supply Side-Channel Analysis. *IEEE Transactions on Computers*, 69(11):1639–1653, 2020.

[40] Michael LeMay and Carl A. Gunter. Cumulative Attestation Kernels for Embedded Systems. *IEEE Transactions on Smart Grid*, 3(2):744–760, 2012.

[41] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. A Symbolic Representation of Time Series, with Implications for Streaming Algorithms. pages 2–11, 06 2003.

[42] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. PLATYPUS: Software-based Power Side-Channel

Attacks on x86. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 355–371, 2021.

[43] Christopher Liptak, Sanchita Mal-Sarkar, and Sathish A.P. Kumar. Power Analysis Side Channel Attacks and Countermeasures for the Internet of Things. In *2022 IEEE Physical Assurance and Inspection of Electronics (PAINE)*, pages 1–7, 2022.

[44] Zheyuan Ma, Xi Tan, Lukasz Ziarek, Ning Zhang, Hongxin Hu, and Ziming Zhao. Return-to-Non-Secure Vulnerabilities on ARM Cortex-M TrustZone: Attack and Defense. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2023.

[45] Andrzej Maćkiewicz and Waldemar Ratajczak. Principal components analysis (PCA). *Computers Geosciences*, 19(3):303–342, 1993.

[46] Yan Michalevsky, Gabi Nakibly, Aaron Schulman, Gunaa Arumugam Veerapandian, and Dan Boneh. PowerSpy: Location Tracking using Mobile Device Power Analysis, 2015.

[47] Microchip Technology Inc. MIPS32® microAptiv™ and M5150 Core Instruction Set Architecture (ISA) OverviewToggle the hierarchy tree under MIPS32® microAptiv™ and M5150 Core Instruction Set Architecture (ISA) Overview. https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/32bit-mcu/PIC32/mz-arch-cpu-overview/isa-overview/, 2023. Accessed: 2025-04-12.

[48] Microsoft. Trusted Platform Module Technology Overview, 2024. Accessed: 2025-02-14.

[49] Microsoft Security Response Center. CVE-2022-21894 - Windows Boot Manager Security Feature Bypass Vulnerability, 2022. Accessed: 2025-02-19.

[50] Jyotiprakash Mishra and Sanjay K. Sahay. Modern Hardware Security: A Review of Attacks and Countermeasures, 2025.

[51] MITRE. CAPEC-401: Hardware Integrity Attack, 2018. Accessed: 2025-02-20.

[52] MITRE. CAPEC-440: Hardware Integrity Attack, 2018. Accessed: 2025-02-20.

[53] MITRE. CAPEC-532: WebView Injection, 2018. Accessed: 2025-02-20.

[54] MITRE. CVE-2019-19642, 2019. Accessed: 2025-03-08.

[55] MITRE. CVE-2024-20456, 2024. Accessed: 2025-03-08.

[56] MITRE. CVE-2024-38279, 2024. Accessed: 2025-03-08.

[57] MITRE. CVE-2024-8105, 2024. Accessed: 2025-03-08.

[58] MITRE. CWE-1247: Improper Protection Against Voltage and Clock Glitches, 2024. Accessed: 2024-04-02.

[59] MITRE. CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition, 2024.

[60] MITRE. Pre-OS Boot: System Firmware, 2024. Accessed: 2025-02-20.

[61] Carlos Moreno, Sean Kauffman, and Sebastian Fischmeister. Efficient program tracing and monitoring through power consumption - with a little help from the compiler. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1556–1561, 2016.

[62] Mehari Msgna, Konstantinos Markantonakis, David Naccache, and Keith Mayes. Verifying Software Integrity in Embedded Systems: A Side Channel Approach. 04 2014.

[63] Maria Mushtaq, Muhammad Asim Mukhtar, Vianney Lapotre, Muhammad Khurram Bhatti, and Guy Gogniat. Winter is here! A decade of cache-based side-channel attacks, detection mitigation for RSA. *Information Systems*, 92:101524, 2020.

[64] Sung Dae Na, Qun Wei, Ki Woong Seong, Jin Ho Cho, Myoung Nam Kim, Carlos Gómez, Severin P. Schwarzacher, and Huiyu Zhou. Noise reduction algorithm with the soft thresholding based on the Shannon entropy and bone-conduction speech cross-correlation bands. *Technol. Health Care*, 26(S1):281–289, January 2018.

[65] Luong Nguyen, Chia-Lin Cheng, Frank Werner, Milos Prvulovic, and A. Zajic. A Comparison of Backscattering, EM, and Power Side-Channels and Their Performance in Detecting Software and Hardware Intrusions. *Journal of Hardware and Systems Security*, 4, 06 2020.

[66] Yekini Nureni. OVERVIEW OF EMBEDDED SYSTEM ITS APPLICATION. In *3rd International Academic Conference Organized by Academic Staff Union of Polytechnics IREE Chapter*, 06 2022.

[67] P.R. Perez-Alcazar and A. Santos. Relationship between sampling rate and quantization noise. In *2002 14th International Conference on Digital Signal Processing Proceedings. DSP 2002 (Cat. No.02TH8628)*, volume 2, pages 807–810 vol.2, 2002.

[68] Masoud Pourasghar, Anh-Tu Nguyen, and Thierry-Marie Guerra. Guaranteed State Estimation for H/L Fault Detection of Uncertain Takagi-Sugeno Fuzzy Systems with Unmeasured Nonlinear Consequents. *IEEE Transactions on Fuzzy Systems*, PP, 01 2025.

[69] PX4 Documentation. CubePilot Cube Orange+ Image, 2023. Accessed: 2025-03-17.

[70] Ajitha Rajan, Adel Noureddine, and Panagiotis Stratis. A Study on the Influence of Software and Hardware Features on Program Energy. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '16, New York, NY, USA, 2016. Association for Computing Machinery.

[71] John G. Saw, Mark C. K. Yang, and Tse Chin Mo. Chebyshev Inequality with Estimated Mean and Variance. *The American Statistician*, 38(2):130–132, 1984.

[72] Sebastian Schmidl, Phillip Wenig, and Thorsten Papenbrock. Anomaly detection in time series: a comprehensive evaluation. *Proc. VLDB Endow.*, 15(9):1779–1797, May 2022.

[73] C. O. S. Sorzano, J. Vargas, and A. Pascual Montano. A survey of dimensionality reduction techniques, 2014.

[74] SP Devices. ADQ32 Digitizer Image, 2025. Accessed: 2025-03-17.

[75] SpeedyBee. SpeedyBee F405 Wing Image, 2023. Accessed: 2025-03-17.

[76] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1057–1074, Vancouver, BC, August 2017. USENIX Association.

[77] Mohammad Tehranipoor, Hassan Salmani, and Xuehui Zhang. *Hardware Trojan Detection: Untrusted Manufactured Integrated Circuits*, pages 31–38. Springer International Publishing, Cham, 2014.

[78] Trustonic. What is TrustZone?, 2019. Accessed: 2025-02-14.

[79] Jean-Philippe Vasseur and Adam Dunkels. Chapter 1 - What Are Smart Objects? In Jean-Philippe Vasseur and Adam Dunkels, editors, *Interconnecting Smart Objects with IP*, pages 3–20. Morgan Kaufmann, Boston, 2010.

[80] Lin Wang, Faming Lu, Minghao Cui, and Yunxia Bao. Survey of Methods for Time Series Symbolic Aggregate Approximation. In Xiaohui Cheng, Weipeng Jing, Xianhua Song, and Zeguang Lu, editors, *Data Science*, pages 645–657, Singapore, 2019. Springer Singapore.

[81] Rui Wang and Yonghang Yan. A Survey of Secure Boot Schemes for Embedded Devices. In *2022 24th International Conference on Advanced Communication Technology (ICACT)*, pages 224–227, 2022.

[82] Jinpeng Wei and Calton Pu. TOCTTOU vulnerabilities in UNIX-style file systems: an anatomical study. In *Proceedings of the 4th Conference on USENIX Conference on File and Storage Technologies - Volume 4*, FAST'05, page 12, USA, 2005. USENIX Association.

[83] Nicola Williams. Internet of Things, Semiconductors, and Organic Electronics, 2025. Accessed: 2025-03-17.

[84] Qinyu Zhu, Quan Chen, Yichen Liu, Zahid Akhtar, Kamran Siddique, and Saed Alrabaee. Investigating TrustZone: A Comprehensive Analysis. *Sec. and Commun. Netw.*, 2023, January 2023.