

AFRL-RI-RS-TR-2021-119

FUZZING/CONTROLLED EXCITATION AND MULTI-MODAL SENSOR MONITORING/FUSION FOR HARDWARE-FIRMWARE-SOFTWARE INTEGRITY VERIFICATION

NYU TANDON SCHOOL OF ENGINEERING

JULY 2021

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2021-119 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
SERGEY PANASYUK
Work Unit Manager

/ S /
JAMES S. PERRETTA
Deputy Chief, Information
Exploitation & Operations Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
JULY 2021	FINAL TECHNICAL REPORT	JUN 2020 – FEB 2021

4. TITLE AND SUBTITLE

FUZZING/CONTROLLED EXCITATION AND MULTI-MODAL SENSOR MONITORING/FUSION FOR HARDWARE-FIRMWARE-SOFTWARE INTEGRITY VERIFICATION

5a. CONTRACT NUMBER
N/A

5b. GRANT NUMBER
FA8750-20-1-0502

5c. PROGRAM ELEMENT NUMBER
Other AF

6. AUTHOR(S)

R. Karri
F. Khorrami
P. Krishnamurthy

5d. PROJECT NUMBER
DEST

5e. TASK NUMBER
RO

5f. WORK UNIT NUMBER
01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NYU Tandon School of Engineering
Electrical & Computer Engineering Department
5 Metrotech Center
Brooklyn NY 11201

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RIGB
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2021-119

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Untrusted parties in the commercial-off-the-shelf (COTS) printed circuit boards (PCB) supply chain may poison PCBs with hardware, firmware, and software implants. This project addressed the development of methodologies for Trojan detection in a complex PCB-based system without a golden model without assuming any knowledge of Trojans. The data-driven detection strategy fuses multimodal side channel measurement data, such as Hardware Performance Counters (HPCs), processor use, temperature, and power fluctuations. We develop an anomaly detector that uses design-time hardware and software information about the networked PCB system to implement a run-time evaluator of side channel signals. Our approach comprised of two complementary methodologies: 1) mapping a COTS PCB system to a COTS graph and applying graph-based mathematical construction on ``node" and ``edge" equivalences, clustering of identical nodes and paths, and validation of hypothesized statistical properties on collected side channel data, 2) a simulator-based proxy to generate training data for a one-class machine learning (ML) based classifier for anomaly detection in combination with a probabilistic behavior analyzer. Additionally, we perform hardware and software level fuzzing to amplify side channel information. We integrated a testbed of hierarchically networked PCBs and tested various Trojans.

15. SUBJECT TERMS
Trojan, PCB, Supply Chain, Machine Learning, Trojan Detection, Fuzzing, Anomaly Detection, Sensor Fusion, Side Channels

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**SERGEY PANASYUK**
U	U	U	UU	85	19b. TELEPHONE NUMBER (*Include area code*) **N/A**

Contents

List of Figures.....iii
List of Tables.....v

1.0 Summary 1

2.0 Introduction 2
2.1 Background 5
2.2 Threat Model and Exemplar Trojan Types 7
2.3 Outline of the Report 8

3.0 Methods, Assumptions, and Procedures 9
3.1 A Complex Distributed System Testbed 9
3.1.1 Testbed Architecture 10
3.1.2 Complexity of the Testbed 11
3.1.3 Reconfigurability of the Testbed Architecture 11
3.1.4 Example Applications Implemented on the Testbed 11
3.2 Side Channel Signals 14
3.2.1 Digital Hardware Performance Counters (HPC) 14
3.2.2 Power side channels 18
3.2.3 Thermal side channels 20
3.2.4 Timing side channels 20
3.3 Equivalence/Invariance Based Trojan Detection 24
3.3.1 Graph Model of a COTS PCB System 25
3.3.2 Similarity Metrics 25
3.3.3 Trojan Detection Methodology 26
3.4 Simulator-Trained ML-Based Trojan Detection 29
3.4.1 Strategies for Golden-Free Design/Simulator-Driven Detection 29
3.4.2 Multi-Modal Side Channel Fusion for Anomaly Detection 30
3.4.3 Simulator to Generate Golden-Free Training Data 34
3.4.4 Behavioral Analyzer 37

3.5 Controlled Excitation and Fuzzing 47

4.0 Results and Discussion 49
4.1 Experimental Studies of Equivalence/ Invariance Based Trojan Detection 49
4.1.1 Trojan Detection Using Communication Side Channels . . 49
4.1.2 Trojan Detection using Power side channels 51
4.1.3 Trojan Detection Using HPC Side Channels 53
4.1.4 Trojan Detection Using Thermal Side Channels 55
4.2 Simulator-Trained ML-Based Trojan Detection 57
4.2.1 Multi-Modal Side Channel Measurement Framework . . . 57
4.2.2 Representative Trojans for Experimental Studies 57
4.2.3 Trojan Detection Results 59

5.0 Conclusions 70

6.0 References 71

7.0 List of Symbols, Abbreviations, and Acronyms 76

List of Figures

1 Controlled excitation+multi-modal sensor fusion to verify integrity of a PCB. 2
2 Trojan risks in a third party supply chain. 3
3 Taxonomy of PCB-level hardware Trojans. 6
4 A sample topology of the testbed. It can be reconfigured via software into different topologies without modifying hardware. . . . 9
5 Implementation view of the experimental testbed to emulate a multi-PCB system. 10
6 Mapping of the Gauss-Siedel numerical solver on to the testbed. . 13
7 Mapping of a distributed image acquisition and processing system on to the testbed. 13
8 HPC traces collected from XU4 when running image acquisition performing RGB→Grayscale conversion. 16
9 HPC measurements of CPU-intensive ((a),(b)) and IO-intensive ((c),(d)) processes. 17
10 Power consumption profile of a CPU-intensive application in the four DVFS frequency scaling modes. 18
11 Voltage values when running a CPU-intensive application in the four DVFS modes. 19
12 Temperature readings obtained by running the CPU-intensive application in the four scaling frequency modes by DVFS. 21
13 Server-client connection sequence in MPI communication protocol. 21
14 Time-series plots of timing side-channels: connection setup time, network latency, and network throughput. 23
15 2D frequency distributions of timing side-channels for connections. 24
16 Graph model of a multi-PCB system: (a) Cluster graph around *communication nodes*, (b) Graph G. 25
17 Golden-free Anomaly Detection. 30

18 Anomaly detector for single-/multi-PCB systems. 31
19 Autoencoder for feature extraction from multimodal side channel time series. 33
20 Architecture of simulator for Odroid XU4 for generation of training data for anomaly detector. 35
21 Data Collection from Behavioral Simulator (XU4, Quiescent, No Trojan). 37
22 On-Device Data Collection (XU4, Quiescent, No Trojan). 38
23 Data Collection from Behavioral Simulator (XU4, Test Process on an A15 Core, No Trojan). 39
24 On-Device Data Collection (XU4, Test Process on an A15 Core, No Trojan). 40
25 Data Collection from Behavioral Simulator (XU4, 2 Test Processes on 2 A15 Cores, No Trojan). 41
26 On-Device Data Collection (XU4, 2 Test Processes on 2 A15 Cores, No Trojan). 42
27 Data Collection from Behavioral Simulator (XU4, 3 Test Processes on 3 A15 cores, No Trojan). 43
28 On-Device Data Collection (XU4, 3 Test Processes on 3 A15 cores, No Trojan). 44
29 Data Collection from Behavioral Simulator (XU4, 4 Test Processes on 4 A15 Cores, No Trojan). 45
30 On-Device Data Collection (XU4, 4 Test Processes on 4 A15 Cores, No Trojan). 46
31 UART latency measurements of a connection between SBCs. . . . 48
32 Trojans can be embedded at multiple levels (hardware, firmware, software, interconnects) in a multi-PCB COTS system. 50
33 Measurement snapshots collected over a time window from the communication side channels 51
34 ECDFs and Bhattacharyya distance calculated for RR3 bucket. . . 51
35 ECDFs and Bhattacharyya distance on *network throughput* for RG1 bucket. 52
36 ECDFs and Bhattacharyya distance on *network latency* for RG1 bucket. 52
37 Power traces (in Watts) for the PCBs in the red bucket. 52

38 Thermal side channels of XU3 and XU4 when running a CPU-intensive (RGB2GRAY) test code at 1 GHz. 56
39 Types of Trojans. 58
40 Data collection on XU4 (Quiescent with Fan Info Leak Trojan). . 60
41 Data collection on XU4 (Quiescent with Info Leak Trojan). 61
42 Data collection on XU4 (Quiescent with UART Info Leak Trojan). 62
43 Data collection on XU4 (4 Test Processes on 4 A15 Cores, UART Info Leak Trojan at Fixed Frequency). 63
44 Data collection on XU4 (4 Test Processes on 4 A15 Cores, Performance Degradation Trojan). 64
45 Data collection on XU4 (Test Processes on 4 A15 Cores, Stealthy CPU Info Leak Trojan). 65
46 Data collection on XU4 (Test Processes on 4 A15s, Stealthy Fan Info Leak Trojan). 66
47 Data collection on XU4 (Test Processes on 4 A15s, A15 → A7 replacement Trojan). 67
48 Data Collection on XU4: Quiescent, Stealthy Fan Info Leak Trojan. 67
49 ROC curves for Trojan detection without test processes. 68
50 ROC curves for Trojan detection when test processes run on the four cores. 69

List of Tables

1 Complexity of the testbed. 11
2 SBCs and communication modules in the testbed. 12
3 Side channels supported by the various SBCs in the testbed. . . . 15
4 Deviation in DTW distances in power side channels when XU4-2 is subverted with XU3. 53
5 HPC side channels in the presence of two firmware Trojans: cache flush and an MPI library Trojans. 54
6 % deviation of avg. of HPC side channels from its expected invariance under two firmware Trojans. 54
7 Statistics of thermal side channels when software rootkit runs in MC1-2 (in bold). 55
8 % deviation in thermal side channel when MC1-2 has a rootkit. . . 56
9 Categories of developed Trojans. 58
10 List of developed Trojans. 59
11 Trojan detection performance on Trojans by considering all side channels and without fuzzing (test process instantiations). 61
12 Trojan detection performance for stealthy, performance degradation, and hardware replacement Trojans when considering all side channels with and without fuzzing. 63
13 Trojan detection performance by combining side channels and no fuzzing. 65
14 Trojan detection performance on multi-PCB Trojans using all side channels and no fuzzing. 65

1.0 Summary

Untrusted third parties in the commercial-off-the-shelf (COTS) printed circuit boards (PCB) supply chain may poison PCBs with hardware, firmware, and software implants. Hence, methodologies for detecting malicious implants in PCBs are of crucial importance. State-of-the-art hardware Trojan detection methods require a golden PCB system or a golden model of such a PCB system to detect malicious implants. Furthermore, they do not scale to large-scale COTS PCB systems. Hence, this project addressed the development of methodologies for Trojan detection in a complex PCB-based system – one PCB or a network of PCBs – *without* a golden model. We do not assume knowledge of Trojans, their behavior and impact on any device in the PCB. The data-driven detection strategy fuses multi-modal *side channel* measurement data, such as Hardware Performance Counters (HPCs), processor use, temperature, and power fluctuations. We develop an anomaly detector that uses design-time hardware and software information about the networked PCB system to implement a run-time evaluator of side channel signals to detect discrepancies against expected side channel signal patterns.

The approach developed in this project has two complementary methodologies. In the first methodology, we map a COTS PCB system to a COTS graph and propose a golden-free Trojan detection methodology based on i) a graph-based mathematical construction on "node" and "edge" equivalences, and ii) clustering of identical nodes and paths and validation of hypothesized statistical properties on side channel information taken from COTS components on the PCB. In the second methodology, we develop a simulator-based proxy to generate training data rather than requiring data from an actual device to train a one-class machine learning (ML) based classifier for anomaly detection. We augment the ML-based classifier with a probabilistic behavior analyzer formulated based on design information to validate expected inter-relationships between side channels to detect anomalies. Additionally, we perform hardware and software level fuzzing to detect malicious Trojan implants by enhancing their visibility in side channel signals. The observed multi-modal side channel signals and their inter-relationships are analyzed to detect deviations from expected behavior captured through equivalence/invariance properties, behavioral characterizations, and ML classifier models. To evaluate efficacy of the methodologies, we integrated a testbed comprised of hierarchically networked PCB devices to emulate a multi-PCB system. The efficacy of the developed golden-free detection methodologies was tested on several Trojans at the hardware, software/firmware, and interconnect (machine-in-the-middle – MITM) layers.

2.0 Introduction

System integrators employ offshore PCB manufacturers exposing them to malicious actors in the commercial PCB chain. Hence, verifying the integrity of PCBs is crucial. This project summarized in Figure 1 addressed the development and demonstration of techniques to detect anomalies in hardware/firmware/software using robust multi-modal monitoring of processor operation of multiple on-processor, off-processor, and inter-processor digital and analog side channels. The approach developed in this project introduces controlled excitation/fuzzing at hardware-firmware-software levels to expand the observation envelope to increase visibility into anomalies.

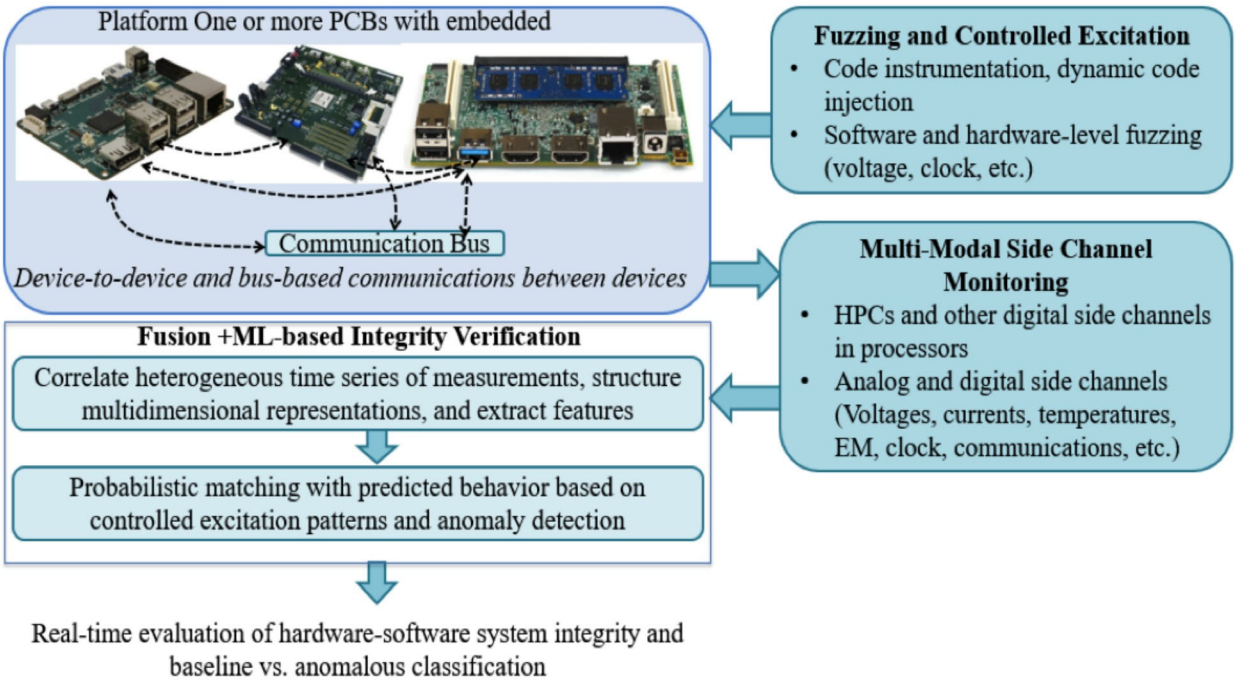


Figure 1: Controlled excitation+multi-modal sensor fusion to verify integrity of a PCB.

The COTS PCB supply chain as shown in Figure 2 is complex and may be compromised by malicious implants such as replacement of components, hardware Trojans and malicious firmware and software. This is analogous to vulnerabilities in Integrated Circuits (ICs) due to the complex

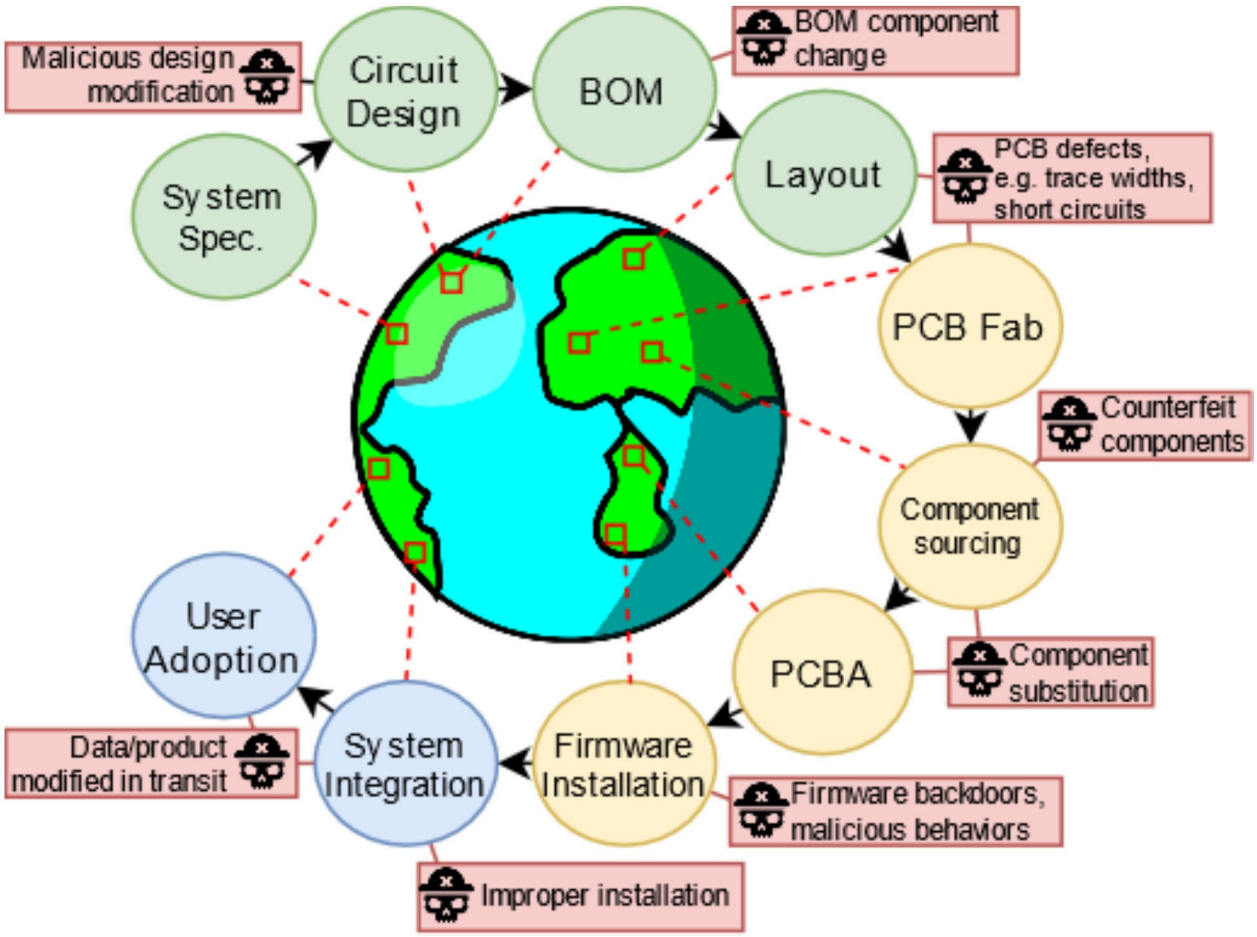


Figure 2: Trojan risks in a third party supply chain.

and distributed nature of the IC supply chain making malicious modifications,Hardware Trojans, by third parties a critical threat [1, 2, 3]. Malicious actors in the supply chain in the offshore development of complex COTS PCB systems can significantly impact crucial applications. Hence, detection of Trojans in PCBs hs been identified as a research need. DARPA SHEATH [4] focuses on Hardware Trojans targeting internet routers and computer server motherboards. This echoes Bloomberg's reported 'Big Hack' [5] and 'Long Hack' [6], which alleged that Hardware Trojans were being introduced in the PCB supply chain [7, 8, 9] to modify computer motherboards such that they are infected with malware each time they were powered up [10]. The alleged 'Big Hack' Trojan was notable for its cross-domain nature. It altered the PCB to hide a microcontroller inside a capacitor used for signal conditioning. This microcontroller could then alter the data on a signal line used to load firmware from a memory chip. This modification can install a software backdoor in the *baseboard management controller* on the PCB. This backdoor enables remote access to the machine to install malicious code that can exfiltrate sensitive data. Though Bloomberg's claims were met with skepticism, such attacks *are* possible [11], highlighting the research need. Research has shown how third parties may add, remove, or alter components, replace legitimate components with counterfeits, and add, remove, or alter PCB traces [12].

State-of-the-art side channel based Trojan detection uses supervised learning to identify a sys-

tem as clean or Trojan-infected. However, a significant challenge in detecting Trojans in PCBs is the absence of golden (known-good) PCBs, implying that there is no empirical 'ground truth' for comparison for the purpose of anomaly detection. Typically, a system architect creates the blueprint of complex COTS PCB system under a Joint Design Manufacturing (*JDM*) model and outsources it to a third-party vendor for manufacturing the integrated system. Hence, the system architect does not have access to a "golden" integrated system. A malicious third-party manufacturer may inject the Trojan so that Trojan functionality is hidden and survives post-manufacture tests before triggering. Finding such implants that could be at hardware, firmware, interconnect, or software levels is a challenging problem since it is not feasible to test for all possible Trojans (including zero days).

The focus in this project is to explore the research question: Can we detect Trojans in a complex PCB-based system – one PCB or a network of PCBs – *without* a golden PCB? The approach should be oblivious to characteristics of the Trojans such as their behavior, size, location, and impact on the system. We propose a data-driven detection strategy using *multi-modal side-channel data*, such as Hardware Performance Counters (HPCs), temperature, and power fluctuations. The anomaly detector uses design-time hardware and software information about the networked PCB system to implement a run-time evaluator of side channel signals to detect discrepancies against expected side channel signal patterns. The approach developed in this project is comprised of two complementary methodologies:

- In the first method, we develop a graph-based approach using multiple side channel signals to detect Trojans in an integrated system without a golden integrated system. For this purpose, we map a COTS PCB system to a COTS graph and propose a golden-free Trojan detection methodology based on i) a graph-based mathematical construction on "node" and "edge" equivalences, and ii) clustering of identical nodes and paths and validation of hypothesized statistical properties on side channel information taken from COTS components on the PCB. The expected side channel invariances in identical/similar devices and among groups of devices with the same communication interfaces and specifications are formulated as *Node Invariances* and *Path Invariances*, respectively. Statistical dissimilarities from expected invariances expose Trojans.

- In the second methodology, we develop a simulator-based proxy to generate training data rather than requiring data from an actual device to train a one-class machine learning (ML) based classifier for anomaly detection. We augment the ML-based classifier with a probabilistic behavior analyzer designed based on design information to validate expected inter-relationships between side channels to detect anomalies.

We perform hardware and software level fuzzing to collect side channel information and facilitate detection of malicious Trojan implants. The observed multi-modal side channel signals and their inter-relationships are analyzed to detect deviations from expected behavior captured through equivalence/invariance properties, behavioral specifications, and ML classifier models. Primary focus areas addressed in the project included:

- Development of the golden-free, multi-modal side channel based Trojan detection methodologies outlined above.

- Demonstration of the efficacy of Trojan detection on stealthy and multi-PCB Trojans.

- Demonstration of the utility of *fusing* multiple side channel modalities and *fuzzing* (e.g., defender-chosen codes to exercise the PCB during side channel collection, hardware knobs) for enabling higher accuracy in detection of Trojans.

To evaluate the efficacy of the methodologies developed in this project, we integrated a testbed comprised of hierarchically networked PCB devices to emulate a multi-PCB system. The efficacy of the developed golden-free detection methods was tested on several Trojans at the hardware, software/firmware, and interconnect (machine-in-the-middle – MITM) layers.

2.1 Background

Trojan attacks and defenses on PCBs have been addressed in the literature [8, 10]. In one attack, inter-trace spacings on PCBs are modified [13, 8]. Other Trojan implants replace components with counterfeits or components with unadvertised radio links to the attacker [10]. State-of-the-art defenses against PCB Trojans compare measurements such as delays on the suspected PCB and compare them against those from a golden PCB [3]. A golden integrated system is a pre-requisite for these approaches and is counter to the way such systems are built using the distributed supply chain for PCB manufacturing, integration, and testing. Further, extant methods do not offer real-time detection in the PCBs. Hence, these methods cannot detect Trojans inserted into an integrated COTS system of heterogeneous PCBs as a firmware/software patch during system operation.

Trojan detection approaches in ICs have been explored. Side-channel fingerprinting of ICs has been studied to obviate prior knowledge of golden IC design. [14] did destructive testingof ICs from the same family to obtain multiple side channel fingerprints and validate for consis- tency with untrusted ICs post-manufacturing. While their method is golden-free and is a frontier inside-channel fingerprinting, destructive testing is essentially expensive and when applied to Trojan detection in large-scale PCBs with thousands of ICs, cost grows exponentially. Further, works in [15], [16] present golden-free methods by fingerprinting various side channels of IC simulations. While these approaches provide methods in a golden-free setting using side channel fingerprints of IC families and simulations, our domain-agnostic approach, in contrast, uses side-channel fingerprints of ICs and communications between ICs, as a low-cost golden-free technique scalable to hundreds of inter-connected ICs. In addition to IC fingerprinting, various side channels have been used to monitor COTS systems and to detect malware (e.g., thermal imagery in [17], CPU core temperatures and fan acoustics in [18], and HPCs in [19]). *We combine multiple side chan- nels with their multi-modalities in a golden-free framework to detect different Trojan implants in multi-PCB systems.* Further, [15] presents a golden-free method by fingerprinting side channels of simulations of the ICs obtained before manufacturing. [16] uses path delay side channel. Golden-free Trojan detection in a processor analyses bit power consistency was proposed in [20]. A signal processing method in [21] detects Trojans in mobile platforms without a reference model.

In order to categorize and reason over PCB-level Trojans, a few taxonomies have been proposed. For instance, [13] is a taxonomy where Trojans are classified as malicious modifications or hidden elements. A more comprehensive taxonomy builds on the IC taxonomy of [22] to account for variations between ICs and PCBs is reported in [23]. Another taxonomy in [24] classifies IC and PCB Trojans, but fails to capture the PCB implant characteristics.

A more comprehensive taxonomy in [25] classifies Trojans according to the modification, location, integration, trigger and effect. These characteristics are based on their effect on the Trojan

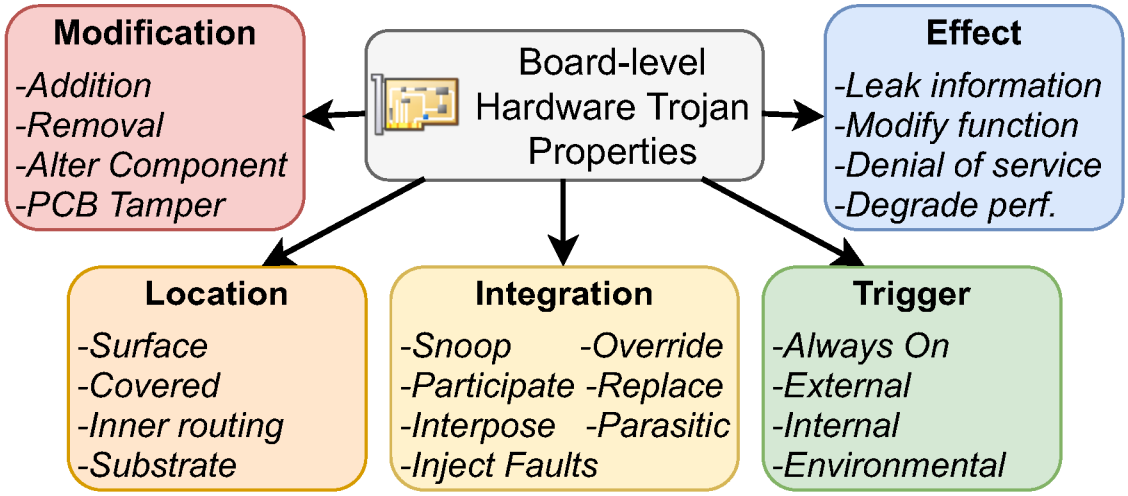


Figure 3: Taxonomy of PCB-level hardware Trojans.

detection methods. Figure 3 shows a classification of PCB-level Trojans according to these characteristics. Here, a 'modification' refers to alteration of the PCB to introduce the Trojan effect. Several modifications are possible such as addition of components [5], removing/tampering with components [11] or removing/tampering with PCB routing layers/substrate [8, 13]. The type of modification affects the majority of suggested Trojan detection methods. The location of a Trojan is important for multi-modal imaging-based methods. To detect Trojans in various locations, different imaging modalities are required: DSLR cameras, for example, cannot detect modifications in inner-layer routing, despite the fact that they are fast and efficient at checking IC serial numbers. However, other detection methods (for instance, a side channel like power measurement) will not be as impacted by the physical location of the Trojan.

Trojans can interact with the PCB such as snooping signals [26, 27], overriding signals to alter system behaviour [28], interposing on a signal from the legitimate device to control the signal without causing conflict [29, 30], replacing a genuine component with a counterfeit [31], participating in inter-component communications [32, 33], altering parameters by thinning PCB traces [8], and injecting faults by operating components outside their specifications [34, 35]. These integration characteristics reveal security flaws on a PCB.

Some PCB-level Trojan detection techniques borrow from IC Trojan detection strategies. A number of these come from functional tests, for instance, through careful test pattern generation targeting locations most susceptible to attacks [13]. Though test patterns and testing may detect basic Trojans, PCBs are large and Trojans with rare triggers can escape. Functional testing is not effective in complex PCB systems [36].

Other Trojan detectors rely on side channel fingerprints. Destructive tests are performed on ICs from the same family to collect side channel fingerprints and verify post-manufacturing compatibility with untrusted ICs in [14]. Although the approach is gold-free and is a breakthrough in side-channel fingerprinting, destructive testing is costly, when applied to Trojan detection in large-scale PCBs with thousands of ICs. PCBs will have unique characteristics based on design and manufacturing variations, and a Trojan will alter the signature of these side channels. Such variations may be measured by checking resistance, capacitance, or other electrical quantities on a PCB [10]. PCB Trojans can be detected by statistical analysis on the side channel measurements.

For instance, [13] measures side channels to verify security-critical nets against a simulated behavior. Another side-channel detection mechanism employs the JTAG scan chains to determine delays of PCB traces [37]. It tests the PCB route delays by flipping the JTAG test clock and filling the JTAG scan chain with specific patterns of ones and zeros. The number of ones that spread to registers that previously had zeros is determined by the delay properties of the traces on the PCB. Variations in the outcome of this test may suggest tampering or presence of Trojan components. The study proposes this approach to authenticate PCBs at run-time.

Since PCB properties are similar to those used for physically unclonable functions (PUFs), another approach explores PCB routing patterns and passive components to produce a PCB-level PUF [38]. Similar to this, [7] quantifies the resistance shift on a line caused when a drop of solder is added. They design a circuit to verify the resistance of sensitive PCB traces against factory-calibrated values. The circuit could track changes in resistance caused by solder balls,IC gates, and probe wires, all valid Trojan attacks. To monitor variations in impedance caused by tampering, [39] employs PCB-level ring oscillator (RO) PUFs. It measures RO frequencies during device initialization and compares them to measurements programmed at a trusted facility.This approach assumes that the presence of an implant increases trace impedance, causing the frequency of the RO PUFs to increase. Likewise, [40, 41] employ RO PUFs to detect counterfeits and malicious modifications by adding the resonant frequency of a PCB's power and ground planes. They discover that the resonant frequency of a PCB' control and ground planes is affected by manufacturing variance, and that at the resonant frequency, the impedance of the PCB's power network rises. When the resonant frequency is added to the PCB control pins, the power applied to the RO PUFs is affected by PCB production and IC variations that specify RO PUF frequency. They derive a system-level signature for the PCB.

ML-based methods have been used to detect Trojans in ICs. [42] uses a one-class Support Vector Machine (SVM) and K-nearest-neighbors over de-capped IC images, operational characteristics, and side channels to detect Trojans. In [43], an ML method determines Trojan trigger command bytes and measures their effect on the system using one-class SVM, Local Outlier Factor, and Isolation Forest. Golden-free Trojan detection is proposed in [44, 45] by training a one-class SVM on the data generated from a simulated version of the IC (Trojan-free) while considering short-term aging of the IC.

2.2 Threat Model and Exemplar Trojan Types

We consider a Joint Design Manufacturer (*JDM*) model where a commercial enterprise designs a distributed system (e.g., a data center or distributed compute platform) and outsources its manufacturing/integration to a possibly untrusted party and supply chain (the adversary). These systems may have server motherboards, routers, etc. as part of a multi-PCB system of COTS hardware. The complex, distributed supply chain involved in building multi-PCB systems introduces Trojan risks (e.g., as in the widely discussed "Big Hack" [5, 6]) anywhere in the COTS system, compromising integrity and security. *The challenge and the focus in this project is to detect anomalous behaviors absent a golden baseline [4].*

Using controlled excitation and ML-based anomaly detection, the approach in this project seeks to detect a range of anomalies at hardware, firmware, and software levels. Representative anomalies include for example:

- Hardware-level anomalies 1: an attacker inserts/modifies a hardware component to modify the I/O operation between boards. This is emulated by inserting an additional machine in the middle (MITM) board connected to the I/O lines between SBCs in the experimental testbed.
- Hardware-level anomalies 2: a SBC or a component on a SBC is disabled or tampered with (e.g., by disabling power supply to a component or subsystem on the board).
- Firmware-level anomalies: an attacker modifies firmware of one or more SBCs and/or routers (e.g., modification of available open-source firmware of the SmartPower2 component for power monitoring in Odroid SBCs).
- Software-level anomalies: kernel- or user-level modifications (e.g., overriding system call behavior using a kernel-level rootkit, modifying kernel-level device drivers, running additional kernel-level modules or user-level applications to cause anomalous behavior).
- Multi-step/cross-layer attacks: the attacker can trigger malicious firmware or compromised software using hardware, firmware, or software level attacks. For instance, a privilege escalation attack may deploy malicious firmware, rootkits, or malware leading to a range of software-based attacks.

Formally, the problem being addressed can be stated as follows: Given (some) off-line information on the target integrated multi-SBC system design and on-line access to the integrated system (integrated by a possibly malicious third party), collect side channel signals to detect presence of Trojans and determine their locations. The defender can deploy controlled excitations (hardware and/or software fuzzing) as part of the Trojan detection methodology, collect multi-modal side channels under different excitations, and analyze the collected side channel time series signals to determine likelihood of the presence of Trojans in the system.

2.3 Outline of the Report

This report is organized as follows. The methodologies and implementations developed in this project are described in Section 3.0 including the integration of the experimental testbed (Sec- tion 3.1), considered side channels (Section 3.2), Trojan detection methodologies (Sections 3.3 and 3.4), and controlled excitation and fuzzing methodologies (Section 3.5). The experimental studies of the efficacy of the Trojan detection methodologies are described in Section 4.0 consid-ering several types of Trojans in single-PCB and multi-PCB scenarios. Concluding remarks are summarized in Section 5.0.

3.0 Methods, Assumptions, and Procedures

3.1 A Complex Distributed System Testbed

To experimentally evaluate the Trojan detection methodology developed in this project, the first task addressed in the project was to build a scalable, distributed, heterogeneous and multiple SBC testbed. This testbed can support multiple industrial grade distributed applications and represents/ emulates complex distributed systems composed of heterogeneous SBCs. While we consider interconnections of SBCs in our testbed, note that the methodologies also apply to applications involving complex multi-processor single boards. The SBCs connect to a network switch which is controlled by various routers. Each router has their own virtual local area networks (VLANs) each with one or more SBCs. The routers are interconnected and can be reconfigured to emulate distributed architectures as per user need. In subsequent sections, we describe the testbed and discuss its complexity.

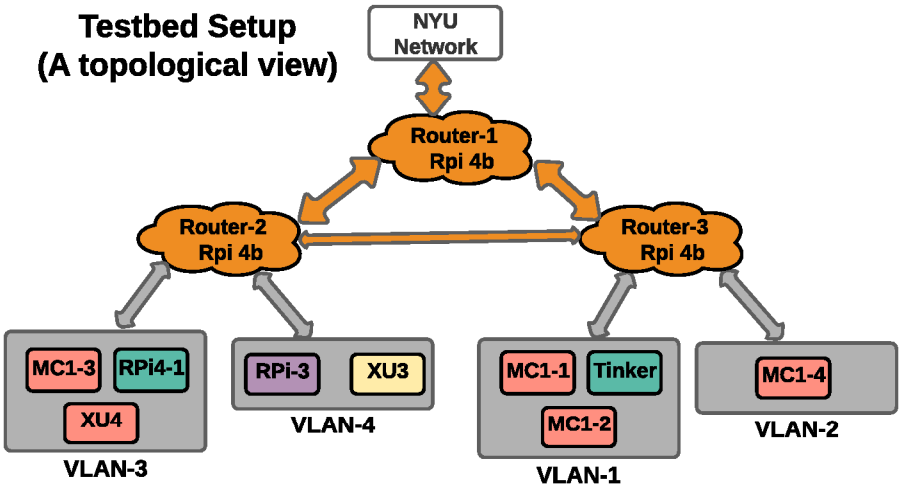


Figure 4: A sample topology of the testbed. It can be reconfigured via software into different topologies without modifying hardware.

3.1.1 Testbed Architecture

The NYU testbed is assembled as a hierarchically interconnected architecture as shown in Figures 4 and 5. In this testbed, SBCs are interconnected via a network switch and routers (to emulate a multi-PCB system). The design has clusters of PCB devices connected to a router, emulating clusters of components on a PCB or multiple PCBs in a COTS system. We used a 48-port Netgear switch as a network hub for the various routers. This enables support for multiple VLANs (emulating multiple components on a single PCB or multiple PCBs in a multi-PCB system). The master router $R1$ allows remote access to the testbed to configure the testbed and to work on the testbed (run applications, monitor side channels, dynamically configure system and PCB parameters, etc.). Two slave routers $R2$ and $R3$ are connected to the master router $R1$ and can support dynamic setup of multiple VLANs (to emulate interacting components on a PCB or interacting PCBs in a multi-PCB system). We used Raspberry Pi 4b SBCs as the routers. Every SBC is connected to one of the routers and is part of a VLAN/cluster. SBCs in the same VLAN communicate with each other via the ethernet protocol without going through a router. Communicating between SBCs in different VLANs involves a router. The SBCs can also communicate using serial bus protocols such as UART and SPI.

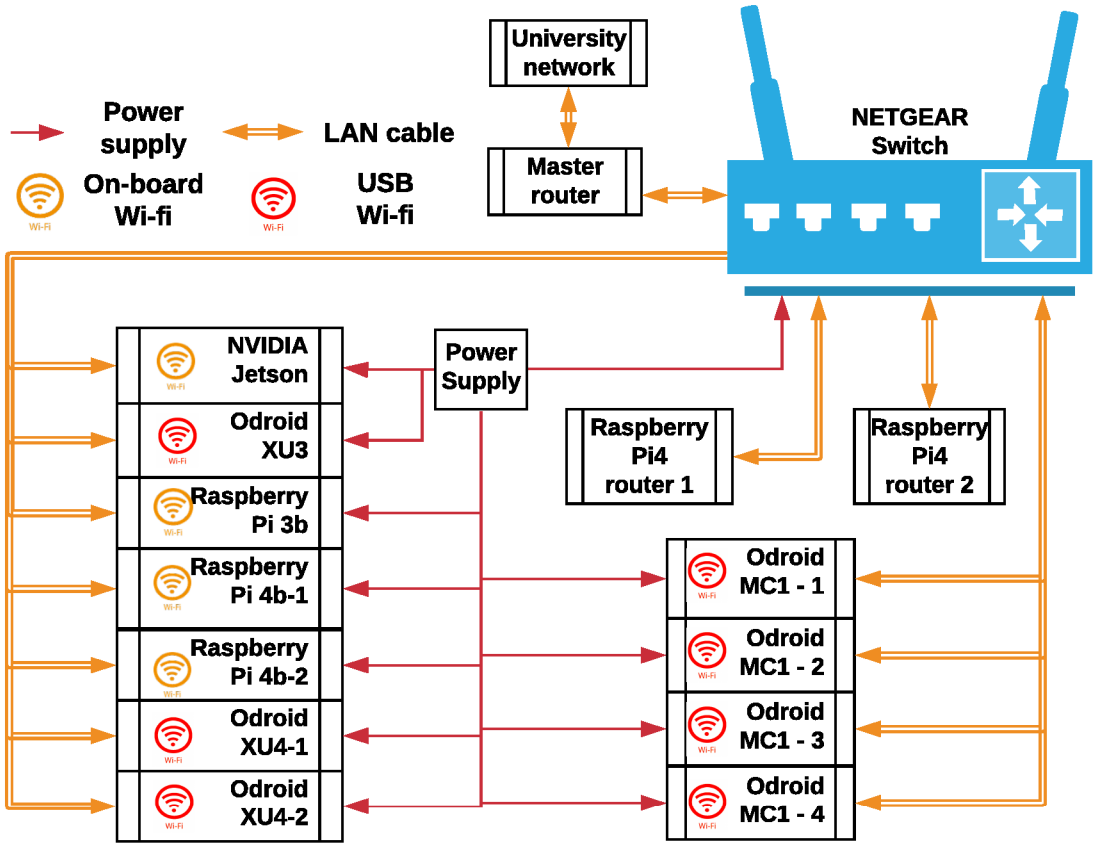


Figure 5: Implementation view of the experimental testbed to emulate a multi-PCB system.

We assembled the testbed as a heterogeneous collection of SBCs. These SBCs are diverse in terms of the CPU speed, the CPU architecture bit width, the instruction set architectures, and

integrated network capabilities. Table 2 summarizes the characteristics of every SBC in the testbed. For SBCs lacking integrated power sensors, we continuously monitor the power consumption using smart power boards. The testbed is remotely accessible, dynamically reconfigurable, and scalable to a large system by adding more switches, routers and SBCs.

3.1.2 Complexity of the Testbed

The testbed framework supports a range of distributed systems – simple to complex. The complexity figure-of-merit and goals of the testbed in Figure 4 are listed in Table 1. This figure-of-merit from the SHEATH BAA [4] considers the number of CPUs, the number of GPUs, cores/CPU, cores/GPU, and threads/core in every SBC used in the testbed. The figure-of-merit for every SBC on the testbed is tabulated in the Table 1. The SBCs have complexity metrics ranging from 4 to 132 yielding a cumulative figure-of-merit of 340. This figure of merit however does not consider the diversity and complexity of the SBCs in terms of the input-output communication peripherals.

Table 1: Complexity of the testbed.

	Testbed Complexity: Single Board Computers (SBC)							
	Odroid			Raspberry		Atomic	ASUS	NVIDIA
	XU3	XU4	MC1	Pi 4b	Pi 3b	Pi	Tinkerboard	Jetson Nano
CPUs	2	2	2	1	1	1	1	1
GPUs	1	1	1	-	-	-	1	1
Cores/CPU	4	4	4	4	4	4	4	4
Cores/GPU	8	8	8	-	-	-	16	128
Threads/core	1	1	1	1	1	1	1	1
# in testbed	1	5	4	4	1	2	1	1
Figure-of-Merit	16	80	64	16	4	8	20	132

3.1.3 Reconfigurability of the Testbed Architecture

The developed testbed is easily reconfigurable. Different distributed system architectures can be explored without manually intervening. The testbed offers several benefits. First, the testbed provides "knob-like" configuration scripts. This allows users to switch between different distributed architectures by specifying the IP addresses of the SBCs belonging to each of the VLANs. This in turn allows automatic exploration of several distributed architectures. Second, the users can study various topologies since the testbed can establish bi-directional communication between SBCs and among SBCs and routers. Furthermore, additional VLANs can be created using the same configuration scripts to study more complex distributed system architectures.

3.1.4 Example Applications Implemented on the Testbed

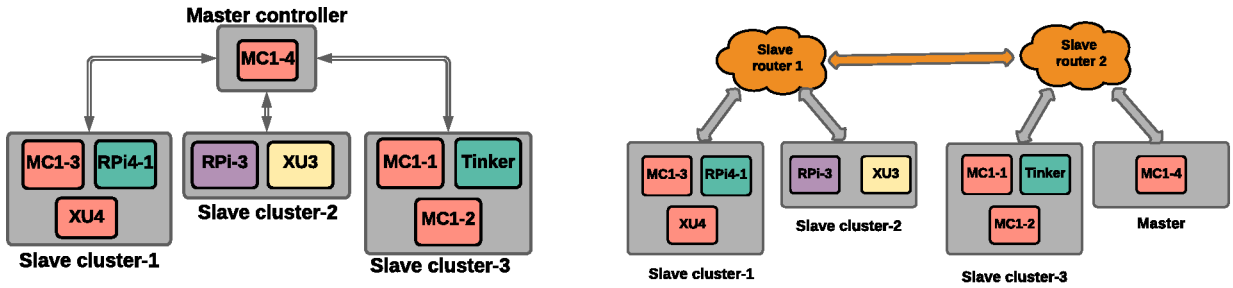
To demonstrate the reconfigurability of the testbed, we implemented two real-world distributed applications as outlined below.

Table 2: SBCs and communication modules in the testbed.

Category	Charac-teristics	XU4					Atomic pi	Asus Tinkerboard	NVIDIA Jetson Nano
CPU Configuration	type	Arm Cortex A7+A15					Intel atom x5-Z8350	Arm Cortex A17	Arm Cortex A57
	architecture	32-bit	32-bit	32-bit	64-bit	64-bit	32-bit	32-bit	64-bit
	micro-architecture	armv7f	armv7f	armv7f	armv8-A	armv8-A	x86	armv7-A	armv8-A
	# cores	8	8	8	4	4	4	4	4
	#processors	2	2	2	1	1	1	1	1
	frequency	1.5GHz (A7), 2GHz (A15)	1.5GHz (A7), 2GHz (A15)	1.5GHz (A7), 2GHz (A15)	1.5GHz	1.2GHz	1.92GHz	1.8GHz	1.43GHz
	RAM	2GB	2GB	2GB	4GB	2GB	2GB	2GB	4GB
Cache	L1	64KB+ 64KB	64KB+ 64KB	64KB+ 64KB	32KBx4 D + 48KBx4 I	32KBx4 D + 32KBx4 I	2MB smart	32KB D+ 32KB I	32KBx4 D + 48KBx4 I
	L2	512KB+ 2MB	512KB+ 2MB	512KB+ 2MB	1MB	2MB	cache	1MB	2MB
GPU	type	Arm Mali T-628	Arm Mali T-628	Arm Mali T-628	-	-	Intel Integrated Graphics unit	Arm Mali T760 MP4	NVIDIA Maxwell
	frequency	600MHz	600MHz	600MHz	-	-	450MHz	650MHz	921MHz
Peripherals	# USB 2.0	4	4	1(host)	2	2	2	4	3
	# USB 3.0	1	1	0	2	2	1	-	1
	GPIO pins	20+10	20+10	0	40	40	20	40	40
Connectivity	ethernet	Gigabit ethernet (over USB3.0)	100 Mbps ethernet	Gigabit ethernet (over USB 3.0)	Gigabit ethernet (on-chip)	Gigabit ethernet (over USB 2.0)	Gigabit ethernet (on-chip)	Gigabit ethernet (on-chip)	Gigabit ethernet (on-chip)
	Wi-fi	USB dongle	USB dongle	no	on-board	on-board	no	on-board	on-board
	UART	2	1	0	6	1+1(mini)	1	4	3
	SPI	2	2	0	4	2	1	2	2
Side-channels	Power	externel	on-board	externel	external	external	external	external	on-board
	Thermal	yes	yes	yes	yes	yes	yes	yes	yes
	HPC	yes	yes	yes	yes	yes	yes	yes	yes
Firmware	Linux version	4.14.180	4.14.180	4.14.141	5.4.51	5.4.51	4.19.126	4.17.134a	4.19.126

[Grab your reader's attention with a great quote from the document or use this space to emphasize a key point. To place this text box anywhere on the page, just drag it.]

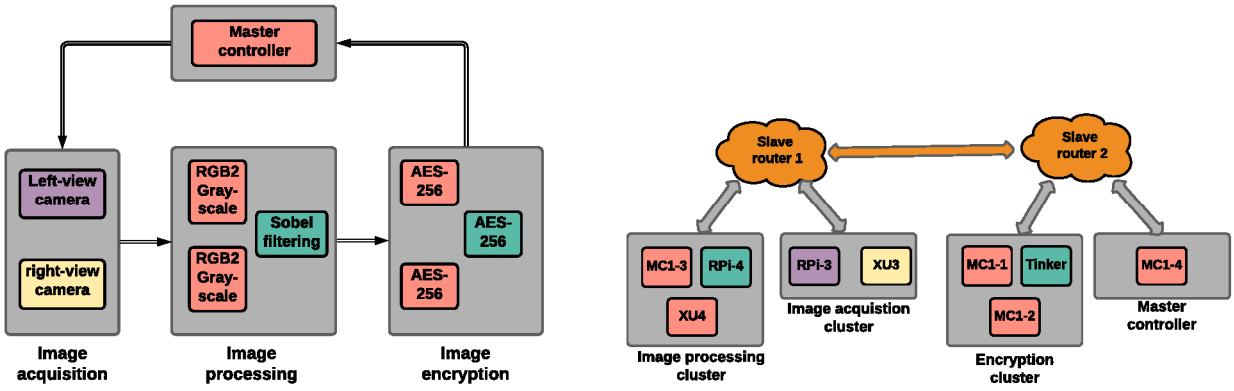
Approved for Public Release; Distribution Unlimited.



(a) (b)

Figure 6: Mapping of the Gauss-Siedel numerical solver on to the testbed. A master SBC (MC1-4) sends tasks to compute on the three slave clusters of SBCs.(a) Gauss-Siedel distributed application, (b) Implementation on testbed.

Gauss-Siedel distributed numerical solver: Many large-scale applications can be translated into linear algebraic systems. Gauss-Siedel solver is an iterative numeric algorithm to provide approximate solutions to such large-scale linear algebraic systems. We designed the Gauss-Siedel distributed application shown in Figure 6(b) and implemented the application on our testbed as shown in Figure 6(a) and study its characteristics. We selected a master SBC in a VLAN to implement the master which offloads iterations of the numeric matrix computations to the four slave SBCs. The master SBC sends the tasks in parallel to the slave SBCs and receives the results of the computations asynchronously. The slave SBCs in VLAN 1 have different communication characteristics compared to the slave SBCs in VLANs 2 and 3. SBCs on VLANs 2 and 3 are heterogeneous in terms of computation characteristics.



(a) (b)

Figure 7: Mapping of a distributed image acquisition and processing system on to the testbed. (a) Distributed application structure for image acquisition and processing. (b) Implementation on testbed.

Distributed image acquisition and processing: We implemented an image processing acquisition and processing system as described in Figure 7(a). A **Master controller** initiates the acquisition of a 2-dimensional image from a **Image acquisition cluster** by sending request signals. The

image acquisition controllers, upon capturing a "left" angle (RPI-3) and "right" angle (XU3) of an image, send the captured data to the Image processing cluster. This image processing networked cluster extracts edges of the image using a sobel filter on the left and right images; each SBC in the cluster operates on partial workloads. Upon completion of this job, this cluster sends the processed image to the encryption cluster that then encrypts the image edges and returns the encrypted data to master controller which saves the data to disk. The topology of the distributed application is shown in Figure 7(b).

3.2 Side Channel Signals

The side channels that we considered include (i) digital hardware performance counters (HPCs) and other CPU-based measurements (e.g., CPU load and frequency), (ii) on-chip power consumption, measurements from the thermal subsystem such as temperature and fan speed, and voltage consumed by the SBCs in the different operating modes, and (iii) timing side channels (SBC- to-SBC- network latency, throughput and connection setup time). These side channels provide complementary and supplementary insights into the applications running on the testbed which can in turn inform anomaly detection.

3.2.1 Digital Hardware Performance Counters (HPC)

HPCs are built into all CPUs and GPUs in the SBCs. HPCs can be used to monitor low-level micro-architectural events when an application runs on the SBC. The HPCs supported differ from one CPU to another and from one GPU to another. The defender can monitor some or all HPC measurements shown in Table 3 that are supported by the processor on the PCB. The HPCs provide measurements related to processor activity over a time period. These measurements include, for example, numbers of instructions, cycles, branches, and L1/L2/L3 cache hits and cache misses. The particular HPCs that are available is processor-dependent and there are usually constraints on which and how many HPCs can be read simultaneously. HPC measurements can be collected for specific processes/threads or system-wide.

We use perf [46] and PAPI [47] software to monitor HPC events on the SBCs in the testbed. To understand the role of HPCs as digital side-channels, we captured the HPC variations in SBCs while running two distributed applications. For the distributed image acquisition and processing system, two SBCs do image acquisition, wherein they perform RGB→Grayscale conversion. Figure 8 shows the HPC variations of the SBCs that run RGB→Grayscale conversion. The plots offer insights into the characteristics of the CPU at run time. The periodic dips in the HPC plots are due to wait time between successive runs of the application when the CPU is idle.

Aiding Trojan detection: The HPC plots show the behavior and the workload of CPU while performing an application. A Trojan machine-in-the-middle would introduce delays causing unexpected changes in the amplitude in the side-channel measurements of an SBC on the receiving end of a connection. This may manifest as a time-shift in the peaks of HPC measurements (e.g. increased time-period between consecutive peaks in Figure 8) which can be used to detect anomalous behavior. Our prior work [19] has detected such anomalies using HPC side-channels.

We studied variations in HPCs in the SBCs during runs of CPU- and IO-intensive applications. These applications use IO and CPU resources. We study CPU behavior when running processes

Table 3: Side channels supported by the various SBCs in the testbed.

Side-channels		Single Board Computers in the Testbed							
		Odroid			Raspberry		Atomic	ASUS	NVIDIA
		XU3	XU4	MC1	Pi 4b	Pi 3b	Pi	Tinkerboard	Jetson Nano
Instructions	Total	✓	✓	✓	✓	✓	✓	✓	✓
	Load	✗	✗	✗	✓	✓	✗	✗	✗
	Store	✗	✗	✗	✓	✓	✗	✗	✗
	Branch	✓	✓	✓	✓	✓	✓	✓	✓
Branches	taken	✗	✗	✗	✗	✗	✓	✓	✗
	mispredictions	✓	✓	✓	✓	✓	✓	✓	✓
Cycles consumed		✓	✓	✓	✓	✓	✓	✓	✓
L1-Cache	d-cache miss	✓	✓	✓	✓	✓	✗	✓	✓
	i-cache miss	✓	✓	✓	✓	✓	✓	✓	✓
	load miss	✗	✗	✗	✓	✓	✗	✓	✗
	store miss	✗	✗	✗	✓	✓	✗	✗	✗
	d-cache access	✓	✓	✓	✓	✓	✓	✗	✓
	i-cache access	✓	✓	✓	✓	✓	✓	✗	✓
	total access	✗	✗	✗	✗	✗	✓	✗	✗
	i-cache hits	✗	✗	✗	✗	✗	✓	✗	✗
	i-cache read	✗	✗	✗	✗	✗	✓	✗	✗
	d-cache read	✗	✗	✗	✓	✓	✗	✗	✗
	d-cache write	✗	✗	✗	✓	✓	✗	✗	✗
L2-Cache	d-cache miss	✓	✓	✓	✓	✓	✓	✓	✓
	i-cache miss	✗	✗	✗	✗	✗	✓	✗	✓
	store miss	✓	✓	✓	✓	✓	✓	✗	✓
	total miss	✗	✗	✗	✗	✗	✓	✗	✗
	d-cache access	✓	✓	✓	✓	✓	✓	✗	✓
	d-cache hit	✗	✗	✗	✗	✗	✓	✗	✗
Look aside buffer misses	data translation	✓	✓	✓	✗	✗	✓	✓	✗
	instn translation	✗	✗	✗	✗	✗	✓	✗	✗
Power	CPU	✓	✗	✗	✗	✗	✗	✗	✓
	DRAM	✓	✗	✗	✗	✗	✗	✗	✗
	Total	✓	✓	✓	✓	✓	✓	✓	✓
Thermal	CPU	✓	✓	✓	✓	✓	✓	✓	✓
	Total	✓	✓	✓	✓	✓	✓	✓	✓
Networking	Setup time	✓	✓	✓	✓	✓	✓	✓	✓
	round-trip time	✓	✓	✓	✓	✓	✓	✓	✓
	throughput	✓	✓	✓	✓	✓	✓	✓	✓

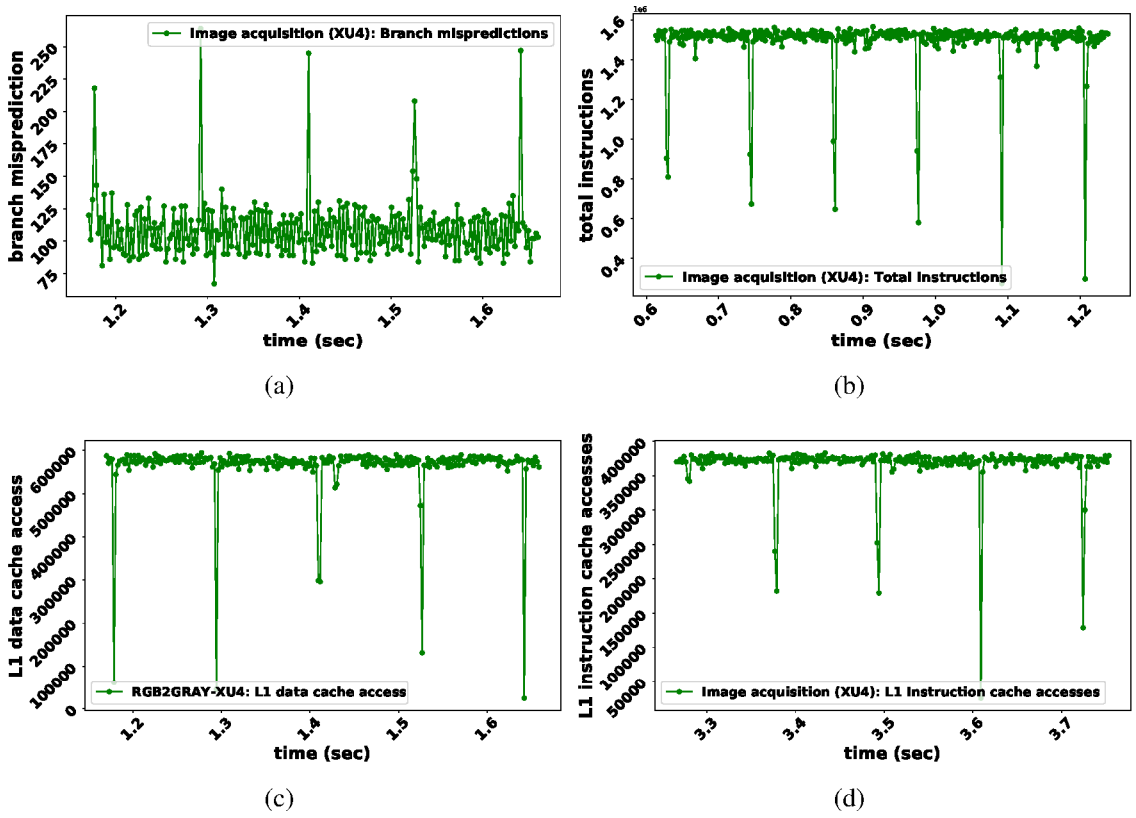


Figure 8: HPC traces collected from XU4 when running image acquisition performing RGB→Grayscale conversion.

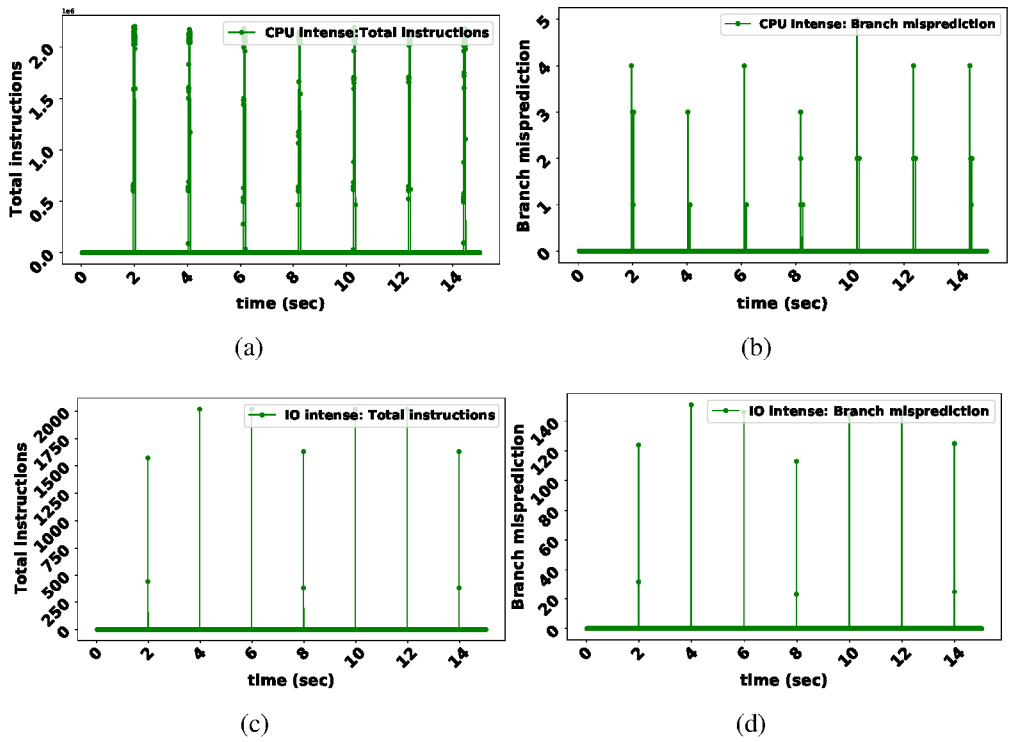


Figure 9: HPC measurements of CPU-intensive ((a),(b)) and IO-intensive ((c),(d)) processes.

with diverse characteristics. Figure 9 shows the HPC variations when running diverse applications. Periodic spikes in the plots indicate that the resource consumption by the application causes bursts in the number of instructions and branch mis-predictions. The plots indicate that these sudden bursts in branch mis-predictions are on average higher when running CPU-intensive applications than when running IO-intensive ones. This is because CPU-intensive applications use the CPU resources more than IO-intensive applications which mostly consume the I/O buffer for read-write operations. The multiple, subtle peaks for a single run of a CPU-intensive application in Figure 9(a) validate this understanding. A single run of the IO-intensive application produces a single peak in Figure 9(c).

Aiding Trojan detection: HPC plots in Figure 9 show the behavior of applications that use IO- and CPU-based resources as periodic peaks. A Trojan intervening in a connection between SBCs adds delay and noise between the consecutive peaks. HPC measurements can be analyzed to spot the delays as possible anomalies.

In addition to HPC readings, measurements of the CPU load and frequency time signals can be useful to infer CPU utilization patterns. The level of activity of the CPU can often be measured using kernel-level readings of CPU usage (e.g., separate measures for user, system, and idle states from Linux kernel). When a CPU governor dynamically changes the CPU frequency based on the load, readings of the CPU frequency are a proxy of CPU usage patterns. In systems with multiple processors, the frequency of each processor can be separately set and readings of all CPU frequencies can be used.

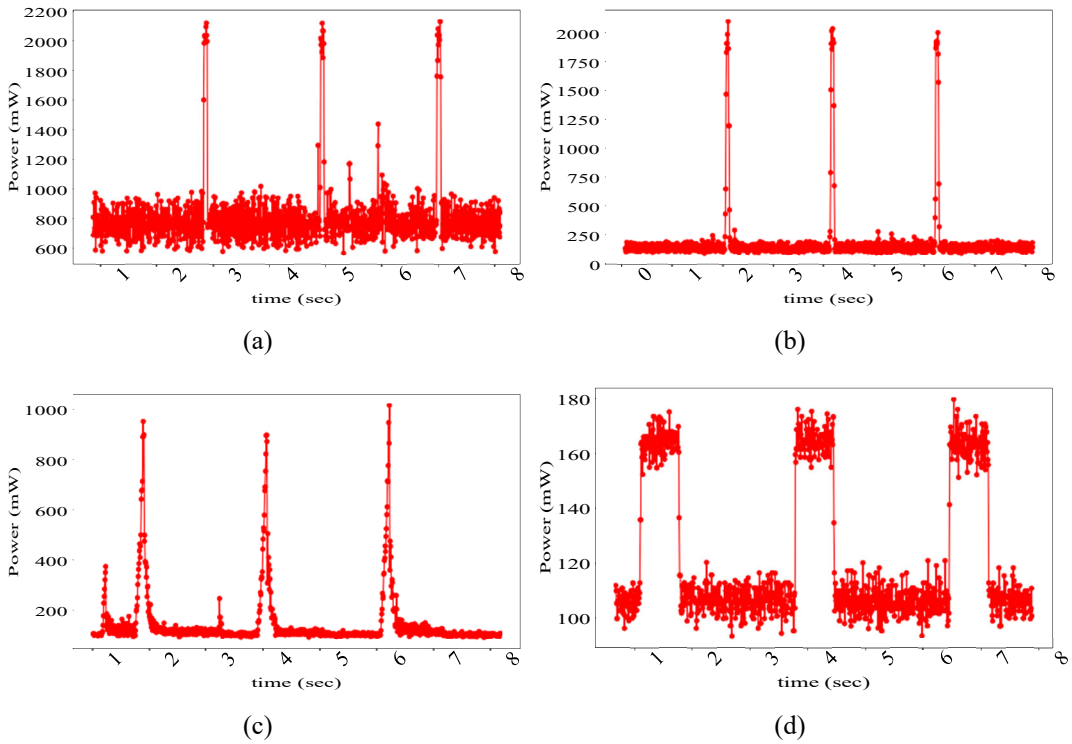


Figure 10: Power consumption profile of a CPU-intensive application in the four DVFS frequency scaling modes. (a) performance, (b) on-demand, (c) conservative, and (d) power-save.

3.2.2 Power side channels

Power consumed by all SBCs can be monitored using on-board power sensors or external power meters. An SBC with a Trojan implanted in its nodes potentially consumes more power relative to a clean node. We studied the behavior of the on-chip power consumed by the SBCs for IO-intensive and CPU-intensive processes. The CPUs in the SBCs can be configured to be in one of four modes of operating frequencies to trade between power consumed and performance. In the on-demand mode, the CPU conserves power even when not running a CPU intensive application. CPU performance in the conservative mode is similar to on-demand mode, except that the power consumed is further minimized in the on-demand mode. In the performance mode the CPU runs full throttle to deliver the maximum performance at all times. The power-save mode conserves power even when running resource intensive applications. These modes are supported by operating the CPU in different ranges of operating frequency. One can switch between them using the Dynamic Voltage and Frequency Switching (DVFS) regulator in the SBCs. OS kernels in the SBCs expose the DVFS regulator via the scaling governor software that one can use to adjust CPU frequency and switch between the states.

Depending on the PCB, voltage and current measurements could be obtained using an on-board sensor or using an off-board sensor that measures power draw from the power source. Some COTS PCBs with on-board power sensors provide separate power measurements for the different modules in the PCB (e.g., separate measurements for GPU and CPU). Due to on-board voltage regulators, the voltage measurements have minor variations while the current measurements show

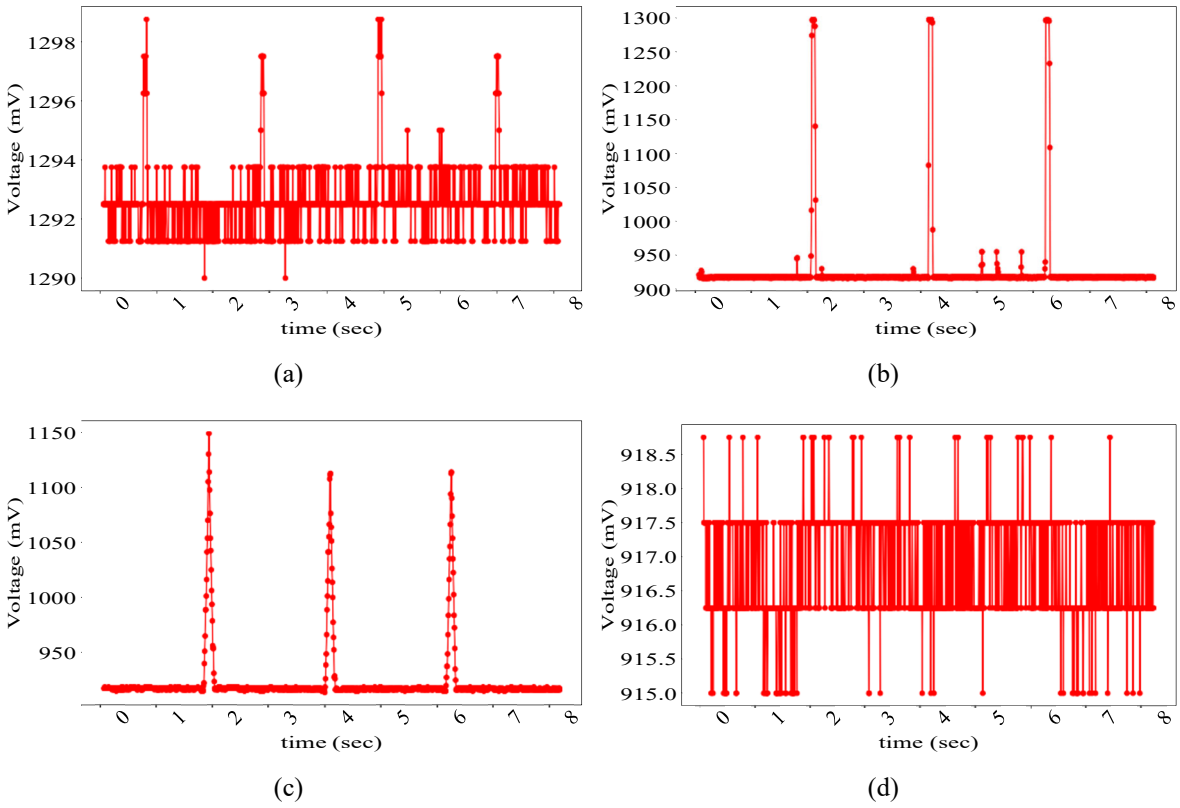


Figure 11: Voltage values when running a CPU-intensive application in the four DVFS modes. (a) performance mode, (b) on-demand mode, (c) conservative mode, (d) power-save mode.

larger variations depending on the time-varying activity.

We studied the power variations in these operating frequency modes of the CPUs in all SBCs. Figure 10 shows the variations in the CPU power on the XU4 SBC. The on-chip power variations in the Figure 10(c) depicts the power consumption when the CPU is in the conservative mode. The periodic power spikes indicate short bursts of increase in CPU performance, followed by power conserving intervals. The on-demand mode in Figure 10(b) follows similar trends. Power consumption plots in Figure 10(d) indicate that the bursts of high power consumption continues for longer intervals than in the prior two modes. In contrast to these three modes, the performance mode in Figure 10(a) sets the CPU frequency to highest levels irrespective of CPU workloads enabling highest performance. This is further validated by Figure 10(a), where peaks in power are up to 2200 mW, the maximum among the power measurements in the four modes.

Aiding Trojan detection: Each performance mode in the CPU exhibits unique patterns in the power side channel. These patterns (periodic peaks) in the measurements for every pair of SBC connections form unique features of a connection. A Trojan in the connection will perturb these measured features, resulting in delays between the periodic peaks. Modeling the features of a connection help identify the perturbations as anomalies. In this way, different connections between SBCs and their measured features model the behavior of the testbed, where perturbations are identified as anomalies possibly due to Trojans.

3.2.3 Thermal side channels

Temperature fluctuations across the various regions of an SBC are sensitive to the workload running on the SBCs and peripherals used. Thermal side channels can expose a Trojan especially when it is active for a substantial duration. Temperature measurements are also an alternative source to gain information about the power consumption of a SBC. SBCs in our testbed have thermal sensors to measure the chip temperature. We measured variations in chip temperature in the four different states of CPU (on-demand, performance, conservative and power-saving modes). Moreover, we measured variations in temperature during the execution of IO-intensive and CPU-intensive processes. Plots in Figure 12 show the variations in thermal measurements of a CPU-intensive process in the four CPU modes. The CPU modes of operation are controlled by frequency scaling governors managed by DVFS regulators described earlier.

Aiding Trojan detection: Temperature measurements reveal unique patterns of CPU behavior not uncovered by the power measurements. In the on-demand mode in Figure 12(d), the CPU of an SBC maintains a constant temperature while the power consumption exhibits a wide range of variation. Temperature variation follows a similar trend in performance and on-demand modes in Figure 12(a) and Figure 12(b). The conservative mode has higher temperature variation than the other modes with an average of 45 celsius. These unique patterns in thermal side-channel measurements for a SBC are used to model the behavior of the SBC in addition to aforementioned side-channels. Deviations from the model in the form of additional delays between measurements could be anomalies, a possible indicator of presence of a Trojan in the system.

Similar to power readings, temperature measurements indicate the level of activity of components/subsystems in a PCB. Depending on the PCB design, temperature readings could be available from on-board sensors and could provide measurements for the different components on a PCB. Separate temperature measurements could be available for each core of a multi-core processor. The temperature side channel can be viewed as a low-pass filtered version of power side channel; rapid transient fluctuations in the power signal appear in an averaged out form in temperature side channel.

A cooling fan (if present) is an auxiliary indicator for CPU temperature. Typical governor policies adapt the fan speed based on the temperature (e.g., increase fan speed in steps based on temperature threshold).

3.2.4 Timing side channels

Timing siding channels (Connection setup time, SBC-to-SBC latency, and throughput) can monitor inter-SBC communication patterns. Measurements of SBC-to-SBC data transfer latency, connection setup latency, and network throughput provide useful information on the inter-SBC communication characteristics as shown in Figure 14. The number of hops indicate the number of intermediate nodes in the SBC-to-SBC paths. A 1-hop implies a direct connection between SBCs without any intermediate SBCs/router nodes. From the plots in Figure 14(e) and Figure 14(f), the network throughput characteristics of the 1-hop connections are distinct from that of the 2- and 3-hop connections. Further, the 2- and 3-hop connections have similar trends in network throughput. These insights can inform introduction of Trojan SBCs without being detected.

An example server-client implementation using MPI is shown in Figure 13 and is used to measure side-channels and understand the behavior of the peripherals of the SBCs. The MPI

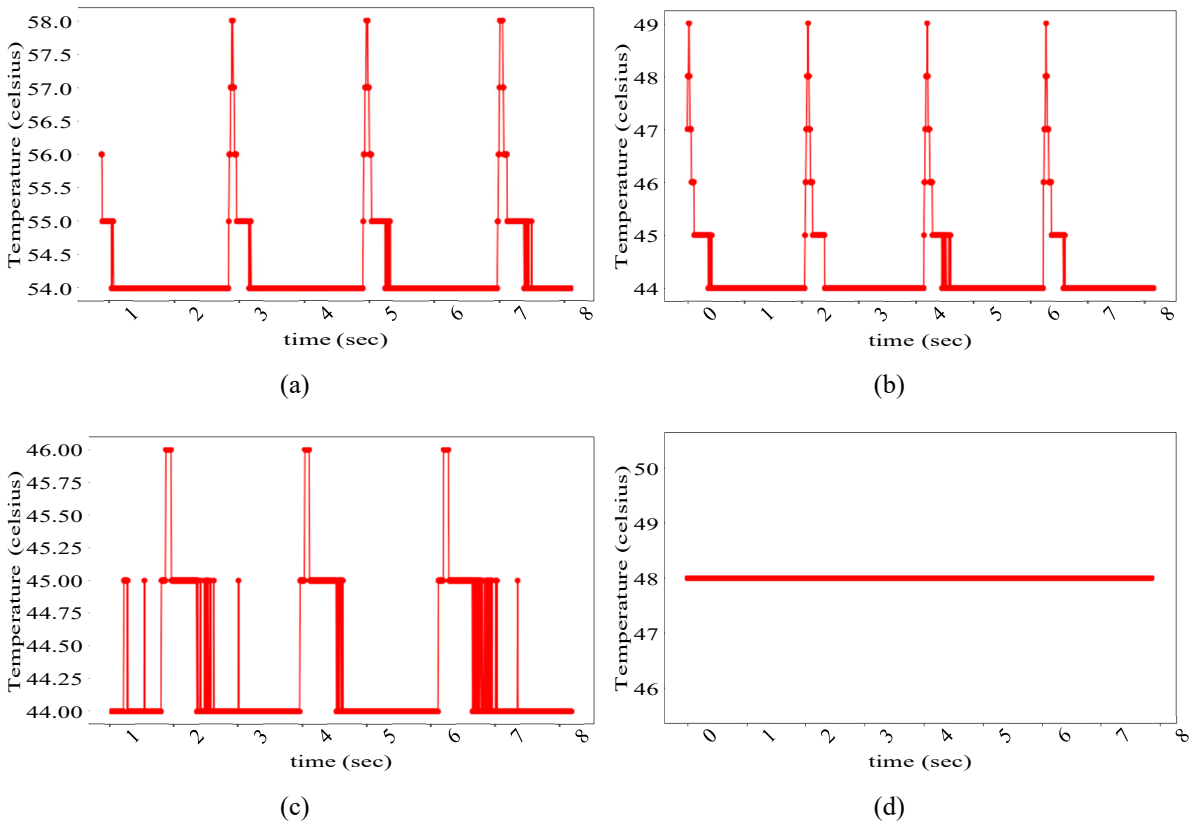


Figure 12: Temperature readings obtained by running the CPU-intensive application in the four scaling frequency modes by DVFS.

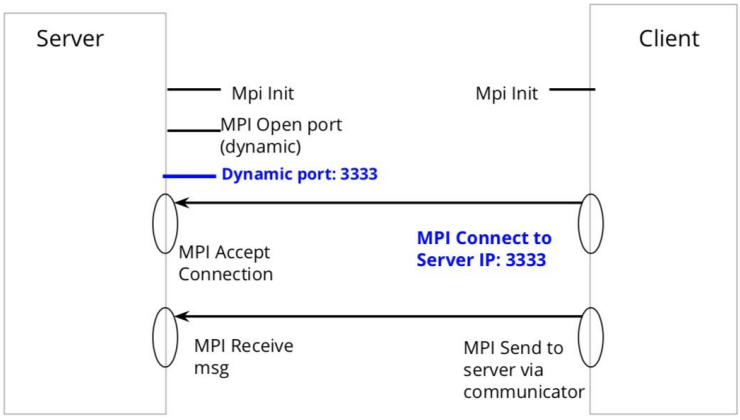


Figure 13: Server-client connection sequence in MPI communication protocol.

protocol was relatively easy to implement and had no packet losses when transmitting variable size data allowing collection of reproducible measurements across multiple runs. Also, the sockets used by MPI are dynamically assigned enabling automation.

The large difference between 1-hop and multi-hop connections (2-,3-hop) can be explained by the variability of latency in the ethernet controllers on the SBCs. The variability in the latency of ethernet controllers in a SBC depends on the SBC on the other end of the connection and the number of hops. This is verified by measuring the throughput of the MC1-2-to-MC1-4 connection via 2 hops in Figure 14(e). The average throughput in this 2-hop case is 500 Mbps while the throughput of the MC1-2-to-XU3 connection via 3 hops is 92.5 Mbps on average. Figures 14(f) and 14(e) confirm that the communications between different SBCs do not have the same impact on their network throughput and depends on the performance of the network controller in the SBC. These insights into the unique characteristics of pairwise connections between the differentSBCs can be used as properties of an integrated system that a defender can exploit for anomaly detection (possibly due to Trojans). Besides the ethernet controller, we experimented with the other communication protocols supported by the SBCs including UART, SPI, GPIO, USB, I2C, and ADC.

As discussed above, multiple side channel measurements of on-SBC activity and communications between SBCs are collected, which under each set of applied controlled excitations, provides a system snapshot that in the context of the graph structure of the interconnected multi-SBC testbed can be viewed as node and edge features. The snapshots in Figure 15 are computed using pair-wise measurements over a time window collected by running instrumented test codes. In Fig. 15, we plot the 2D frequency distribution of communication-based side channel measurements – network throughput, network latency and connection setup time. Fig. 15(a) shows the side channel characterization for the same inter-SBC connection (Odroid MC11↔Odroid MC1). Fig. 15(b) represents the communication characteristics for different inter-SBC connection (Odroid MC1↔Odroid XU3). The 1-hop distance refers to two SBCs directly connected in the same VLAN. Likewise, 2-hop and 3-hop distances refer to SBCs that are separated by one or more routers. The plots show a distinguishable snapshot of frequency distribution based on these hops. The visualizations consolidate the fact that throughput is inversely proportional to hop distance, while network latency and connection setup time increases with the hop distance. Absent any prior data, a defender can measure these communication side channels and compare similar connections to check the presence of Trojans in one or more paths.

Aiding Trojan detection: The timing side channels create unique patterns in SBC-to-SBC connections. These patterns can fingerprint these connections and the SBCs. Introduction of Trojans between SBCs may perturb these patterns. For example, the timing patterns in a connection are influenced by the SBCs and the number of hops between the SBCs. Delays introduced into the timing measurements possibly due to Trojans or anomalies may shift the cumulative distribution of these measurements.

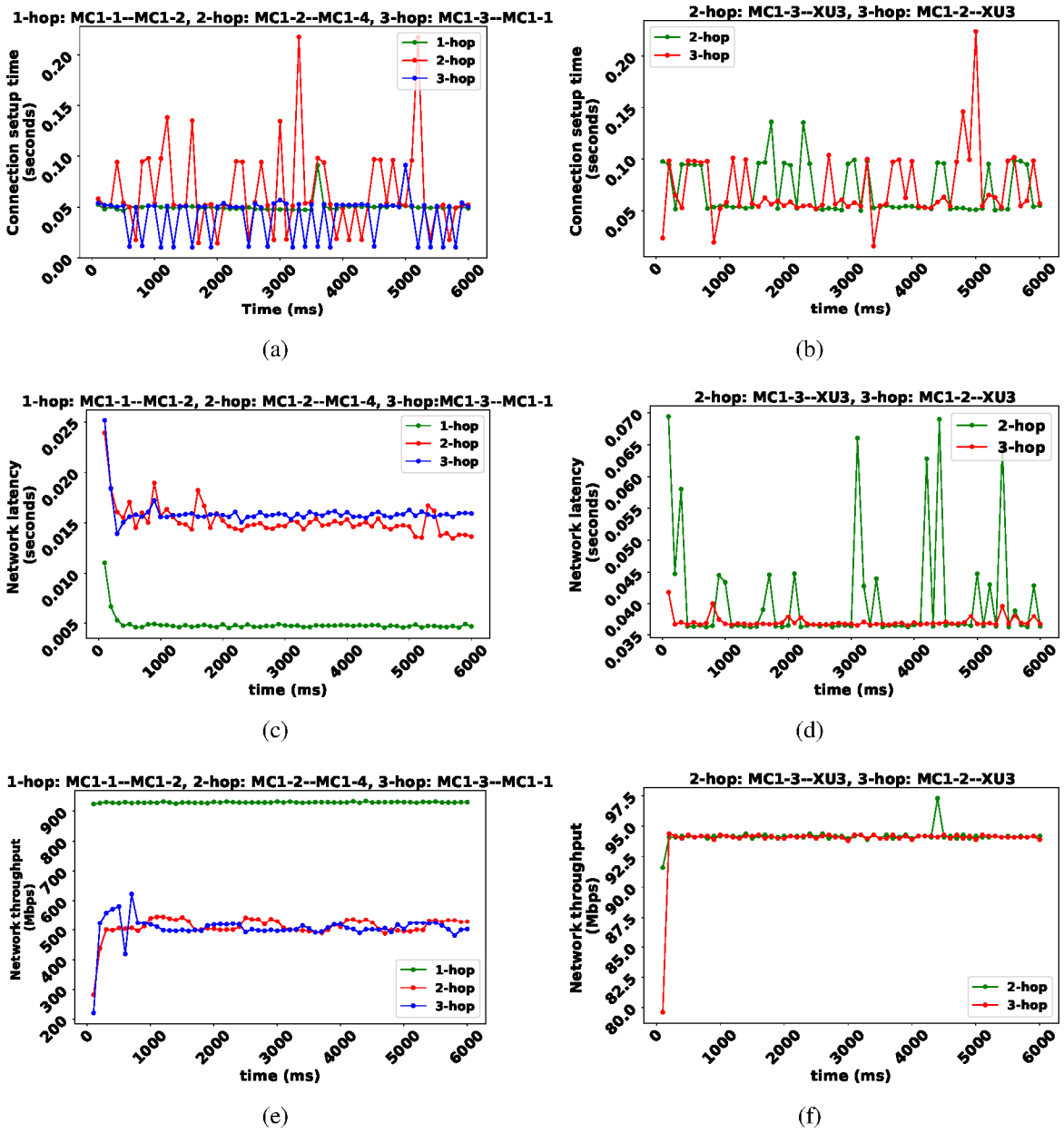
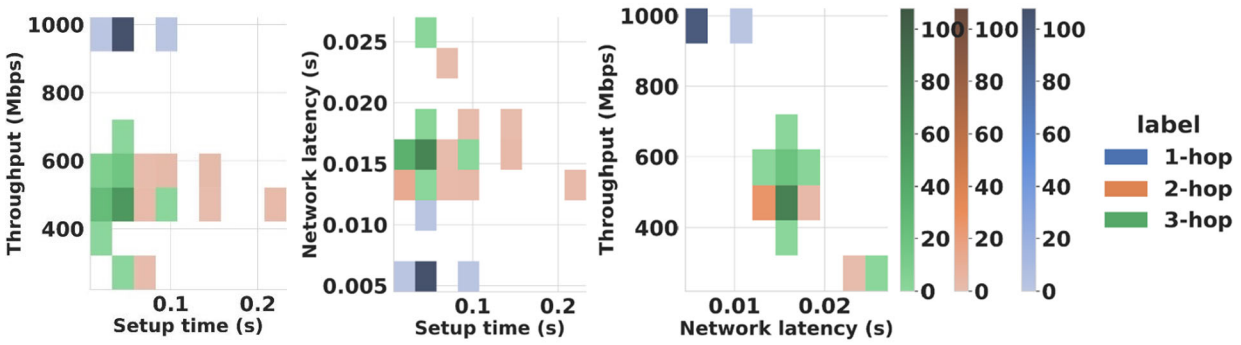
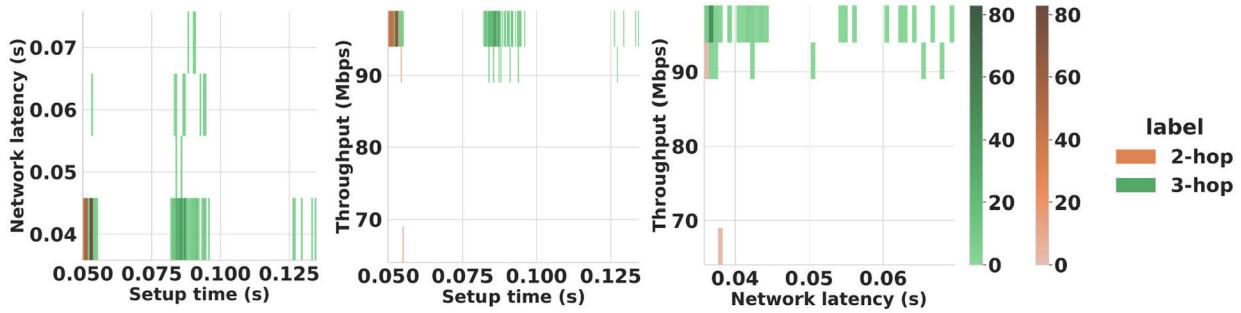


Figure 14: Time-series plots of timing side-channels: connection setup time, network latency, and network throughput. Variations in the connection setup time are shown in (a) for the three pairs of devices and (b) for two pairs of devices connected via 1-hop, 2-hop, and 3-hop. Variations in the network latency are shown in (c) for the three pairs of devices and for the three pairs of devices and (d) for two pairs of devices connected via 1-hop, 2-hop, and 3-hop. for two pairs of devices connected via 1-hop, 2-hop, and 3-hop. Variations in the network throughput are shown in (e) for the three pairs of devices and for the three pairs of devices and (f) for two pairs of devices connected via 2-hop, and 3-hop.



(a) Network latency vs throughput vs setup time for same inter-SBC connection (Odroid MC1s)

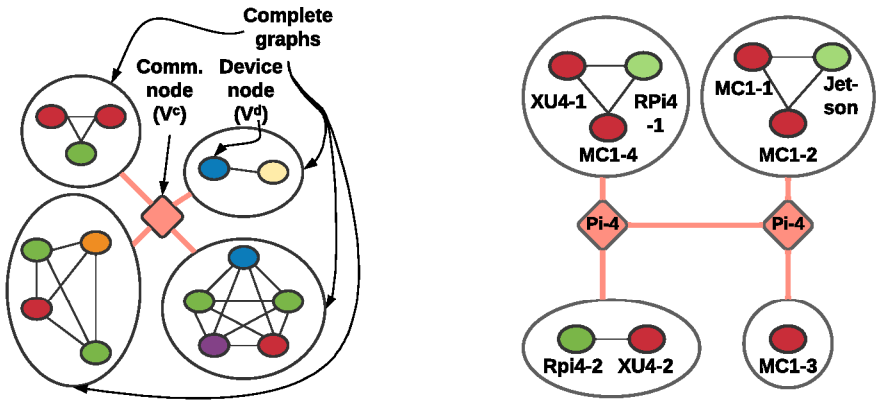


(b) Network latency vs throughput vs setup time for different inter-SBC connection (Odroid MC1 and XU3)

Figure 15: 2D frequency distributions of timing side-channels for connections.

3.3 Equivalence/Invariance Based Trojan Detection

We develop a golden-free baseline modeling and anomaly detection framework to enable detection of Trojans in a multi-PCB COTS system. Given (some) off-line information on the target integrated multi-PCB COTS system design and on-line access to this system integrated by a possibly malicious third party, we seek to detect Trojans in the system by identifying structural anomalies in a graph representation. The problem statement is as follows: Given a graph topology of the system and off-line information of PCB components, find structural anomalies in nodes in the graph and paths between them, which are expected to behave similarly. The defender does not need access to measurements of side channels from a known-good integrated system or from individual known-good components prior to system integration, but instead utilizes design information (e.g., system schematics, functionality specifications, code execution models, communication models, etc.). Such design information enables identification of two types of expected system properties (i.e., predicates to be verified or falsified to demonstrate likely absence or presence of Trojans): firstly, expected side channel characteristics to the fidelity attainable using only the design information; secondly, consistency properties in the integrated system. For example, given an interconnected multi-board testbed as in Figure 4, the expected interconnection graph topology yields sets of homomorphic edges, i.e., sets of device connections that would be expected to have similar properties such as network latency. To validate or invalidate either of these types of system predicates, the instrumentation and side channel collection framework developed in the context of our experimental testbed enables automated application of the following three steps: application



(a) Cluster graph (b) Topology

Figure 16: Graph model of a multi-PCB system: (a) Cluster graph around *communication nodes*, (b) Graph $\mathscr{G}$.

of specified hardware/software fuzzing$\longrightarrow$side channel collection (**the snapshot**)$\longrightarrow$ predicate satisfaction likelihood estimation.

3.3.1 Graph Model of a COTS PCB System

Consider the COTS system graph as $\mathscr{G} = \{\mathscr{V}, \mathscr{E}\}$, where $\mathscr{V} = \{v_1, v_2, ..., v_N\}$ is the node set, and $\mathscr{E}$ the edge set. A node is a PCB and an edge denotes connectivity between PCBs. The connectivity of this graph is defined by the adjacency matrix $\mathbf{A} \in \mathbb{Z}^{N \times N}$, where $[A]_{i,j} = 1$, if $(v_i, v_j) \in \mathscr{E}$. The PCBs can take on application-specific roles. In our experimental studies, the PCBs either performs specific tasks (called dedicated PCB) or function as routers to communicate between dedicated PCBs. We classify nodes in $\mathscr{G}$ into: 1) *Device nodes* v^d performing dedicated computations, 2) *Communication nodes* (routers) v^c. In our COTS instantiation, a PCB device node v_i^d ($v_i \in \mathscr{V}$) associates with one communication node v_j^c ($v_j \in \mathscr{V}$).

First, we form *cluster graphs* of PCB device nodes and connect each cluster graph to a communication node. In a multi-PCB COTS system, a cluster graph corresponds to the set of PCBs grouped to do the same task. All PCBs in a cluster graph use point-to-point communication without intermediate routers. Figure 16a shows a cluster graph of PCB device nodes and a router communication node. PCB device nodes belonging to different complete graphs of same/different clusters communicate through the communication nodes. The cluster graph formed around the communication node v_i^c is $\Pi(v_i^c)$. The communication nodes interconnect to form a network of cluster graphs. Figure 16a shows such a network of PCB device and communication nodes. $\mathscr{G}$ is an interconnection of cluster graphs formed around communication nodes.

3.3.2 Similarity Metrics

Post collecting side channel information for all PCB devices (nodes of G) and connection between various PCBs (edges of G), we need to compare statistical measures obtained from the side-channel data which are expected to be equivalent based on design information. We use three similarity

metrics to satisfy algorithm derived properties of equivalence on nodes and edges.

The Bhattacharyya distance (ρ):

measures the divergence between two unknown probability distributions $P(X)$ and $Q(X)$ over the random variable X. X should span over the same range of values for both distributions P and Q. Formally, the Bhattacharyya distance:

$$\rho(P,Q) = \int_a^b \sqrt{P(X)Q(X)}\, dX$$

where $[a, b]$ is the range of values X can take. Bhattacharyya distance can vary from 0 (distinguishable) to 1 (identical).

The % deviation of average (σ_{pd}):

measures deviation of sample mean (μ) of a target multivariate distribution P_{target} against a reference distribution P_{ref}.

$$\sigma_{ref}^{P_{target}} = \frac{\mu^{P_{target}} - \mu^{P_{ref}}}{\mu^{P_{ref}}} \times 100\%$$

A higher value of σ_{pd} indicates target dataset P_{target} is far away from the reference dataset P_{ref}.

Dynamic Time Warping (DTW):

measures the distance between two discrete time-series signals $X(n)$ and $Y(n)$ of equal/unequal length by finding an optimal alignment between them. The technique computes the cost $c_p(X,Y)$ of an alignment P using the equation:

$$c_p(X,Y) = \sum_{l=1}^{L} c(x_{n_l}, y_{n_l})$$

where $c(x_{n_l}, y_{n_l})$ is the local cost of l^{tn} step of L-length alignment P. The DTW distance $DTW(X,Y)$ between X and Y is defined as the minimal total cost of optimal alignment $p*$ among all possible alignments.

$$DTW(X,Y) = c_{p*}(X,Y)$$
$$= min\{c_p(X,Y) | \text{ p is an alignment } \}$$

3.3.3 Trojan Detection Methodology

While the defender has no *a priori* access to a golden integrated PCB system or even the side channel data of the PCBs in the golden configuration, they do have design information (e.g., PCB system schematics, functional specifications of the devices and the communication interfaces). This offers the defender a foothold to discover predicates (i.e., expected system properties) based on consistency characteristics, relative to the side channels in the PCBs (graph nodes) and the side

channels between PCBs (PCB-to-PCB connection graph paths). Given a cluster graph of a multi-PCB system in Figure 16(a), the graph topology yields homomorphic paths (i.e., sets of device connections) that are expected to have similar properties in the communication side channels.

To validate or invalidate these properties, we devise an algorithmic approach as follows: Use off-line information to color the PCB nodes and collect side channel measurements (such as HPCs and thermal) → Bucket connection paths → Collect communication side channel measurements → Check satisfaction of the properties (using the algorithm discussed further below in this section). The designer/defender runs secret custom test codes having tunable hardware knobs such as CPU operating frequency and including CPU/IO/memory intensive workloads and cache usages and collects the side channels by controlled fuzzing on hardware knobs. For the attacker, these secret codes and hardware knobs are moving targets.

We first formulate our invariance hypotheses based on COTS Graph *G* on various nodes and edges. To detect Trojans in a golden-free setting, we run application test codes and capture the side channels from the PCBs that are part of the interconnected multi-PCB COTS system. We willuse invariance hypotheses to point to anomalous components (hardware, firmware, and software)or connections.

Node coloring

After mapping the PCB-based COTS system to a COTS graph G, the goal is to identify suspicious behavior in the integrated multi-PCB system. In such a multi-PCB COTS system, multiple PCBs with same/similar technical specifications and networking interfaces are present. Such PCBs can be a cluster. Since the designer is aware of the intended system design, they have prior knowledge of G. We color all PCB nodes with similar system-on-chip, peripherals, and networking interfaces and configurations with the same color. In Figure 16(b), all odroid PCBs (XU4 and MC1) are coloured red, and the Raspberry Pi4 and NVIDIA Jetson PCBs are colored green. Formally, the set of all $v_d \in V$ is partitioned into m subsets each marked with a color. Routers are not colored asthey are responsible only for communication.

Bucketing of Paths

We find identical paths in the COTS graph G. Paths are identical for any two pairwise connections if the source and destination nodes have the same color and the connections have the same path length. We enumerate all pairwise connections in G, and apply the bucketing algorithm 1. We collect communication side channels from pairwise connections in the buckets and check their properties for golden-free detection of anomalies possibly due to one or more Trojan implants.

Invariance Properties

Following node coloring and path bucketing, we have sets of device with identical characteristics, and sets of buckets with identical paths. Below, we state types of system properties that can be inferred from design information and apply them on side channels of same color nodes and paths in the same bucket to detect anomalies.

Property 1. *Path invariance: For each bucket b_k in connection buckets B, all paths in b_k should*

Algorithm 1: Bucketing algorithm

Input: COTS graph G, colored nodes $V_C = \{V_{C1}, ...V_{C_m}\}$
Output: Connection buckets B, Connection set S

$B \leftarrow \varphi$; $S \leftarrow \varphi$;
for $edge < V_i^d, V_j^d > \in E$ do
  $C_{source} \leftarrow C_k$, if $V^d_i \in V_{C\ k}$ # Assign colour to source node
  $C_{dest} \leftarrow C_l$, if $V^d_j \in V_{C\ l}$ # Assign colour to destination node
  $P_{i,j} \leftarrow Dijkstra(V_i^d, V_j^d)$
  path-length $\leftarrow$ $length(P_{i,j})$
  $conID_{i,j} \leftarrow <C_{source}, C_{dest},$ path-length>
  if $\exists\ conID_{i,j} \in S$ then
    $b_{conID_{i,j}} \leftarrow b_{conID_{i,j}} \cup P_{i,j}$, $b_{conID_{i,j}} \in B$
  else
    $S \leftarrow S \cup conID_{i,j}$; $b_{conID_{i,j}} \leftarrow P_{i,j}$; $B \leftarrow B \cup b_{conID_{i,j}}$
return B, S

exhibit similar communication side channels, i.e.,

$$SC_{P_m} \equiv SC_{P_n} \forall P_m, P_n \in b_k, b_k \in B$$

where SC denotes communication side channel measurement.

Corollary. *Consider the COTS graph in Figure 16b. Based on the communication specifications of devices, we color XU4 and MC1 as red and Raspberry Pi-4 and NVIDIA Jetson as green. We identify the bucket having red source device node, and green destination device node that are one hop away (RG_1). From the graph, there are five such connections: P_1: MC1-4→Rpi4-1, P_2: XU4-1→Rpi4-1, P_3: XU4-2→ Rpi4-2, P_4: MC1-1 →Jetson and P_5: MC1-2 →Jetson. We capture three connection side channels– setup time, latency, and throughput– for such pairwise connections. Since P_1, P_2, P_3, P_4 and P_5 are in the same bucket, the sampling distribution of connections side channels should be statistically invariant. We use Bhattacharyya distance (ρ) to measure the similarity of side channel distribution of connection setup time (S), latency(L), and throughput(T). Given a given pair of connections, an instance where ρ < acceptable threshold η is flagged as anomalous. Formally, this path invariance is:*

$$\rho(L_{P_m}, L_{P_n}) > \eta_L \wedge \rho(T_{P_m}, T_{P_n}) > \eta_T \wedge \rho(S_{P_m}, S_{P_n}) > \eta_S$$
$$\forall P_m, P_n \in RG_1$$

where η_L, η_T, and η_S are threshold Bhattacharyya distances for Latency, Throughput and Setup Time, respectively, set by defender below which the connection is considered suspicious.

Property 2. *Node invariance: In COTS graph G, device nodes with the same color, i.e., PCBs with same processor/SoC configuration, should exhibit similar side channel characteristics,*

$$SC_{V_i^d} \equiv SC_{V_j^d} \forall V_i^d, V_j^d \in V_{C_k}^d, V_{C_k} \in V_C$$

where SC is side channel characteristics of the devices.

Corollary. *In the COTS graph G, the Odroid MC1 nodes have the same color. We use percentage deviation* σ_{pd} *to measure similarity of side channels. The HPCs (*HPC_k, $k \in [1, m]$*) and thermal side channel (H) data are collected by running the same code on MC1 nodes. Node invariance holds when* σ_{pd} *of the multivariate side channel data (thermal, HPC) for nodes of the same color are* $<$ *a threshold percentage* δ*. Formally:*

$$\forall V_i^{MC1}, V_j^{MC1}; i, j \in [1, 4]$$
$$p = [HPC_k, H] \text{ for } V_i^{MC1}, q = [HPC_k, H] \text{ for } V_j^{MC1}$$
$$\sigma_q^p < \delta.$$

A violation of this equation indicates that node invariance property failed for the MC1 devices.

3.4 Simulator-Trained ML-Based Trojan Detection

3.4.1 Strategies for Golden-Free Design/Simulator-Driven Detection

Trojans of various kinds (hardware, software, firmware) can be inserted during manufacturing or integration by untrusted third parties. Hence, in real applications, a golden 'clean' or 'known-good' system- under-test can not be assumed to be available. However, it is reasonable to assume that *design information* is available. This design information will come in various forms depending on the context being addressed, such as:

- behavioral specifications of system and/or its components, e.g., input-output specifications, behavioral operation models.
- source code of the kernel, application, etc.
- datasheets for system components, e.g., datasheets for communication interfaces.

Here, the design-time information would be directly obtainable if the designer is performing first-party tests, or it may be obtained from the designer by third parties who are evaluating the board. As we assume that the threat is originating from an *untrustworthy third party in the supply chain* we can consider that the design files are trustworthy: that is, any Trojan does not originate from the original designer. This trustworthy design-time information may be used in golden-free Trojan detection.

To utilize the design-time information of the various kinds outlined above for golden-free Trojan detection, the architecture shown in Figure 17 was developed. Time series of side channel measurements from the system-under-test are evaluated using two complementary techniques to probabilistically determine if the system-under-test contains a Trojan. Both these techniques seek to assess to what extent the side channel signals from the system-under-test match the expected characteristics based on the design information. Firstly, the design information is used to implement a simulator to generate data for training the anomaly detector discussed in Section 3.4.2. The purpose of the simulator is to stand in as a proxy for the unavailable golden system and to provide simulated side channel signal traces that while possibly not precisely matching the hypothetical golden system's side channels do still provide an adequate approximation to enable training of the

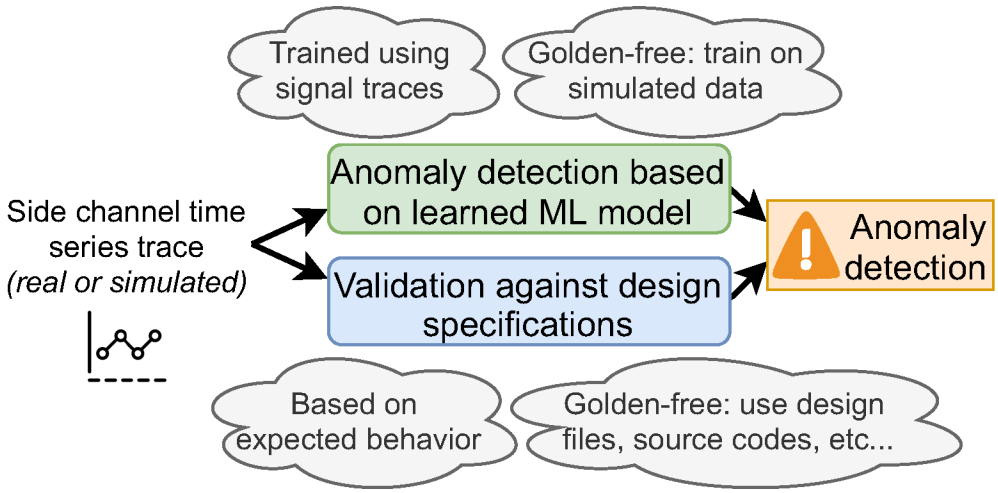


Figure 17: Golden-free Anomaly Detection.

anomaly detector. The implementation of the simulator can draw upon known source code, circuit designs, component-level specifications/data, etc., to enable modeling of the side channel signals. Secondly, the design information is utilized (by the system designer) to formulate probabilistic specifications/rules that would be expected to be satisfied by the side channel signals based on the system design. These rules could utilize, for example, known source code or circuit designs or equivalence/inequivalence properties inferred based on the components in the system design. Given side channel signal traces, the probabilistic evaluation of whether these rules appear to be satisfied by the side channel signals can be used to flag anomalies. The multi-modal side channel based anomaly detection methodology is described in Section 3.4.2. Thereafter, the simulator-based training of the anomaly detector is discussed in Section 3.4.3 and the behavioral analyzer is discussed in Section 3.4.4.

3.4.2 Multi-Modal Side Channel Fusion for Anomaly Detection

Side Channel Data Collection

While the anomaly detection methodology can be applied with a variety of side channels discussed in Section 3.2, we focus on the following digital/analog side channels using Odroid XU4 SBC as a representative PCB: HPCs, Power, CPU load and frequency, Temperature, Fan speed. Each side channel provides a different view into the behavior of the system. Digital side channels such as HPCs provide fine grained information on the CPU operation and can be used to detect Trojan-induced variations in code execution characteristics. On the other hand, power, temperature, and fan speed side channel measurements provide aggregate views of the behavior of the PCB and can detect variations in system-level operation. For example, an additional Trojan component on the PCB might be detectable from power measurements but might not cause any detectable variations in the HPC measurements. On the other hand, subtle code execution modifications due to a firmware Trojan might be detectable by HPC variations but might not cause sufficient power draw variations as to be detectable from power, temperature, or fan speed. The CPU use measurements (load, frequency) complement HPC measurements and analogously to the relation between power and temperature side channels, can be seen as filtered signals related to HPC time

series.

While the different side channels are related to some extent as they provide different views into the same underlying activity of the PCB, the separate measurements of multiple side channels can detect more types of Trojans and more accurately.This means that Trojans that have appreciable impacts on different subsets of side channels can be detected. We hypothesize that by combining side channels we can reduce false negatives by detecting anomalies from the perspectives of the different side channels, and lower the false positive rate by reducing sensitivity to benign random variations. Consider the scenario where the fan speed increases without a temperature or CPU load increase – this would prompt the question 'why has the speed increased?' and could indicate anomalous behavior of the fan.

Given a single or an integrated multi-PCB system, multi-modal side channel based Trojan detection entail collecting time series measurements of side channels, and analyzing them to assign a likelihood score of whether the measurements match the system characteristics. The operating conditions of the system when collecting the side channels can be controlled by the defender. These operating conditions provide "fuzzing" degrees of freedom to excite the system in different operating regimes as discussed further in Section 3.5. This may increase the visibility of Trojans when measuring side channels over a wider operating condition, improving detection accuracy. While the fuzzing degrees of freedom are open-ended, one fuzzing mode of use is running multiple instances of defender-created *test codes* and collect side channel measurements. The test codes can contain a mix of integer/floating-point calculations and memory accesses. By running different numbers and instances of the test code and pinning the instances to different cores in a CPU, the defender can set up a configurable operational condition for the PCB while collecting side channel measurements. Similarly, the defender can fix CPU frequency and fan speed to different values. Controlling these fuzzing modes is a moving target defense; the adversary does not know when and what fuzzing modes will be applied when testing a system for Trojans.

The side channels measure PCB activity from different physical viewpoints and have different sampling rates and granularity. HPC and power are high bandwidth signals (i.e., vary rapidly) and benefit from high sampling rates. Temperature, fan speed, CPU usage, and CPU frequency vary slowly and need lower sampling rates. While measurements from each side channel s can

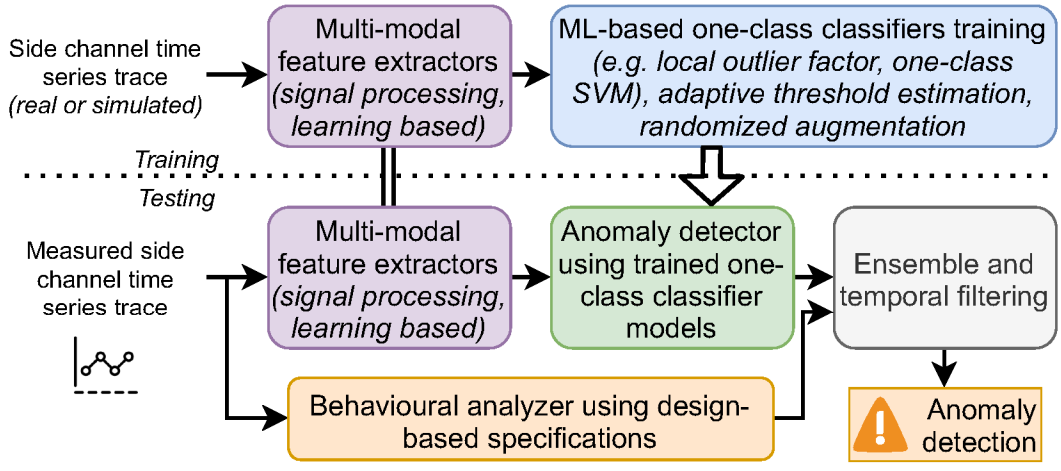


Figure 18: Anomaly detector for single-/multi-PCB systems.

be viewed as a sequence of time stamps $t_{s,1}, \ldots, t_{s,n_s}$ (taken over a common time base for all side channels), they are collected at different sampling rates. The inter-sample time intervals are not uniform (due to process scheduling jitter, limitations of system clock accuracy, off-board nature of measurements). Some side channels are scalar time series (e.g., total current draw) others have vector measurements at each sampling time instant (e.g., HPC side channel has multiple types at each instant).

Feature Extraction

Starting with the multi-modal side channel time series measurements, we window normal vs. anomalous classification to detect Trojans. Let the time interval over which side channel measurements have been collected as $[t_{min}, t_{max}]$, the defender-chosen time window be T and the stride be Δ. The i^{th} time window for evaluation is $[t_{min} + i\Delta, t_{min} + i\Delta + T]$ and the number of time windows is given by $i_{max} + 1$ where $i_{max} = \lfloor \frac{t_{max} - t_{min} - T}{\Delta} \rfloor$. Since, each side channel can have a different sampling and non-uniform inter-sample time intervals, identifying subsets of side channel measurements in each time window can be done using two approaches. First, the time stamps $t_{s,1}, \ldots, t_{s,n_s}$ for side channel s can be used directly and the subset of measurements located in the i^{th} time window can be identified as the samples with time stamps $t_{s,k_1}, \ldots, t_{s,k_2}$ where

$$k_1 = \min_k \{k \in \{1, \ldots, n_s\} | t_{s,k} \geq t_{min} + i\Delta\} \tag{3.0.1}$$

$$k_2 = \max_k \{k \in \{1, \ldots, n_s\} | t_{s,k} \leq t_{min} + i\Delta + T\}. \tag{3.0.2}$$

Second, interpolate the side channel time series measurements such that all side channel signals have a common sampling rate and a uniform inter-sample time interval. We pick the common sampling rate to be the highest sampling rate (either HPC or power) and use a linear interpolation using neighboring samples for any sampling instant. After interpolation, the time windowing will use the same min and max indices for each side channel. In either approach, we obtain *overlapping* time windows if $\Delta < T$, i.e., the time shift between successive time windows is less than the width of the time windows. We use the approaches for complementary purposes; the first for statistical or signal processing feature extraction and the second for deep learning feature extraction.

The algorithmic architecture for the anomaly detection system is shown in Figure 18, including both the components during training-time (i.e., when training the anomaly detector) and during inference-time (i.e., when evaluating a given system for presence of Trojans). As indicated in Figure 18, the anomaly detector can be trained using either real data from a known-good "golden" system or data from a simulator standing in as a probabilistic proxy for an unavailable known-good system. Since we seek to develop a golden-free method in this project, the training data will be generated using a simulator as discussed in Section 3.4.3. However, the training procedure for the anomaly detector itself is identical regardless of whether the training data is obtained from a real or a simulated system.

After time windowing on the training data, low-dimensional features are computed for the side channel time series over each time window using two methods. First, from the subset of side channel measurements over a time window, statistical/signal processing based features such as mean, standard deviation, minimum, and maximum are extracted for each component of each side channel. Next, cross-component features such as cross-correlation and the more general features such as skewness, and kurtosis are extracted, analogous to the feature extraction used for HPC time

series from multithreaded processes in [19]. Second, an unsupervised deep learning method is applied to extract the low-dimensional feature vectors from the side channel measurements over the time windows. The measurements from side channels are combined into a two-dimensional matrix in which columns correspond to side channel components. To enable column-wise concatenation across the different side channels, the side channel time series are interpolated into a common sampling rate. The two-dimensional matrices obtained over the different time windows are used to train a deep autoencoder which has a low-dimensional middle layer representing the latent feature space. The structure of the autoencoder is shown in Figure 19. The autoencoder has an encoder, a latent layer, and a decoder. The encoder has two convolutional layers. Likewise, the decoder has two transposed convolutional layers. There are three maxpooling layers in between the convolutional layers. Each layer has ReLU activations, and mean squared error is used as the loss function to train the deep autoencoder.

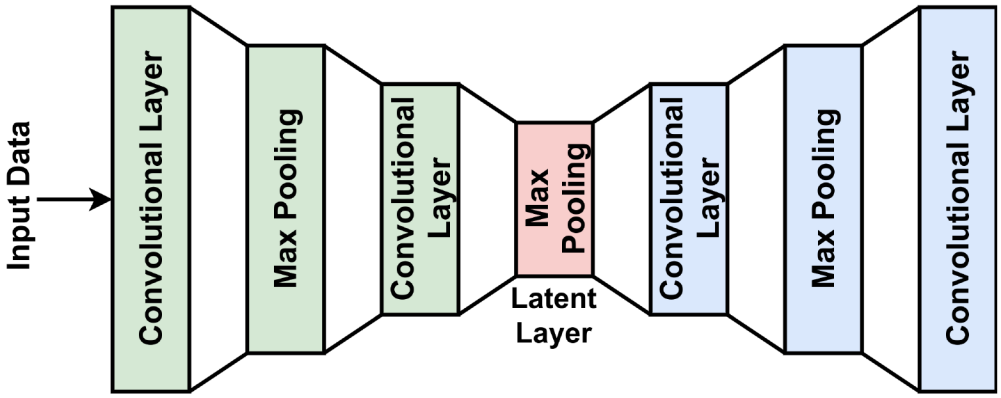


Figure 19: Autoencoder for feature extraction from multimodal side channel time series.

One-class Classifier Training

The low-dimensional feature vectors computed over time windows are used to train a one-class classifier. Absent real or simulated data for a Trojaned system, the classifier is trained only on real/simulated data representing non-Trojaned systems. To make the one-class classifier robust to small random variations (as well as to variations between the simulated and the real system), we add random augmentations to the training data. The random augmentations are constructed by adding zero-mean Gaussian noise to the feature vectors previously computed, with the standard deviation of the Gaussian noise proportional to the magnitudes of the feature values. The original feature vectors and the random augmentations comprise the training data for the one-class classifier.

For one-class classification, approaches such as Local Outlier Factor (LOF) and one-class Support Vector Machine (SVM) algorithms apply. They provide complementary benefits, and to a large extent, can be interchanged. The LOF algorithm estimates the local deviation of a data point with respect to its neighbors (in input feature space) to find anomalies. Hence, the LOF-based approach is faster than one-class SVM which estimates the boundary for the set of all possible normal data points by picking a set of support vectors on the boundary. Either of these classifiers (or other approaches such as Isolation Forest) estimate an outlier score for a data point during testing; the higher this score, the more different the classifier deems a data point to be compared to the

training data. Hence, irrespective of the one-class classifier used, a primary question to address is to determine a threshold for the outlier score above which an anomaly is declared.

The choice of this threshold is crucial. If it is too low, it will cause the anomaly detector to be too sensitive – generating false positives during benign random fluctuations of the ambient environment. If the threshold is set to be too large it can result in false negatives, since small variations caused due to subtle Trojan effects would not be classified as anomalous. We use an adaptive thresholding to estimate an appropriate threshold by computing the outlier scores on a separate validation dataset (based on simulated data using approach in Section 3.4.3) and picking the threshold such that most of the validation data is classified as benign.

Anomaly Detection

When evaluating a system for Trojans during testing, we apply time windowing and feature extraction algorithms from this section to the side channel measurements. The resulting feature vectors over time windows are used by the trained one-class classifier. The normal vs. anoma- lous classifications from the one-class classifier are temporally filtered to generate the final normalvs. anomalous time series. For this temporal filtering, we use a sliding median filter to temporally smooth the normal vs. anomalous classifications. For this, we use a a median filter width W, where W consecutive classifications of normal vs. anomalous are considered in a sliding manner and the majority label determines the final normal vs. anomalous classification. Additionally, as shown in Figure 18, we apply a behavioral analyzer discussed in Section 3.4.4 to determine the likelihood of a Trojan by probabilistically matching the observed behavior against design-time specifications. The normal vs. anomalous classifications over time windows from the anomaly detector and the behavioral analyzer are combined using an ensemble approach by weighting the anomaly detector and the behavioral analyzer classifications and applying the temporal filtering discussed above. After ensemble and temporal filtering, the time series of normal vs. anomalous classifications can be a binary classification using majority voting to determine whether the system-under-test has a Trojan or not. However, the time series of normal vs. anomalous classifications are used in experiments in Section 4.2 to evaluate accuracy, false positives, and false negatives.

3.4.3 Simulator to Generate Golden-Free Training Data

Training of anomaly detectors in Section 3.4.2 relies upon time series data of side channels. The golden-free setting in this project seeks to remove requirement for a known-good system for acquisition of such training data. In the place of such a golden system, we substitute a digital twin based approximation that draws upon the various sources of design information to build a proxy for the unavailable golden system. While the simulator structure will depend intrinsically on the specific design and architecture of the system-under-test, a basic principle is to compose the simulator as a combination of components that each model a part of the overall system and capture a hardware/software interrelationship between different observable side channel signals. The separate components could model, for example, behaviors in device source code that affect mappings between different side channels (e.g., a CPU governor that adjusts CPU frequency based on usage), physics-based/empirical dependencies between side channels (e.g., heating of the CPU cores based on their usage), etc. As a proof-of-concept in this work, we developed the simulator with architecture shown in Figure 20 to model the operation of the PCB (Odroid XU4) used in the ex-

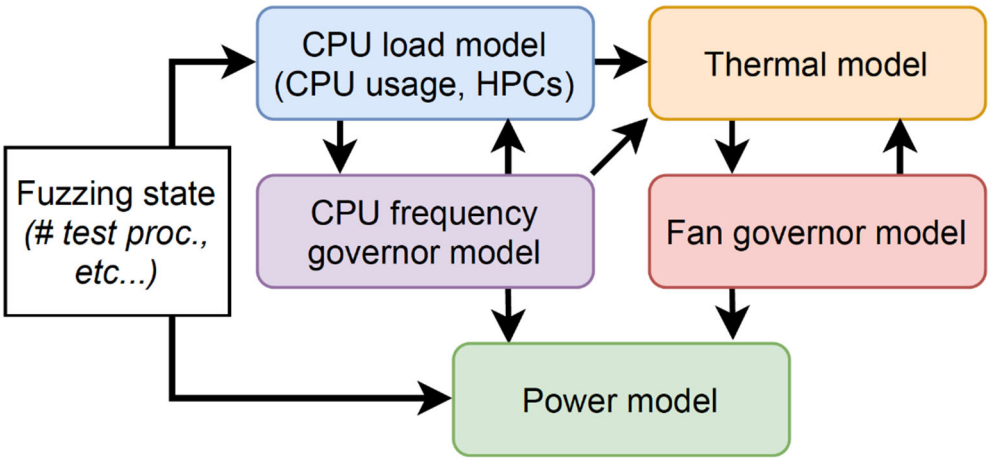


Figure 20: Architecture of simulator for Odroid XU4 for generation of training data for anomaly detector.

perimental studies (Figure 4.2). As an example of fuzzing outlined in Section 3.4.2, we consider a defender-specified test code that can be run some number of times in parallel (with optionally the test processes pinned to cores in the multi-core CPU). Hence, as shown in Figure 20, the simulator takes an external input that specifies the number of test processes. The "fuzzing state" input to the simulator can specify defender-chosen fixed CPU frequencies, number of instantiations of multiple test codes that exercise memory accesses and integer/floating-point calculations. The developed simulator is a composition of the following models:

- Temperature – models variation of temperature of the CPU core as a function of frequency, load, and fan state
- Load – models CPU load (including CPU usage level and HPCs) as a function of the workload command (in this case, number of test processes)
- Power – models variation of power drawn by the system as a function of frequency, workload, and fan speed
- Frequency – models the CPU frequency as a function of CPU load and temperature
- Fan – models variation of fan speed as a function of temperature

Temperature, load, and power model the physics/hardware of the system (i.e., thermal heating based on operating states and workload, CPU load, power draw). On the other hand, the frequency and fan speed model software aspects of the system (i.e., the CPU frequency governor which dynamically varies the CPU frequency depending on load and temperature, fan governor which varies the fan speed based on temperature). Hence, the last two models –governor models for brevity– are essentially deterministic given known source code for the governors. In practice, they are somewhat pseudo-random due to insufficient observability since the internal measurements used by the kernel for the CPU frequency governor for example will differ somewhat from the measurements that are externally accessible. On the other hand, the first three models (referred to as physics models below for brevity) are essentially stochastic since they depend on several other factors such as ambient conditions, presence of other processes, etc.

The implementations of the models utilizes the available source code of the Linux kernel and probabilistic characterizations of the signal interdependencies outlined above. While these probabilistic characterizations could be derived from circuit and processor simulations (e.g., Qemu), we base the derivation of the probabilistic characterizations on empirical data from a separate board for ease of proof-of-concept implementation and to enable focusing on the main challenges addressed in this project. However, we intentionally limit the fidelity and granularies of the probabilistic characterizations and add noise so as to ensure that the derived probabilistic characterizations mimic what could be reasonably obtained from simulation. Specifically, we base the implementations of the models above in terms of conditional probabilities such as $P(\text{load}|\text{workload})$, $P(\text{power}|\text{workload}, \text{frequency}, \text{fan})$, $P(\text{HPCs}|\text{workload})$ etc. To limit the fidelity of the probabilistic characterizations as discussed above, we model these conditional probability distributions in terms of only the means and standard deviations (i.e., simple Gaussian models). In addition, for governor models, we use the kernel source code (which, for example, for the fan model entails a look-up table of temperature thresholds for fan command switching).

While the simulator models specific aspects of the overall system, their choice is motivated by the observation that we want to capture inter-relationships between different side channels measured so that we can then use the simulator as a source of training data for the anomaly detector. While, detailed circuitry of the PCB is not modeled explicitly, the granularity of the simulator adequately generates realistic time-series traces for the side channels. The structure of the data captured from the simulator is such that these data files have identical schema to the data collected from the physical system. Hence, any analysis that can be done on the real data can be done on the simulated data and vice versa. Depending on the architecture of the system being evaluated for Trojans, different component models and different fidelities of the models can be used. For example, SPICE analog models and Synopsys circuit-level models of subsystems on the PCB, QEMU processor emulation models, and communication models (ethernet and UART).

The developed simulator models frequency scaling and power behavior as a function of test workload through a component-based composition of simple models. The simulator includes stochastic variability in component models and initial conditions so that each simulated trace is different. The simulator can rapidly generate large amounts of data for training the anomaly detector. Sample data generated using the simulator is shown in Figure 21 for the quiescent test case (i.e., workload without defender-chosen test processes running, with operating system processes and side channel measurement codes running). Data collected from the Odroid XU4 PCB (with no Trojan) under the same test conditions is shown in Figure 22. The side channel signals considered include HPCs (e.g., number of instructions, branches, cache activity), power, CPU load, frequency, temperature (on-chip and on-board thermal sensors), and fan speed (PWM commands). Data generated from the simulator under workloads specified as 1, 2, 3, and 4, instantiations, respectively, of the defender-chosen test code (with each instantiation pinned to a different core on the A15 processor in XU4) are shown in Figures 23, 25, 27, and 29. We show data collected from the real board under corresponding workload test cases in Figures 24, 26, 28, 30.

While the simulator-generated data does not have the occasional spikes and other stochastic characteristics that are observed in the real data, the simulator-generated data provides a reasonable approximation of the data from the real board and therefore can plausibly stand in as a proxy for an unavailable golden board for training the anomaly detector. As noted above, the stochastic nature of the simulator (both in the component models and the initial conditions) implies that the simulator-generated data is different each time we run the simulator. Hence, the data generated

from multiple runs of the simulator can be expected to provide a reasonable variety of training data to enable the anomaly detector to learn salient trends in the side channel signals to achieve the twin goals: recognize data from a good (i.e., Trojan-free) PCB as normal despite stochastic variations because of ambient environment and mismatches with the simulator; flag data from a Trojan-injected PCB as anomalous by picking up changes in interrelationships between side channels.

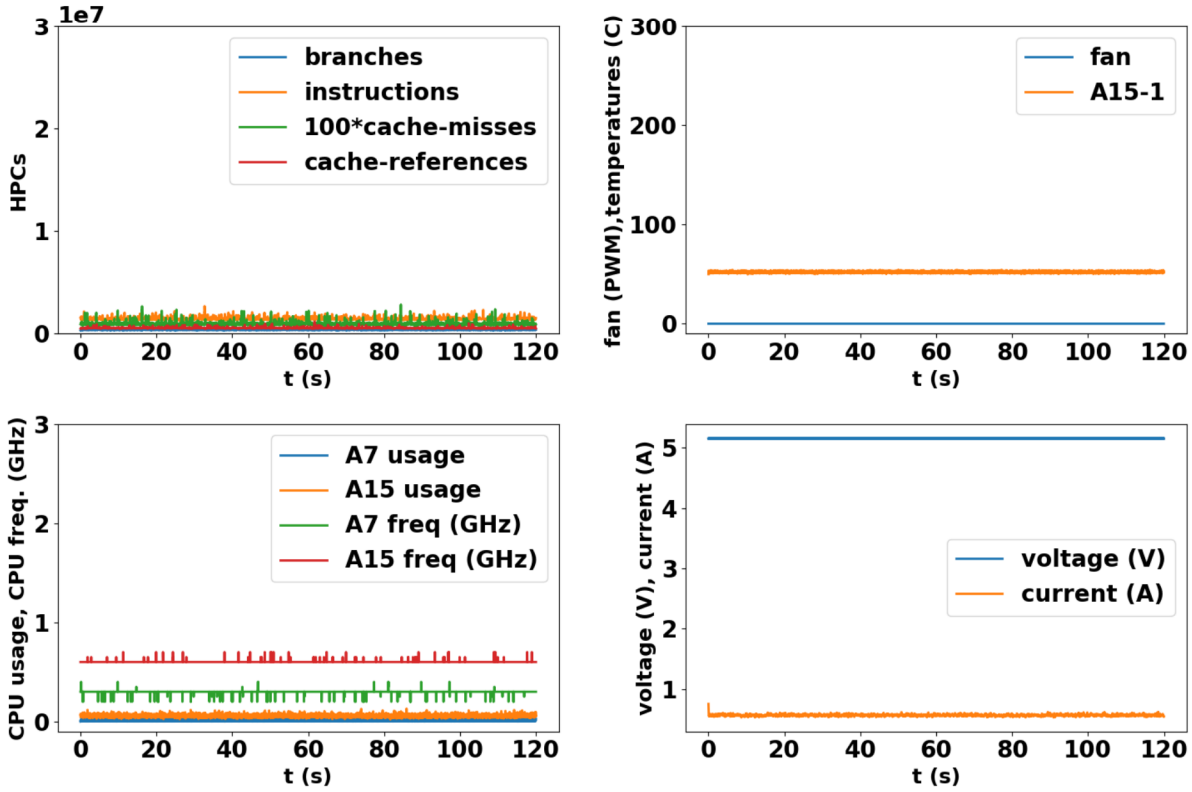


Figure 21: Data Collection from Behavioral Simulator (XU4, Quiescent, No Trojan).

3.4.4 Behavioral Analyzer

While the simulator-based data generation provides training data for the anomaly detector (Section 3.4.2), the component models in the simulator (Figure 20) can themselves be used as a means to validate measured side channel time series data. In particular, the governor models being relatively deterministic (since they model behaviors resulting from software) can be directly verified in measured side channel data. For example, time series measurements of temperature and fan speed can be analyzed to determine whether these measurements appear to be in accordance with fan governor source code that prescribes variation of fan speed as a function of temperature. For component models that are defined through probabilistic characterizations as discussed above, the conditional probability distributions can be used to assign likelihoods to measured side channel data, i.e., to determine the likelihood that the observed measurements would result from a system that was evolving in accordance with the modeled conditional probability distributions. As representa-

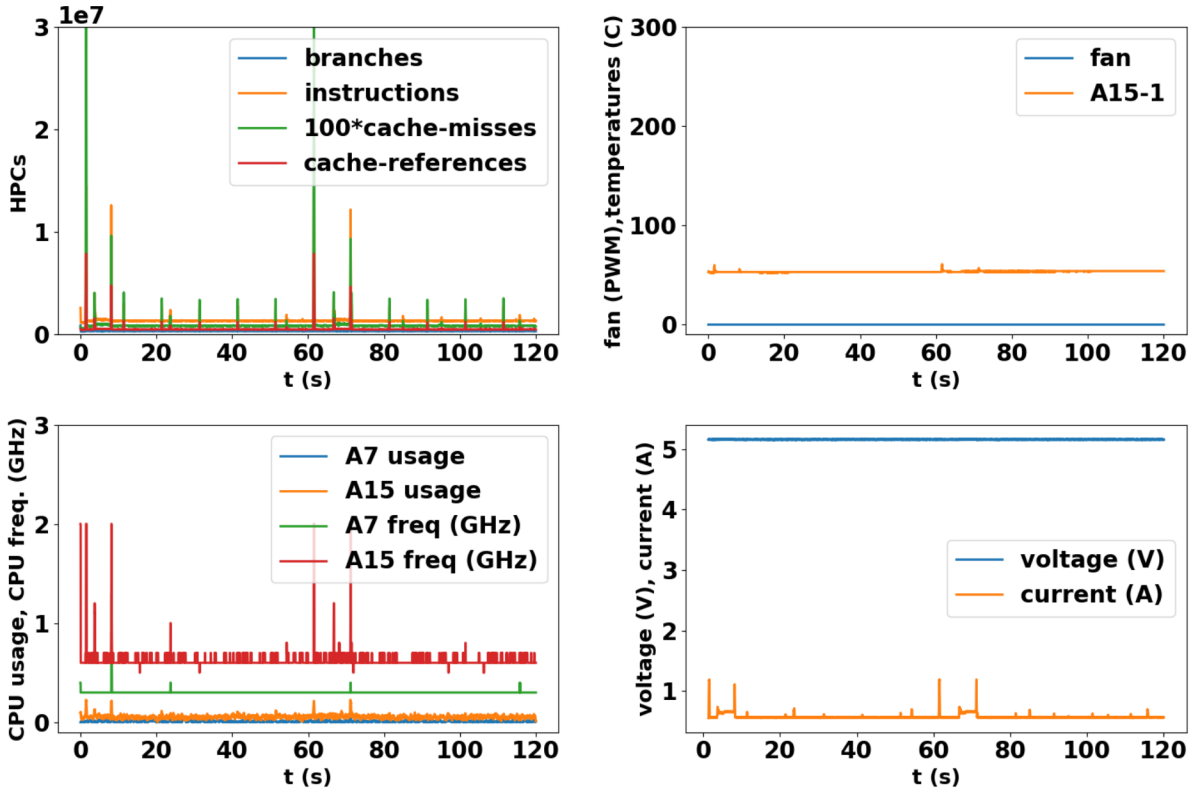


Figure 22: On-Device Data Collection (XU4, Quiescent, No Trojan).

tive examples for the implementation used for experimental studies in this project, we consider the following side channel signal mappings based on the component models in Section 3.4.3:

- Temperature $\longrightarrow$ fan speed: based on the temperature thresholds ("trip points") for fan speed switching defined in the fan governor source code
- CPU usage $\longrightarrow$ CPU frequency: based on $P(\text{freq}|\text{usage})$; the CPU frequency governor on the Odroid XU4 does not implement thermal throttling in the typical configuration therefore removing temperature dependence of the mapping from CPU usage to CPU frequency.

The analysis of the likelihoods that the expected mappings are satisfied in the measured side channel signal data is done over sliding time windows that are defined similar to the formulation (Section 3.4.2) used in the anomaly detector. Since as discussed in Section 3.4.3, even "deterministic" models such as the fan governor are in reality pseudo-random since the underlying measurements used in the governor code will differ somewhat from readings by the side channel measurement code (e.g., due to sampling constraints, variations between kernel and user space, etc.) , the likelihood analysis uses ranges of signal values over sliding windows rather than instantaneous values. Also, a temporal filtering using a sliding average or median filter is used to smooth the time series of likelihood estimates as in Section 3.4.2. An ensemble approach is used to combine the likelihood determinations obtained for the two separate behavioral rules/mappings listed above and with the output of the anomaly detector as discussed in Section 3.4.2.

In general, formulation of behavioral rules describing expected dependencies between side channel signals could draw upon known source code or circuit design to infer that a particular

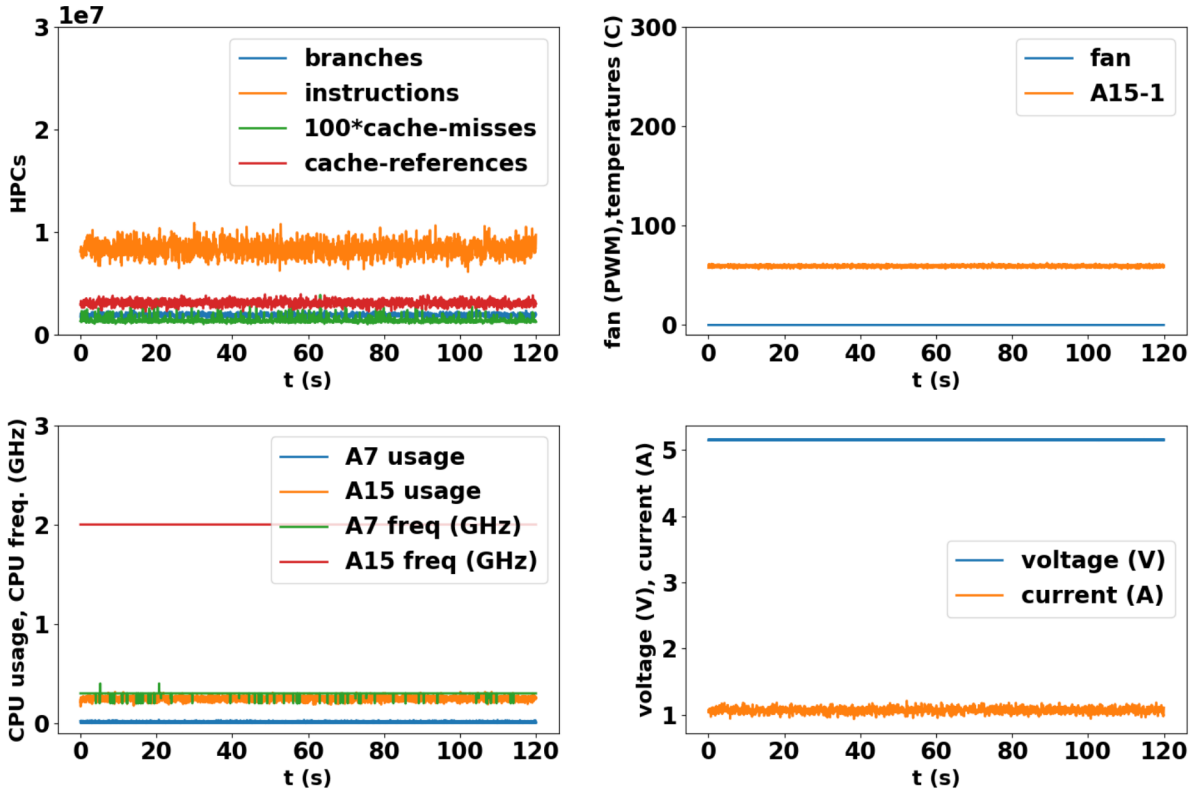


Figure 23: Data Collection from Behavioral Simulator (XU4, Test Process on an A15 Core, No Trojan).

event seen in one side channel should be followed by an event observable in a different side channel (e.g., an increase in temperature above a threshold should be followed by an increase in fan speed within a certain amount of time). As another example, such rules could draw upon known equivalences/inequivalences leveraging the design information (e.g., in a system with two identical/similar devices, side channel patterns could be expected to be similar between the devices; in a system with one device known to be slower than another, inequivalence properties of the side channel patterns of the two devices could be inferred). Such rules could also be viewed as "predictive" in the sense that based on some side channel signals, they enable prediction of properties of some other side channel signals, thereby flagging anomalies if the predicted properties do not hold.

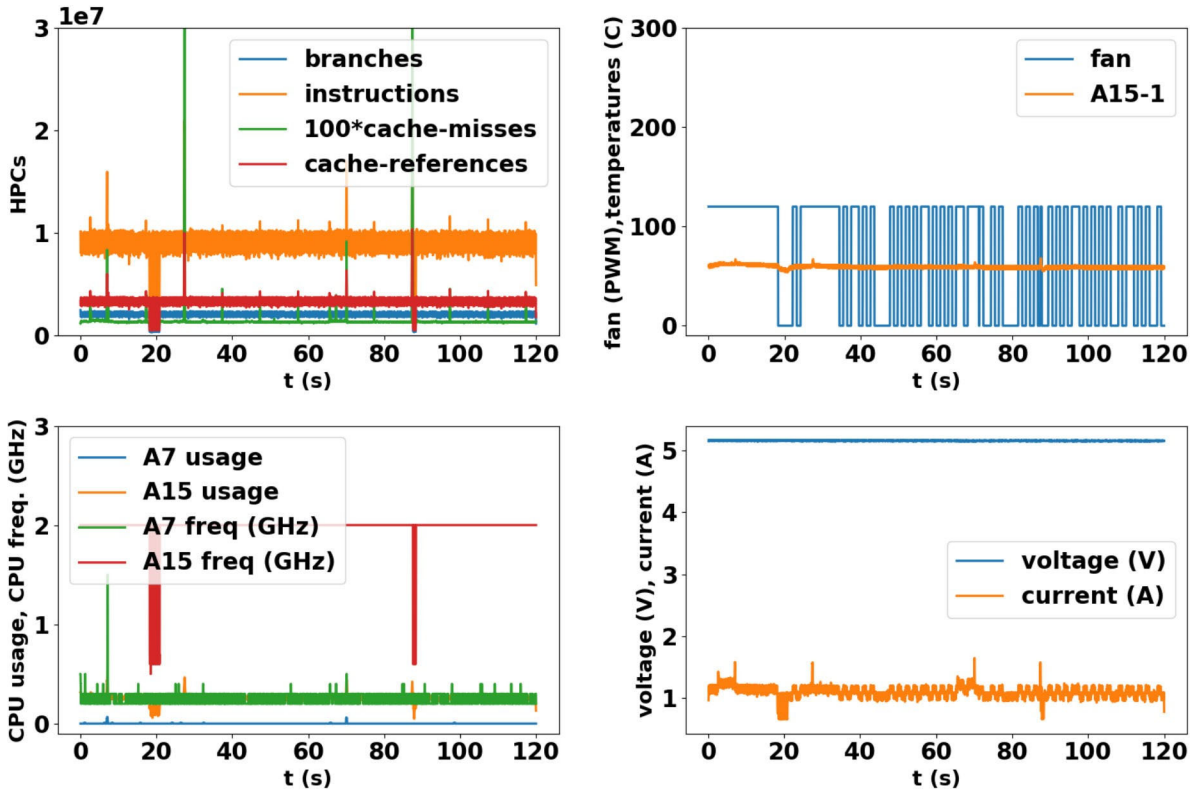


Figure 24: On-Device Data Collection (XU4, Test Process on an A15 Core, No Trojan).

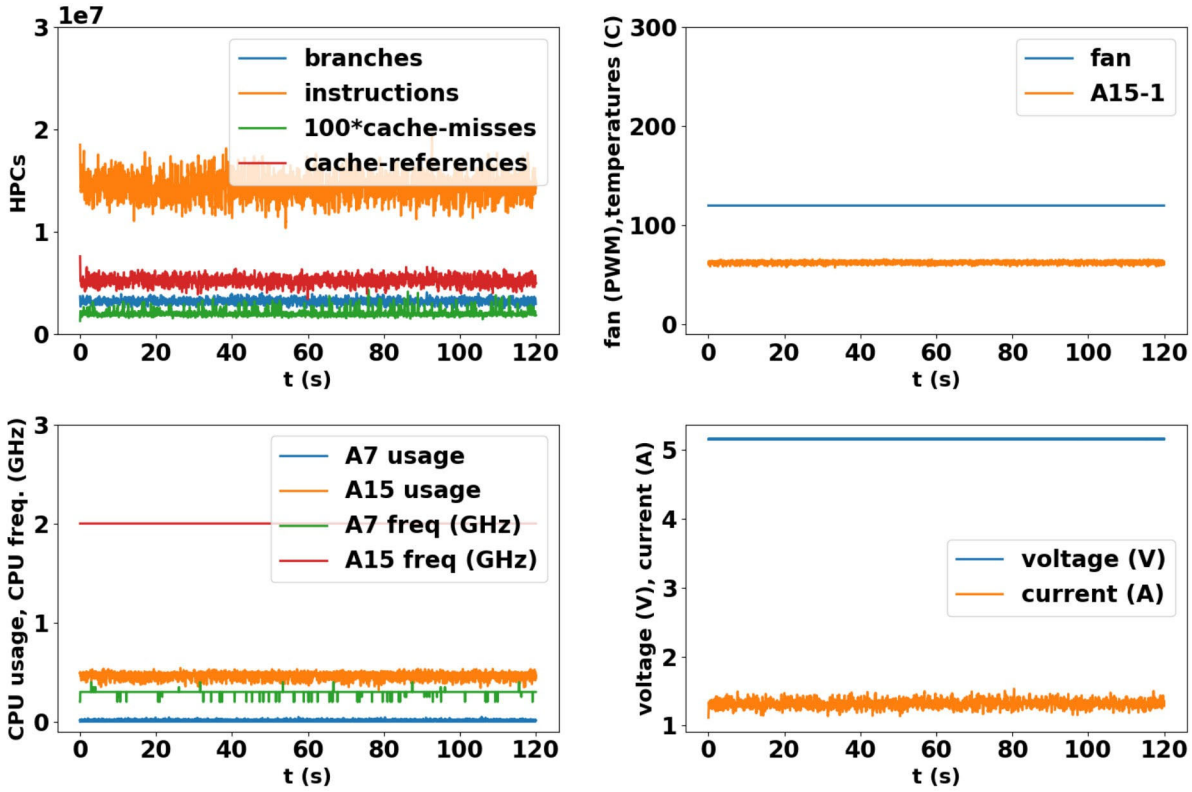


Figure 25: Data Collection from Behavioral Simulator (XU4, 2 Test Processes on 2 A15 Cores, No Trojan).

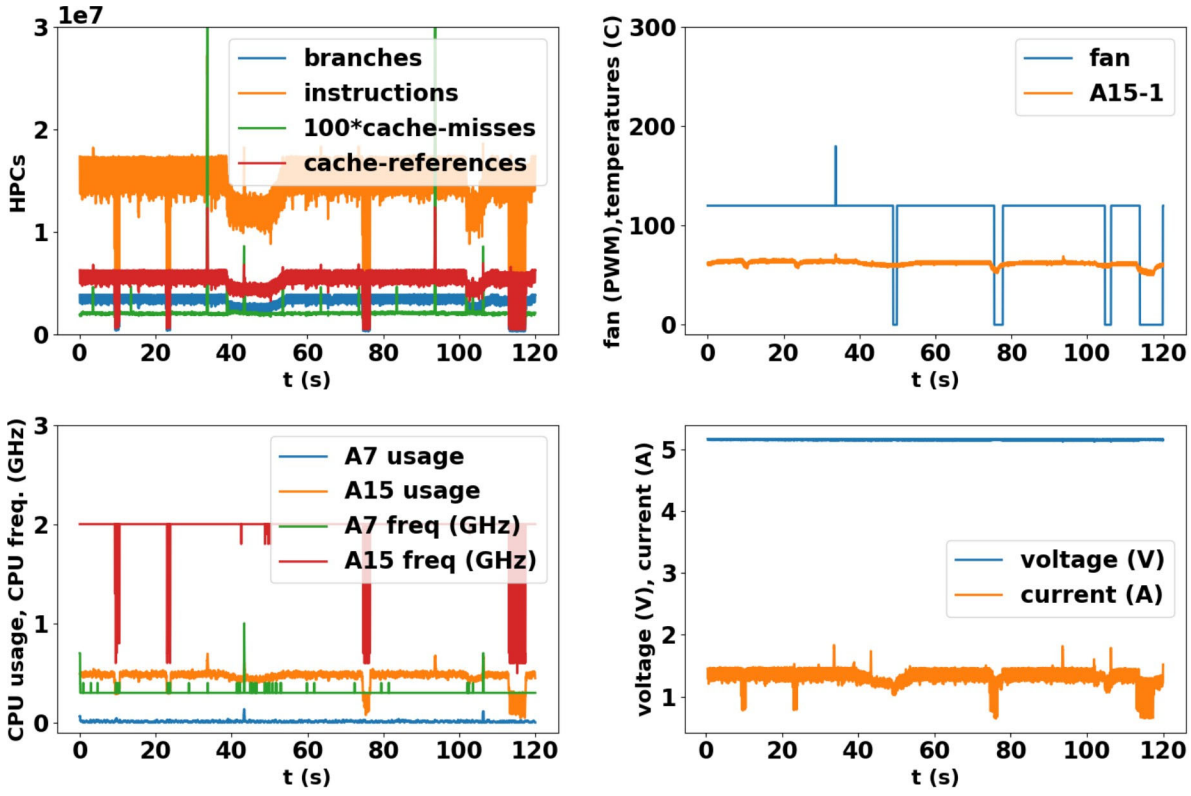


Figure 26: On-Device Data Collection (XU4, 2 Test Processes on 2 A15 Cores, No Trojan).

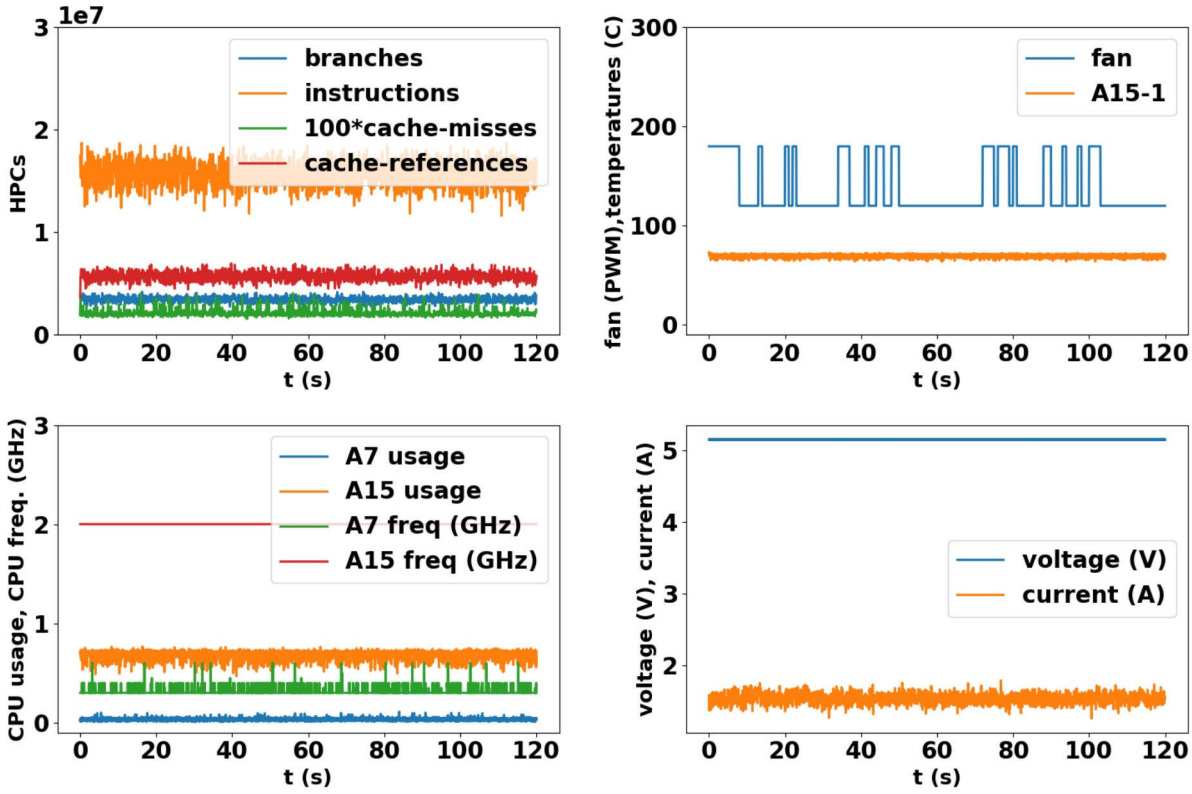


Figure 27: Data Collection from Behavioral Simulator (XU4, 3 Test Processes on 3 A15 cores, No Trojan).

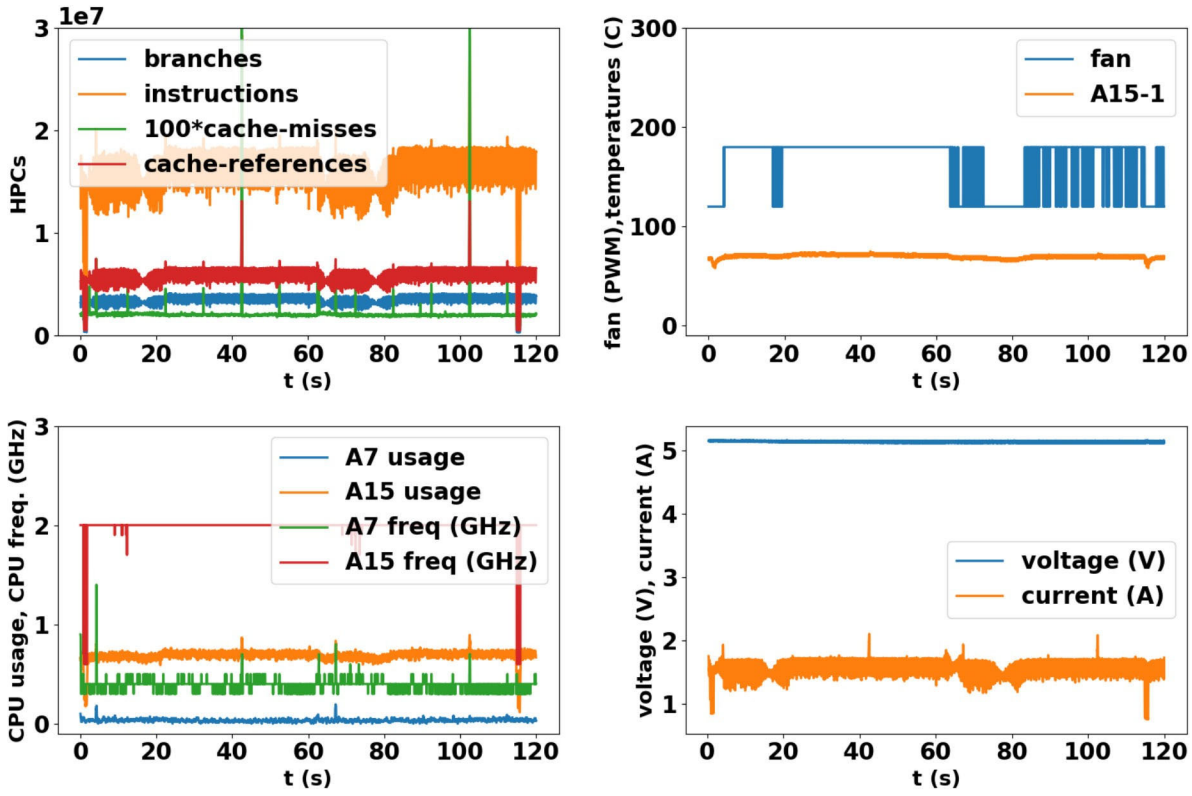


Figure 28: On-Device Data Collection (XU4, 3 Test Processes on 3 A15 cores, No Trojan).

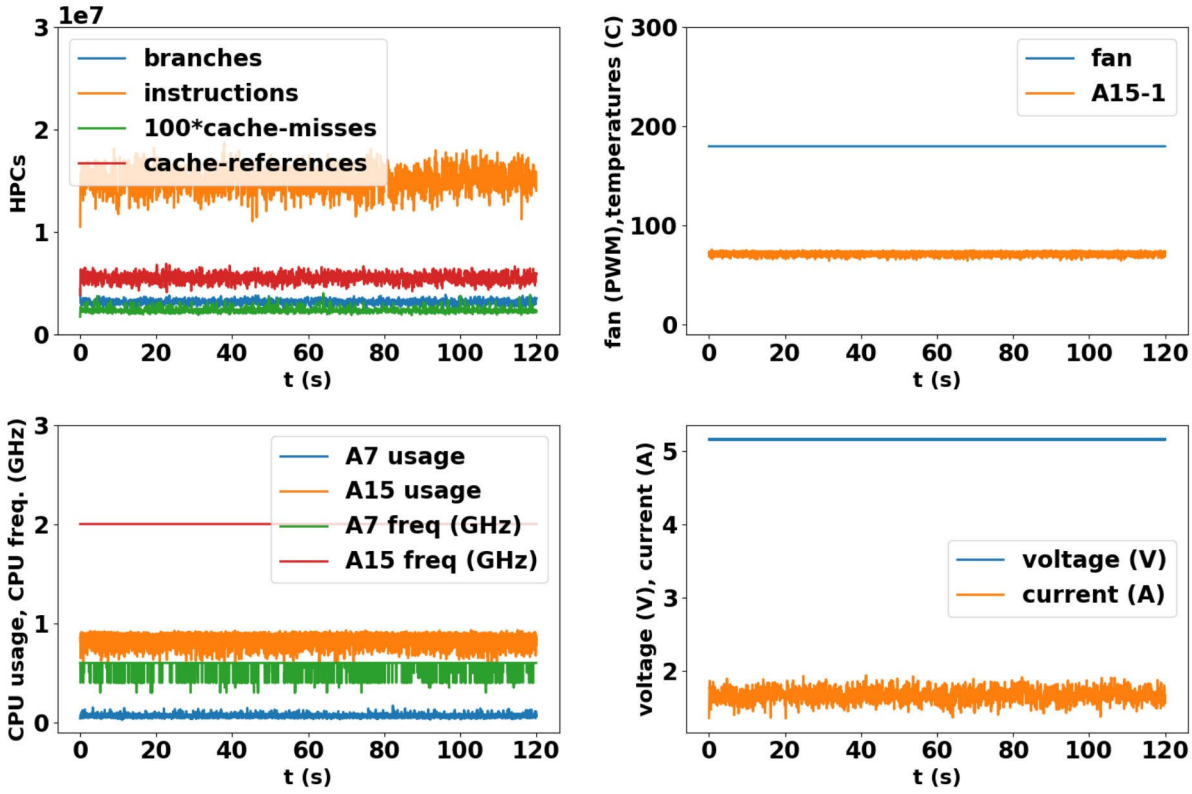


Figure 29: Data Collection from Behavioral Simulator (XU4, 4 Test Processes on 4 A15 Cores, No Trojan).

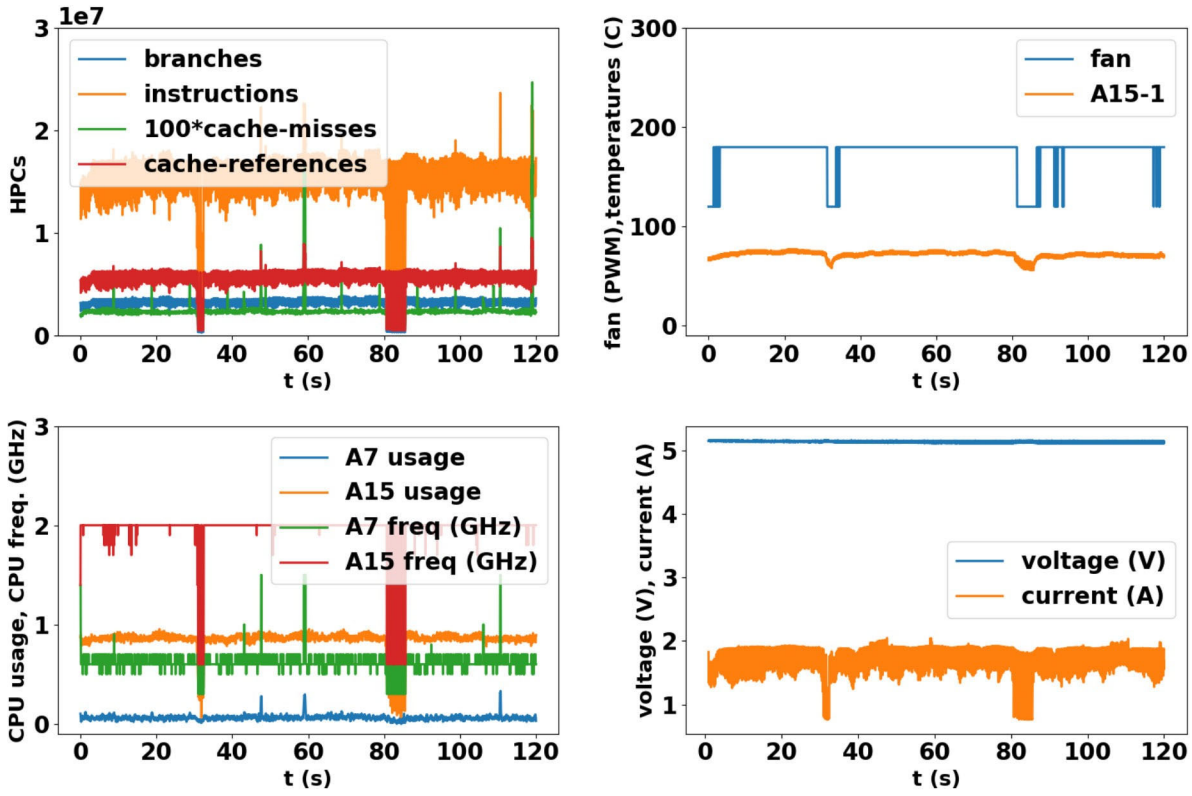


Figure 30: On-Device Data Collection (XU4, 4 Test Processes on 4 A15 Cores, No Trojan).

3.5 Controlled Excitation and Fuzzing

The objective of dynamic fuzzing is to accentuate the observable effects of anomalies/Trojans in the system. By dynamically varying software, parametric, and hardware-level components, controlled dynamic fuzzing seeks to induce greater visibility into the system operation via the monitored side channels and amplify the differential behavior of a Trojan in the system relative to the behavior of Trojan-free baseline design of the complex system. Some types of dynamic fuzzing approaches that can be applied in our proposed approach include:

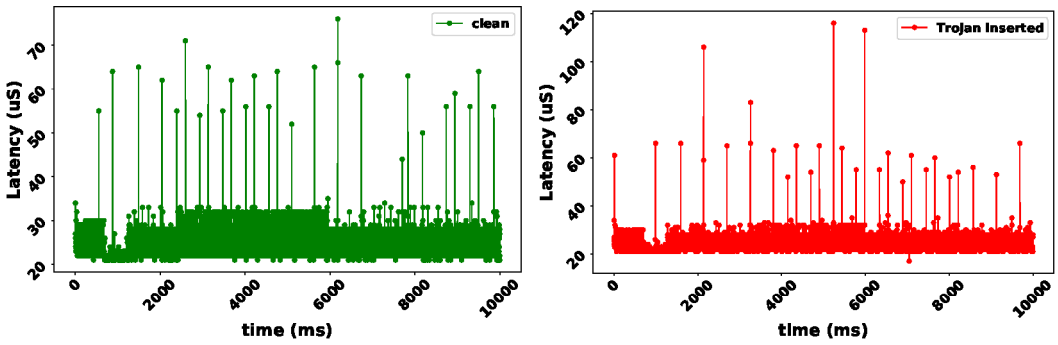
1. Software fuzzing using crafted test codes: We devised light-weight test codes which when injected, collect HPC and power side-channel measurements when the CPUs of SBCs are executing CPU-intensive or I/O-intensive operations. These instrumented test codes enable collection of on-device side channels under controlled CPU-intensive or I/O-intensive operations. We also devised test codes to test SBCs and their communication properties. The test codes injected in a SBC exhaustively sends to and receives varying sizes of data packets from other SBCs for multiple iterations, simultaneously monitoring every SBC's communication side-channels (connection setup latency, network latency, throughput). Our test codes were implemented with MPI communication protocol, thereby providing stability and repeatability to the measurements. Such test code instrumentation enables fuzzing the communication properties via network peripherals of SBCs to create unique patterns in communication side channels. Dynamic test code injection into the application-specific codes was performed to create discernible patterns in the monitored side channels.

2. Parameter fuzzing: We change the protocol parameters such as bit rate in Ethernet card settings and dynamically adjust application-specific configuration parameters to operate the system at its or beyond its boundary conditions. We also dynamically vary the subsets of SBCs that are communicating at any given time.

3. Hardware fuzzing: This includes changing device supply voltages, scaling CPU clock frequencies among modes (using a governor), and controlled over clocking.

These dynamic fuzzing modalities are coupled and overlap to some extent. For example, changing the test codes not only change the SBCs that communicate but can also be applied in combination in parallel and as a sequence. The exponential combinations of controlled dynamic fuzzing modalities offers a moving target defense and probing tool for the defender to increase the likelihood of detecting Trojans, reduce false positives, and evade attacker attempts to undermine them. The defender has the asymmetric advantage over the adversary since they do not know what types of fuzzing will be applied, when, and in what sequence, and with what parameters, and what side channels will be measured. Depending on the Trojan type and location, controlled excitations enable increasing of Trojan visibility either by localizing activity and measurements (e.g., using pair-wise SBCs communications) or by violating intrinsic assumptions made by a Trojan (e.g., by varying bit rates outside the normal system operation).

Hardware fuzzing includes fuzzing UART peripherals of a SBC to test their connection properties. Similar to the ethernet protocol, we developed instrumented test codes that send variable sizes of data packets between SBCs using their UART communication protocols. Figure 31 depicts clean and Trojan suspicious variations in the UART measurements between a pair of SBCs. Trojan



(a) (b)

Figure 31: UART latency measurements of a connection between SBCs. (a) latency variations when SBCs are directly connected via UART (Trojan-free) and (b) latency variations when SBCs are connected via a Trojan in the middle.

measurements in Figure 31(b) were obtained by establishing the UART communication between two SBCs via an intermediate Trojan in the middle. The latencies in the plots correspond to the delay in sending and receiving data packets. The significant difference in latencies between clean (upto 70 microseconds) and Trojan measurements (upto 120 microseconds) indicate the large time-delay for an SBC communicating with an SBC on the other end of the connection, due to a Trojan MITM. Fuzzing the UART channel can also include, for example, varying the bit rate to increase the visibility of an MITM Trojan (e.g., by forcing violation of an assumption made by the Trojan regarding the range of bit rates used in a normally operating system). By varying the various fuzzing "knobs" as discussed above (e.g., scaling CPU frequencies) while running test codes with known CPU-intensive and I/O-intensive patterns and collecting multi-modal side channel signals, the Trojan visibility is increased to facilitate Trojan detection.

4.0 Results and Discussion

4.1 Experimental Studies of Equivalence/ Invariance Based Trojan Detection

Based on the equivalence/invariance based golden-free Trojan detection methodology described in Section 3.3, we describe experimental studies on the multi-SBC testbed outlined in Section 3.1. While the Trojan detection is general, we outline four Trojans which can be maliciously implanted in multi-SBC systems as shown in Figure 32:

1. *Machine-in-the-Middle (MITM) Trojan:* An attacker inserts/modifies a component to modify I/O operation between PCBs. We emulate this by moving MC1-4 to another cluster, with a router Trojan sniffing inter-cluster communications.

2. *Counterfeit Trojan:* An attacker replaces a good device with a counterfeit. We emulated this Trojan by replacing XU4 with XU3, and Rpi4 with Rpi3.

3. *Firmware Trojan:* An attacker may modify the firmware in a device to snoop on data communications. We emulated this with two Trojans. First, we modified an open-source MPI communication [48] firmware in MC1-1 to log outbound packets. Second, we implemented a kernel module that causes excessive data cache flushes in MC1-2.

4. *Software rootkit:* An attacker introduces a library rootkit to run on MC1-2 in the background to accelerate device aging when a user runs an application.

We selected the test codes to provide different resource consumption, such as IO- or CPU-intensive, and communications between devices. The test codes were designed such that in consecutive runs, a single run of the application is followed by an interval of sleep. During the application runs, we ensured that no other applications were running besides the test application, and collected the side channel signal measurements.

4.1.1 Trojan Detection Using Communication Side Channels

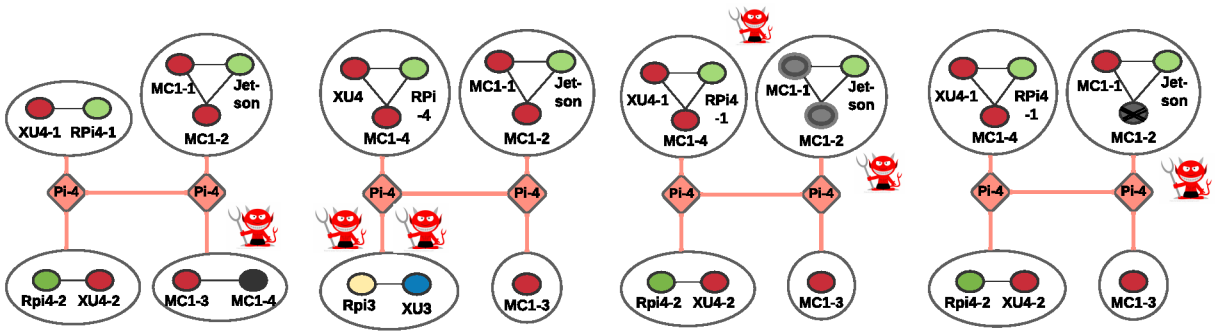
We measured three communication side channels: connection setup time, throughput, and latency between the PCBs in the COTS system. Snapshots of time-series of side channel measurements in

Figure 33 are used to validate the hypothesized properties. The nodes in the graph in Figure 33 are labeled with the device node names and clusters of our topology in Figure 16. There is a marked variation in the mean, deviations of the side channels across different connections paths, and across multiple measurements from a connection.

Different edges exhibit similarities/dissimilarities depending on the PCB type and the interconnection topology (e.g., higher latencies for connections between PCBs in the different VLANs). From snapshots of on-SBC and inter-PCB side channel measurements, predicate validation discussed above entails probabilistic verification of properties relative to the expected behaviors or across different nodes/edges. For example, given an expected interconnection topology of the multi-PCB system, detecting an MITM Trojan involves verification of equivalences or dissimilarity between measurements from the different edges. In the context of the testbed, addition of an MITM Trojan entails introducing a new VLAN with the corresponding router being the adversary-injected MITM Trojan or movement of a PCB from the intended VLAN to a different VLAN.

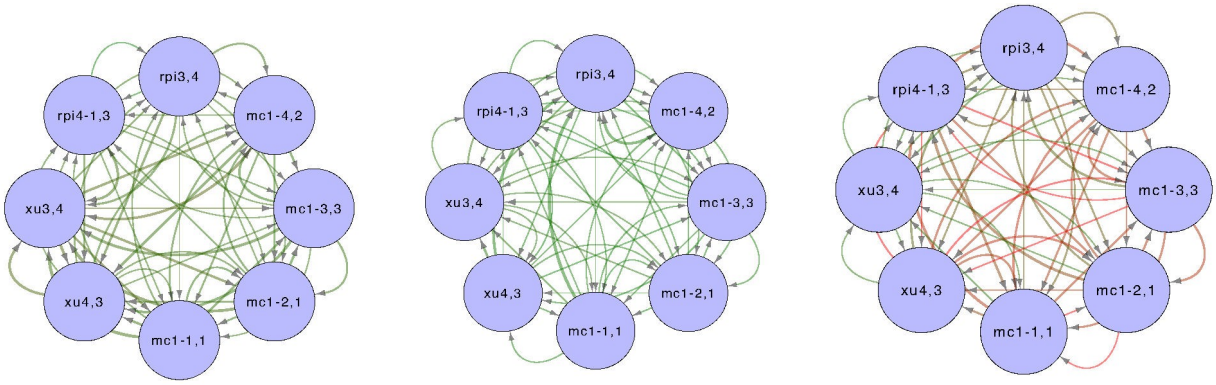
Viewing measurements of features of a side channel signal (e.g., network latencies) as observations of random variables, the likelihood of satisfaction of edge equivalence predicates can be computed by using the empirical cumulative distribution functions (ECDFs) and calculating divergences between the ECDFs. Examples of ECDFs of communication side channels in a snapshot are shown in Figure 33. Divergences between calculated ECDFs for different edges can be quantified using, for example, Bhattacharyya distance metrics as shown in Figures 34, 35, and 36(c) & (d).

For our experiments using communication side channels, we consider two Trojans – an *MITM* Trojan and a *counterfeit* Trojan. Considering *R* and *G* colored devices, the likelihoods of satisfaction of the path invariance property for R↔G and R↔R connections are computed using ECDFs and the distances between the probability distributions of connections using the Bhattacharyya distance ρ in Figures 34, 35, and 36 respectively. We use a threshold of $\eta = 0.5$ to measure similarity using ρ. All pairwise connections in an *RR* bucket should satisfy the path invariance property and validated in Figure 34(a) for R↔R connections with almost identical ECDFs and $\rho > 0.5$ (Figure 34(c)). Higher ρ with identical ECDFs indicates the two connections are path invariant.



(a) MITM Trojan (b) Counterfeit Trojan (c) Firmware Trojan (d) Software rootkit

Figure 32: Trojans can be embedded at multiple levels (hardware, firmware, software, interconnects) in a multi-PCB COTS system. We will focus on the following types of Trojans: (a) Machine in the middle (MITM) that snoops on communications, (b) Counterfeits in lieu of requested ones, (c) low-level firmware Trojan, (d) rootkit in a library.

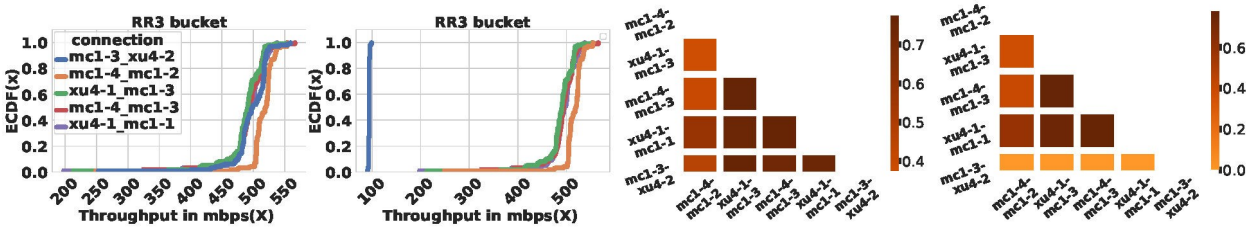


(a) Setup time (b) Latency (c) Throughput

Figure 33: Measurement snapshots collected over a time window from the communication side channels Setup time (left), latency (middle), and throughput (right). Mean of the features are shown as colors of the edges (smaller→larger values as colors from green→red) and standard deviations are shown as edge widths.

If a *counterfeit* (XU3) replaces XU4-2, this is detected as an anomaly by the throughput side channel ECDF of MC1-3↔XU4-2 connection (R↔R) by violating path invariance (Figure 34(b)). ρ(MC1-3→ XU4-2) with other connections < 0.5, failing path invariance in Figure 34(d).

Figure 35(a), 35(c), 36(a), and 36(c) shows that pairwise connections in a RG bucket satisfy the path invariance in the throughput and latency side channels. We move MC1-4 (R) to a different cluster and implant an *MITM* Trojan (router) to snoop on the communications of MC1-4 with PCBs in other clusters. This Trojan is detected in Figure 35(b) and 36(b) since the throughput and latency of ECDF of MC1-4↔RPi4-1 violate path invariance of R↔G connections. Also, the distance values MC1-4↔RPi4-1 in Figure 35(d) and 36(d) are < 0.5 validating this anomaly.

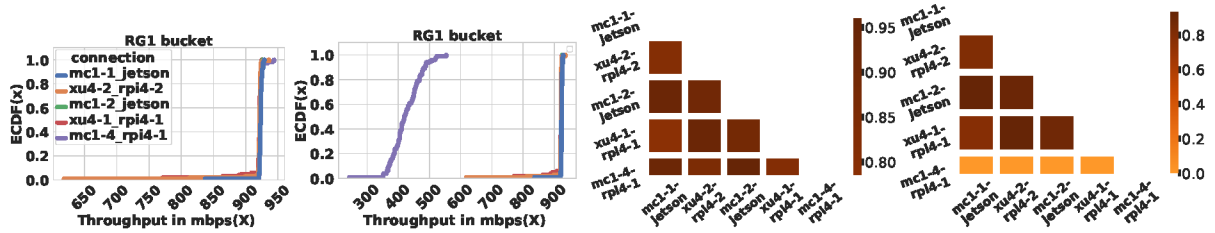


(a) Path invariant (b) Path variant (c) Path invariant (d) Path variant

Figure 34: ECDFs and Bhattacharyya distance calculated for RR3 bucket. (a) and (c) Path invariance among PCB-PCB connections, (b) and (d) with a path variance of a connection indicating an anomaly.

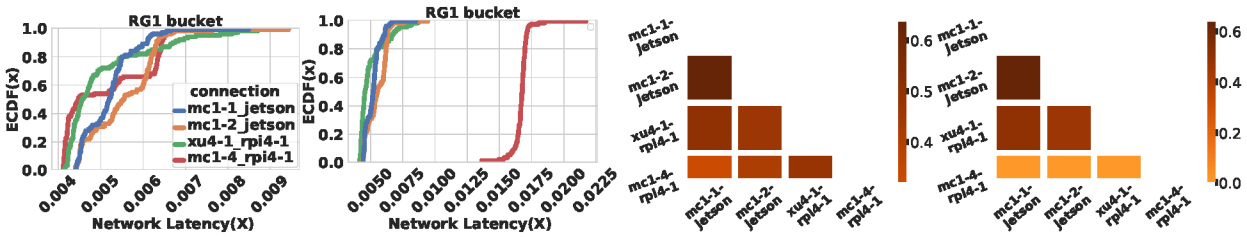
4.1.2 Trojan Detection using Power side channels

Power side channel is a coarse level indicator of CPU/GPU load of device. For Trojan detection, we executed a CPU-intensive workload on every board for 80 seconds and collected board-level



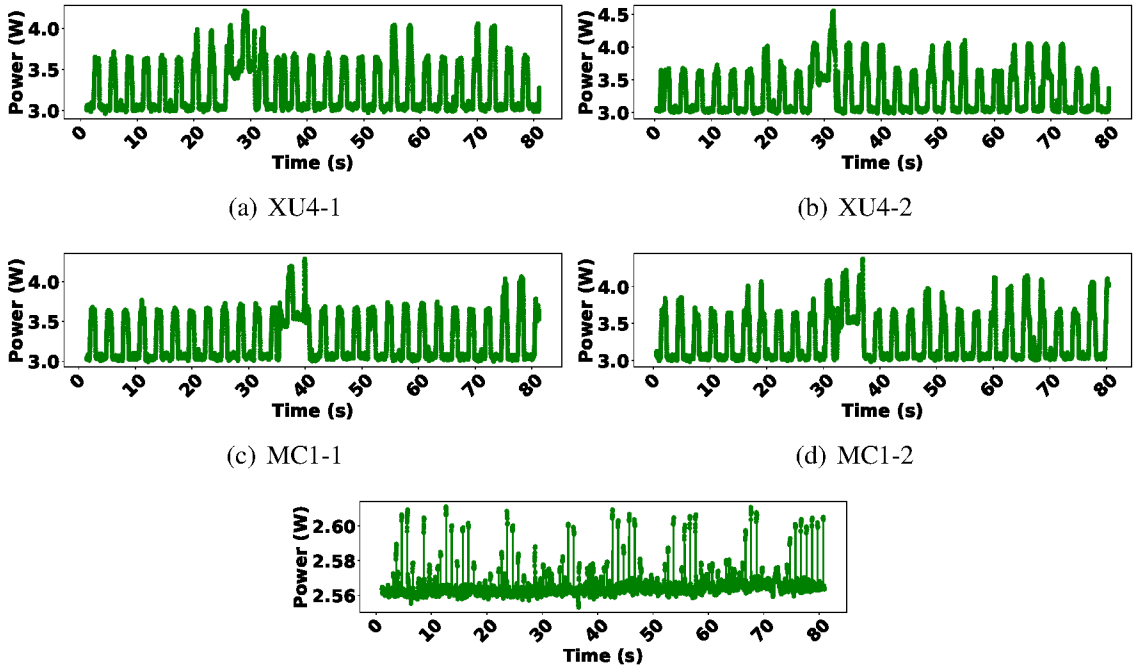
(a) Path invariant (b) Path variant (c) Path invariance (d) Path variance

Figure 35: ECDFs and Bhattacharyya distance on *network throughput* for RG1 bucket. (a) and (c) path invariance and, (b) and (d) path variance indicating an anomaly.



(a) Path invariant (b) Path variant (c) Path invariance (d) Path variance

Figure 36: ECDFs and Bhattacharyya distance on *network latency* for RG1 bucket. (a) and (c) path invariance, (b) and (d) path variance indicating an anomaly.



(a) XU4-1 (b) XU4-2

(c) MC1-1 (d) MC1-2

(e) XU4-2→XU3

Figure 37: Power traces (in Watts) for the PCBs in the red bucket. (e) violates the node variance property indicating an anomaly.

power consumption measurements from different PCBs. Figure 37 shows variations in power consumption of red colored PCBs while executing CPU-intensive application.

We evaluate our hypothesis of *node invariance* using the power side channel measurements for PCBs clustered with same color. To maintain the same run time execution environment, we set all the boards in user space governor using DVFS (Dynamic Voltage Frequency Scaling), andrun the workload at two operating frequencies: 1 GHz and 1.5 GHz. Figures 37(a)-37(e) showa typical power consumption profile for the workload we executed on the PCBs. For power side channel profile, we use Dynamic Time Warping (*DTW*) to measure the distance between the power traces of devices in same colored bucket. We consider the case of counterfeit Trojan, where *XU4-2* is replaced with *XU3*. We collect power traces first in the clean scenario, and after introducing the counterfeit trojan. From the plots 37(a)-37(d), we visualize that the power traces obtained are similar, while Figure 37(e) shows anomalous behavior. In Table 4, we report pairwise DTW distances for the devices in the red bucket. To account for process variations and noise, we set threshold $\delta_{DTW} = 25$. This violates the node invariance ($DTW_{XU4\text{-}2 \to D} > 25, \forall D \in V^d_{red}$) for the red bucket, thereby detecting the counterfeit trojan (XU3). Determination of δ_{DTW} does not require any prior information of the golden model, and is based on prior knowledge of process variations during PCB manufacturing.

Setting	Device under Test	PCB Devices							
		XU4-1		XU4-2		MC1-1		MC1-2	
		1G	1.5G	1G	1.5G	1G	1.5G	1G	1.5G
Node-invariant	**XU4-1**	-	-	12.64	16.33	12.99	22.43	16.38	22.32
	XU4-2	12.64	16.33	-	-	15.44	18.77	14.42	20.91
	MC1-1	12.99	22.43	15.44	18.77	-	-	11.62	19.50
	MC1-2	16.38	22.32	14.42	20.91	11.62	19.50	-	-
Node-variant	**XU4-1**	-	-	**95.21**	**151.95**	12.99	22.43	16.38	22.32
	XU4-2	**95.21**	**151.95**	-	-	**95.27**	**152.59**	**99.93**	**151.02**
	MC1-1	12.99	22.43	**95.27**	**152.59**	-	-	11.62	19.50
	MC1-2	16.38	22.32	**99.93**	**151.02**	11.62	19.50	-	-

Table 4: Deviation in DTW distances in power side channels when XU4-2 is subverted with XU3. $DTW(XU4-2, D) > \delta_{DTW} (= 25)\ \forall D \in V^d_{red}$ fails node invariance.

4.1.3 Trojan Detection Using HPC Side Channels

In our experiments, we use software fuzzing by running CPU- and IO- intensive test codes on same color nodes (in the *R* colored bucket) at two different frequencies (1GHz and 1.5GHz) and collect HPC traces. The *R* colored bucket has Odroid PCBs MC1 and XU4s.To explain the detection capability of our methodology, we collected clean HPC traces from all PCBs and injected two Trojans: 1) a cache flush trojan in MC1-2, and 2) an MPI software library rootkit in MC1-1. We collected HPC traces from these boards. In order to avoid the noise in the HPC traces, we collected a large quantity of samples (~75,000) and computed the mean and standard deviation for the sampled data (see Table 5). We set the threshold $\delta = 20\%$ for checking node invariance property considering process variation across the MC1 PCB chipsets. When the Trojans are active,

the mean statistics of MC1-2 substantially increases as seen in Table 5. As shown in Table 6, σ_{pd} is within δ for PCBs that satisfy the node invariance property, except for the MC1-2 and MC1-1 PCBs. A defender can adjust the threshold δ depending on the test code and check if σ_{pd} is within δ. The defender can use alternate CPU-intensive and IO-intensive codes presenting a moving target for attacker.

Table 5: HPC side channels in the presence of two firmware Trojans: cache flush and an MPI library Trojans. We consider Data cache access (DCA), instruction cache access (ICA) and data cache miss (DCM) side channels.

Type	HPC	Freq. (GHz)	PCB					
			MC1-1		MC1-2		MC1-3	
			Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
Cache-flush	Total Cycles	1	6545.36	1536.25	**8788.87**	**2921.43**	7254.36	1836.25
		1.5	11237.78	3536.57	**17984.26**	**5852.15**	13498.56	4136.25
	L2 DCA	1	29.28	17.83	**181.80**	**121.38**	28.35	18.25
		1.5	32.80	23.00	**393.98**	**264.7**	31.22	23.66
	L1 DCM	1	82.24	33.66	**459.23**	**362.68**	98.87	57.09
		1.5	61.24	36.25	**3174.18**	**2023.85**	60.31	38.75
MPI	L2 DCA	1	**287.36**	**37.89**	211.24	32.25	214.25	29.68
		1.5	**331.72**	**47.89**	268.73	37.53	257.57	33.32
	L1 ICA	1	**795.25**	**99.65**	645.65	52.89	694.36	48.25
		1.5	**955.29**	**115.32**	821.81	77.16	827.15	81.39
	L2 TCM	1	**65.36**	**14.14**	85.25	15.55	81.27	18.14
		1.5	**70.72**	**16.17**	99.03	17.69	91.39	18.89

Table 6: % deviation of avg. of HPC side channels from its expected invariance under two firmware Trojans. (i. flushes cache in MC1-2, ii. modifies MPI comm. library in MC1-1)

Type	Setting	PCB under test	PCB					
			MC1-1		MC1-2		MC1-3	
			1 GHz	1.5 GHz	1 GHz	1.5 GHz	1 GHz	1.5 GHz
Cache-flush	Node-invariant	MC1-1	-	-	15.14	12.19	9.77	16.74
		MC1-2	17.16	14.17	-	-	14.29	7.81
		MC1-3	10.83	19.79	11.12	12.95	-	-
	Node-variant	MC1-1	-	-	**25.90**	**40.72**	9.77	16.74
		MC1-2	**34.83**	**66.19**	-	-	**21.82**	**40.54**
		MC1-3	10.83	19.79	17.99	**29.95**	-	-
MPI	Node-invariant	MC1-1	-	-	3.14	2.52	5.45	4.32
		MC1-2	8.75	8.94	-	-	6.69	1.54
		MC1-3	9.19	4.34	7.15	1.55	-	-
	Node-variant	MC1-1	-	-	**24.68**	**20.55**	**17.17**	**21.22**
		MC1-2	**19.93**	**17.14**	-	-	6.69	1.54
		MC1-3	**14.81**	**17.58**	7.15	1.55	-	-

4.1.4 Trojan Detection Using Thermal Side Channels

We excite the thermal side channels (i.e., the temperature variations in the PCBs in Figure 38) by running CPU-intensive test codes on the PCBs operating at 1 GHz and 1.5 GHz. The circled regions indicate the time intervals of low temperature. This is due to the wait times between consecutive runs of the test code. Counterfeit Trojans, e.g., older versions of an intended PCB, usually result in slower processing and thereby more time to run a workload in a given time window, therefore showing smaller time interval of low temperature zone. In particular, in Figure 38, it is seen that the thermal side channel signals are very different between the XU3 and XU4 PCBs. Hence, if the original design called for these two PCBs to be identical (e.g., both being XU4's) and if the adversary has replaced one of the PCBs with an inferior counterfeit PCB (i.e., the XU3), the invalidation of the expected similarity of thermal side channel signals between the two PCBs indicates an anomaly.

The software-level fuzzing of running the CPU-intensive test code helps in more accurately exposing the Trojan. Similarly, firmware or software Trojans can also interfere with the normal operation of PCBs, causing deviations from expected invariance properties that can be detected using the thermal side channel. We used % of deviation from mean as the metric to identify dissimilarities between thermal side channels of PCBs in Table 8. The node invariance property identifies MC1-1, MC1-2, and MC1-3 PCBs to have similar properties from their specifications. Thermal measurements from them are then expected to be within a threshold of % deviation of mean. However, results in Table 8 indicate an increase in % deviation exposing the dissimilar- ity between PCB side channels, when a software rootkit runs in MC1-2. The results in Table 8 show that if MC1-2 runs a software Trojan running in the background, the % deviation of temper-ature of MC1-2 relative to MC1-1 and MC1-3 increases significantly from -8.5 to 47.4 and from -4.9 to 53.3, respectively. Table 7 reinforces this analysis by showing that the standard deviation of thermal data distribution of MC1-2 varies significantly from the mean compared to the other two. Table 7 also confirms the importance of hardware fuzzing. Mean and standard deviations of thermal measurements of MC1-2 are higher when the PCBs operate at 1 GHz compared to 1.5 GHz. This is because if the Trojan is resource-intensive, the CPU temperature spikes spuriously compared to average temperature rise at the lower, 1 GHz clock frequency while the temperature variations at the higher, 1.5 GHz clock frequency are at a higher level, hiding Trojan behavior.

Table 7: Statistics of thermal side channels when software rootkit runs in MC1-2 (in bold).

Frequency	PCBs					
	MC1-1		MC1-2		MC1-3	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
1 GHz	33913.3	307.6	**50020.4**	**2231.2**	33627.1	2827.2
1.5 GHz	38362.3	694.8	39329.9	819.3	37516.8	675.8

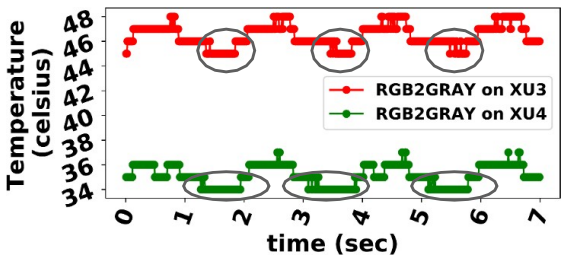


Figure 38: Thermal side channels of XU3 and XU4 when running a CPU-intensive (RGB2GRAY) test code at 1 GHz. Average time interval of low CPU temp for XU3 is less than for XU4, an anomaly when the design expects the PCBs identical.

Table 8: % deviation in thermal side channel when MC1-2 has a rootkit. σ_{pd}(MC1-2) > δ_H = 10% fails invariance.

Setting	Device under Test	PCB Devices					
		MC1-1		MC1-2		MC1-3	
		1G	1.5G	1G	1.5G	1G	1.5G
Node-invariant	MC1-1	-	-	9.3	6.4	3.9	2.2
	MC1-2	-8.5	-6	-	-	-4.9	-3.9
	MC1-3	-3.7	-2.2	5.2	4	-	-
Node-variant	MC1-1	-	-	**-32.2**	-2.4	3.94	2.2
	MC1-2	**47.4**	2.5	-	-	**53.3**	4.8
	MC1-3	-3.7	-2.2	**-34.7**	-4.6	-	-

4.2 Simulator-Trained ML-Based Trojan Detection

The Trojan detection methodology described in Section 3.4 was applied to study detection of several types of Trojans as discussed in this section.

4.2.1 Multi-Modal Side Channel Measurement Framework

For experimental studies in this section, we utilize the Odroid XU4 SBC as the representative complex PCB, which is part of the multi-PCB testbed discussed in Section 3.1. This SBC has a 32-bit 8-core (double quad core) ARM Cortex processor, comprising of a quad-core Cortex A15 and a quad-core Cortex A7, with max frequencies of 2GHz and 1.5GHz, respectively. The SBC has 2GB RAM, a fan for processor cooling, and Ethernet, GPIO, and UART interfaces. All the side channels listed in Section 3.4.2 can be measured on the Odroid XU4 SBC. The HPCs are measured using the Linux `perf` tool (using the system-wide mode). The CPU load/usage measurements are read using the `/proc/stat` file in Linux (this file provides measurements of "jiffies" or time slices sincesystem start of nice, user, system, idle, etc. — filtering and processing a time series of these valuesyields CPU usage estimates). The CPU frequencies are read from `scaling_cur_freq` files underthe `/sys/devices` filesystem. The frequencies of the A7 and A15 processor cores are independentand are read separately. Also, the CPU usage of the A7 and A15 cores are read separately. The temperatures of the four A15 cores are read separately from corresponding `thermal_zoneN/temp` files under `/sys/devices`. The fan speed is measured (in terms of PWM levels) from a `/sys/devices` file. The power draw of the PCB is measured using an external Smart Power monitor from whichpower measurements are streamed wirelessly to the PCB. To collect the various side channels within an integrated collection framework with consistent time stamps, a multithreaded multi- modal measurer was implemented. The sampling rates of the different side channels were setto 100 Hz for HPCs, CPU usage measurements, CPU frequency measurements, and temperature measurements and to 20 Hz for fan speed measurements. The sampling rate for power measurements is configured in the external Smart Power module as 200 Hz. Sample collected data from the Odroid XU4 under different workloads are shown in Figures 22, 24, 26, 28, and 30.

4.2.2 Representative Trojans for Experimental Studies

To determine efficacy of the approach, we needed to develop a suitable corpus of Trojans. Two challenges in the development of a representative set of Trojans were: (1) there is no ready-made dataset of complex PCB-level Trojans, (2) modifying PCB layouts is challenging. While some modifications are possible (e.g. component substitution), others are difficult. Hence, we emulate PCB-level Trojans by modifying on-board firmware. We devised four categories of Trojan and the most relevant side channels to detect them as listed in Table 9.

`LEAK` Trojans (Figure 39(a)) aim to transmit proprietary (secret) information via side channels. An example of this is where a system is air-gapped. Though traditional data transmission mechanisms are blocked by this, a sensitive radio receiver may still be used to pick up certain information from the device - e.g., the frequency the CPU is operating at. An acoustic microphone may also be used to determine fan speed.

`PD` Trojans (Figure 39(b)) aim to degrade the performance of the target system by using improper configurations for the CPU frequency governors. These normally have settings focusing to scale up

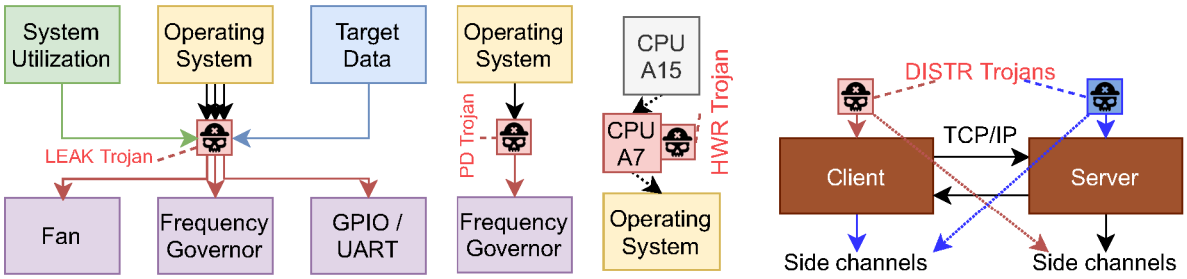
ID	Type	Origin	Most Relevant Side Channels For Detection						
			CPU Usage	CPU Freq	CPU Temp	Power cons.	CPU HPCs	Fan Spd.	Comms. metadata
LEAK	Information leakage	SW	Yes	Yes	Yes	Yes		Yes	
PD	Performance degradation	SW	Yes	Yes			Yes		
HWR	Hardware replacement	HW	Yes		Yes	Yes	Yes		
DISTR	Distributed Trojan	SW	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Table 9: Categories of developed Trojans.

the frequency in times of high demand and scale back when appropriate. A Trojan could configure CPU to run at low speed sporadically when busy, or at all times.

`HWR` Trojans in Figure 39(c) emulate replacement of authentic components with counterfeits or low-quality components. We consider the replacement of an A15 core by the less powerful A7 core. We emulated such a substitution by pinning the defender test code which is intended to be pinned to an A15 core to an A7 core instead when collecting the side channel data.

`DISTR` Trojans in Figure 39(d) are complex, and cover Trojans that are not necessarily detectable locally, but are detectable by the effects on a distinct board. We consider scenarios with two boards: a client and server.



(a) Information Leakage (b) Perf. Degrade (c) HW Replacement (d) Distributed effects

Figure 39: Types of Trojans.

We designed several Trojan exemplars that target a PCB (and may be detected by fusing multiple side channels) and summarize them in Table 10. The *stealthy* trojans trigger only during periods of high activity to mask their presence in the side channel data. We designed ten Trojans (7 leakage, 2 performance degradation, and 1 hardware replacement). Trojans that affect a distributed multi-PCB system are summarized in Table 10. Two PCBs run an *image processing* system, whereby the client PCB has a repository of images it compresses and uploads to the server PCB for processing. The server decompresses images and responds with the processed image. We designed three Trojans (2 on the client and 1 on the server).

Figures 40, 41, 42 show the data of side channel measurements collected on-device when LEAK.FAN, LEAK.CPU, and LEAK.GPIO Trojans are inserted. Figures 43, 44, 45, 46, 47 show the data collected when Trojans LEAK.GPIO, PD.DYN, LEAK.SCPU, LEAK.SFAN, HWR.A15 are inserted while four test processes run on four cores.

4.2.3 Trojan Detection Results

The side channel measurement framework discussed in Sections 3.4.2 and 4.2.1 was used to collect time series side channel data (for a time duration of 120 s in each test case) from the Odroid XU4 SBC under the various Trojans described in Section 4.2.2 and under normal (no Trojan) conditions. As discussed in Section 4.2.1, a multi-threaded data collector is used to ensure consistent time stamps across all the side channels. The sampling rates however are different between different side channels (Section 4.2.1) and the windowing/interpolation approaches discussed in Section 3.4.2 to extract sliding time windows of temporally aligned side channels from which features can be extracted using statistical/signal processing and autoencoder based methods (Section 3.4.2). Since the power side channel measurements have the highest sampling rate (200 Hz), the interpolation-based windowing uses 200 Hz as the common sampling rate for interpolation of the other side channels. The one-class classifier (Section 3.4.2) is trained using the same windowing and feature extraction techniques applied to simulator-generated data as outlined in Section 3.4.3, which stands in as a proxy for an unavailable golden known-good board for generation of training data.

To evaluate the system-under-test for Trojans, the extracted features over sliding time windows are passed through the one-class classifier and the behavioral analyzer (Section 3.4.4). The time

Table 10: List of developed Trojans.

Single board Trojans	
ID	Description
LEAK.FAN	Leak secret information using Fan speed
LEAK.SFAN	Stealthy version of LEAK.FAN1
LEAK.CPU	Leak secret information using CPU Freq.
LEAK.SCPU	Stealthy version of LEAK.CPU1
LEAK.HYB	Combination of LEAK.FAN1 and LEAK.CPU1
LEAK.SHYB	Combination of LEAK.FAN2 and LEAK.CPU2
LEAK.GPIO	Leak secret information using UART and GPIO
PD.DYN	Dynamic lowering of CPU freq.
PD.STA	Statically sets a low CPU freq.
HWR.A15	Replaces A15 core with A7 core.
Multi-board Trojans	
ID	Description
DISTR.BAD	Client board uploads corrupt files
DISTR.BIG	Client board uploads large files
DISTR.SLW	Server board introduces delays

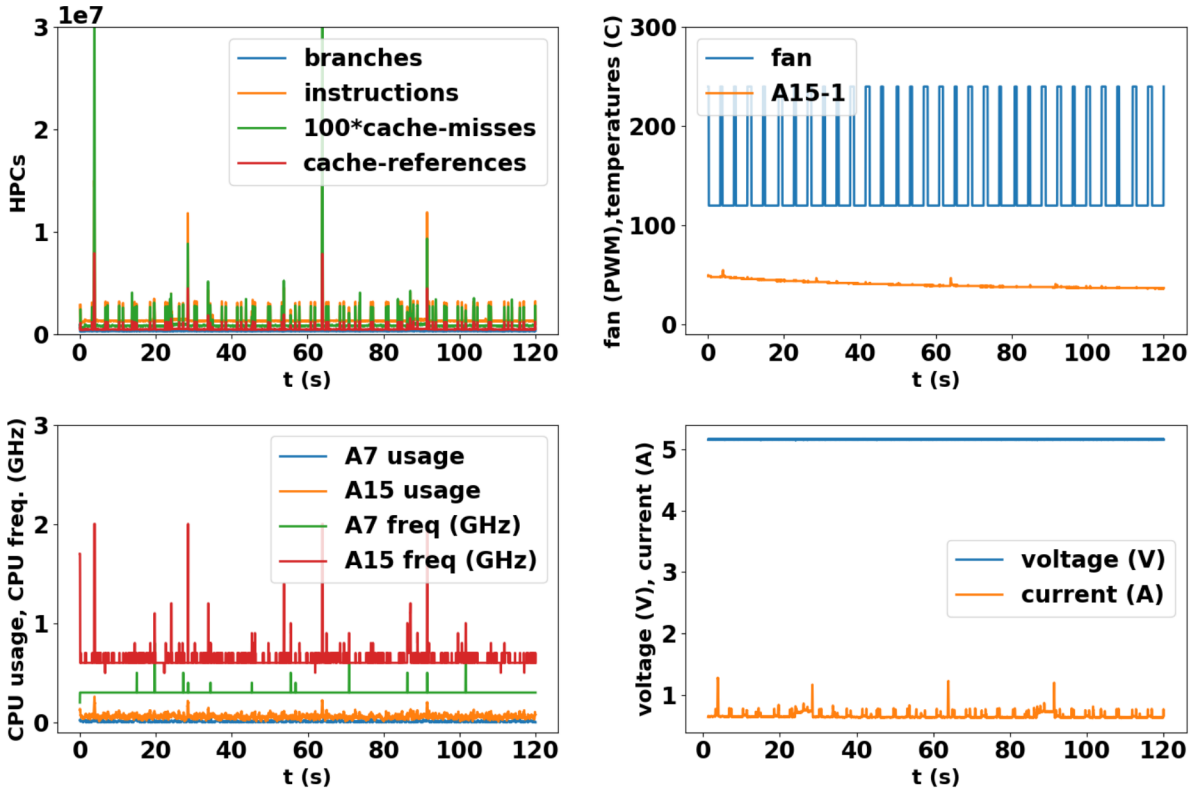


Figure 40: Data collection on XU4 (Quiescent with Fan Info Leak Trojan).

series outputs (i.e., anomaly detection over sliding time windows) are combined through a temporal ensemble in Section 3.4.2 to generate the normal vs. anomalous classifications. The sliding time window parameters, length T and time shift Δ, are set to 0.25 s and 0.02 s, respectively. The training (Section 3.4.2) of the ML anomaly detector is based on simulator-generated data (see Section 3.4.3). For random augmentations in ML anomaly detector training (Section 3.4.2), the standard deviation of the zero-mean additive noise is set to 0.02 times the magnitudes of the feature values. For adaptive thresholding of the ML anomaly detector, the threshold percentile is set to 0.01 (i.e., the threshold is picked such that 1% of the validation data is classified anomalous; 25% of simulation-generated data for training is used as validation data). For ensemble and temporal filtering, we set the median filter width W to 25 and use equal weights for ML anomaly detector outputs and behavioral-analysis anomaly detector.

The performance (accuracy, precision, recall, and F1-score) of our Trojan detector for several Trojans is shown in Table 11. To measure the anomaly detection performance, we consider in each case equal amounts of data from a clean system-under-test (no Trojan) and a system-under-test with the particular Trojan and process both sets of data through our Trojan detection system and record the time series of normal vs. anomalous classifications for the sliding time windows. We then use the time series of normal vs. anomalous classifications to calculate percentages of true positives (with positive defined as marking a time window as anomalous), false positives, true negatives, and false negatives, and then use these percentages to compute accuracy, precision, recall and F1-score. It is seen from Table 11 that high detection accuracy ($\geq$99%) is attained for these Trojans.

We next consider the "stealthy" variants of the LEAK Trojans. As seen in Table 12, these

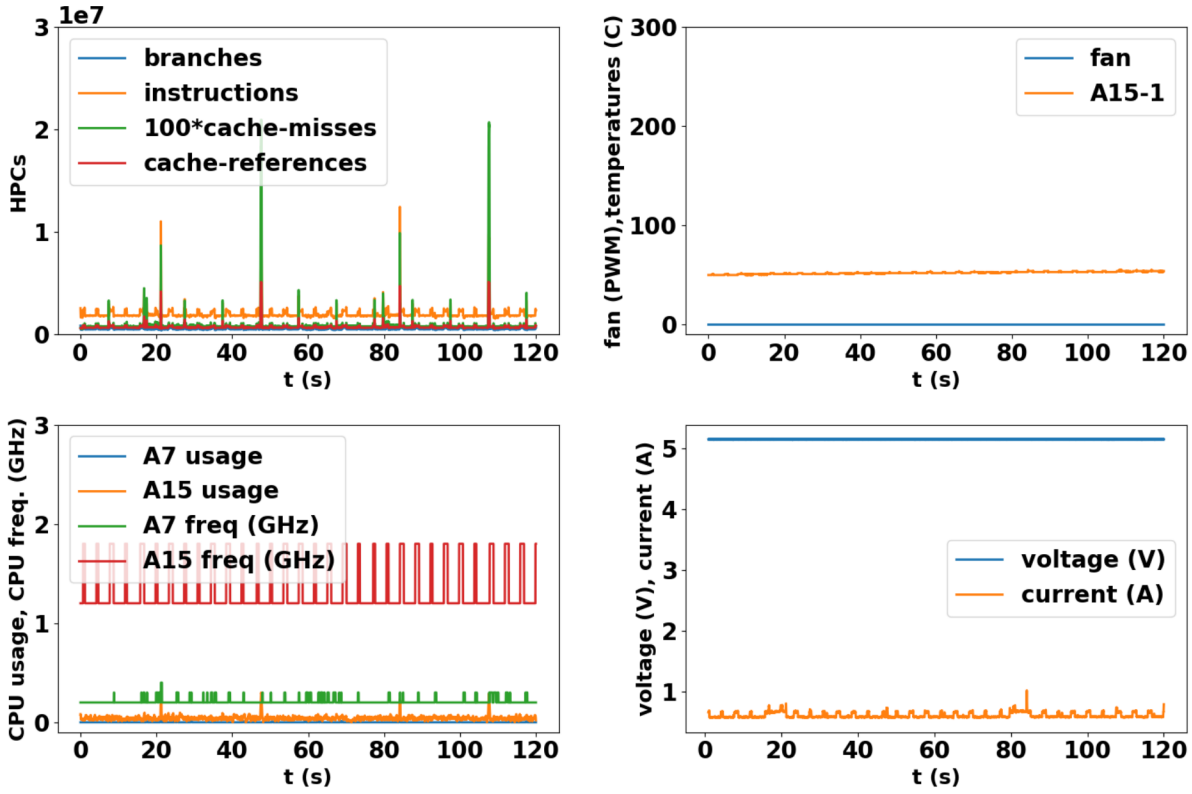


Figure 41: Data collection on XU4 (Quiescent with Info Leak Trojan).

Trojans are effectively invisible when the system-under-test is in quiescent mode since they only perform the information leakage when the system is under load so as to hide in the noise. The side channel signals collected from the PCB with the LEAK.SFAN Trojan are shown in Figure 48 and can be seen to have virtually identical characteristics to the signals collected from the clean (no Trojan) PCB (Figure 22). Similar observations also hold for LEAK.SCPU. Hence, even when all side channel signals are used, their detection accuracy is low ($\leq$ 51%, i.e., random choice or lower), indicating that these Trojans are essentially undetectable unless additional techniques (e.g., fuzzing) are applied. Also, the performance degradation Trojan (PD.DYN) is effectively invisible since its Trojan action is to degrade the performance of the system by artificially throttling the CPU frequency when the system is under load. For all these Trojans, it is seen that fuzzing (in this case, running four instantiations of a test process, one pinned on each of the A15 cores) enables accurate detection of the Trojan since this fuzzing illuminates a tangible difference in side channel

Trojan	Accuracy	Precision	Recall	F1-Score
LEAK.FAN	1.0000	1.0000	1.0000	1.0000
LEAK.CPU	0.9944	0.9890	1.0000	0.9945
LEAK.HYB	0.9976	0.9953	1.0000	0.9976
LEAK.GPIO	0.9931	0.9863	1.0000	0.9931

Table 11: Trojan detection performance on Trojans by considering all side channels and without fuzzing (test process instantiations).

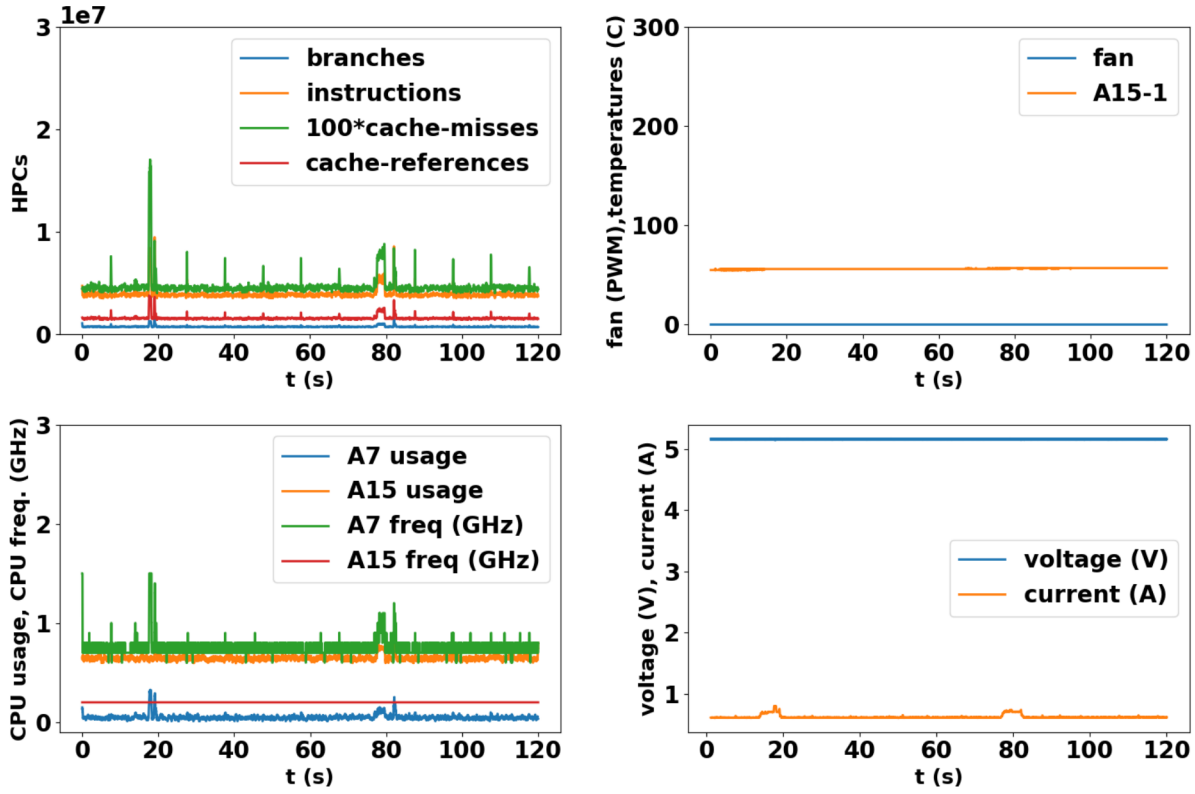


Figure 42: Data collection on XU4 (Quiescent with UART Info Leak Trojan).

measurements between Trojan-free and Trojaned versions of the PCB. For this purpose, a sample image processing application is used as a test process. This application performs the following operations in a loop: open a file containing a color RGB bitmap image and parse header to obtain image metadata; read pixel values, convert to gray-scale by computing a weighted average (using pre-defined floating-point weights) of RGB values, and write pixel values to an output file; close input and output files. Different test processes could be used that exercise integer and floating-point calculations, memory accesses, filesystem reads/writes, other I/O operations, etc. The crucial property of the test processes is that they are defender-chosen to create known signatures of system load unknown to the attacker. The accuracies, precisions, recalls, and F1-scores obtained on each of the stealthy Trojans and the performance degradation Trojan are summarized in Table 12. It is seen that when fuzzing is applied while collecting the side channel data, the performance of the detector increases to $\geq 98\%$. The anomaly detection performance for the hardware replacement Trojan (HWR.A15) is also shown in Table 12. Since the action of this Trojan is defined in terms of the core to which test process is pinned, there is no applicable non-fuzzing scenario when considering this Trojan. Table 12 shows that fuzzing enhances Trojan detection accuracy.

The tests of Trojan detection performance summarized in Tables 11 and 12 utilize all the side channels discussed in Sections 3.4.2 and 4.2.1. To study the utility of fusing side channels, we tested Trojan detection performance for combinations of side channels as summarized in Table 13. Each cell in Table 13 summarizes the performance of the Trojan detector in terms of accuracy, precision, recall, and F1-score. From Table 13, Trojan detection accuracy increases when multiple side channels are used. Specific combinations improve performance depending on the extent to

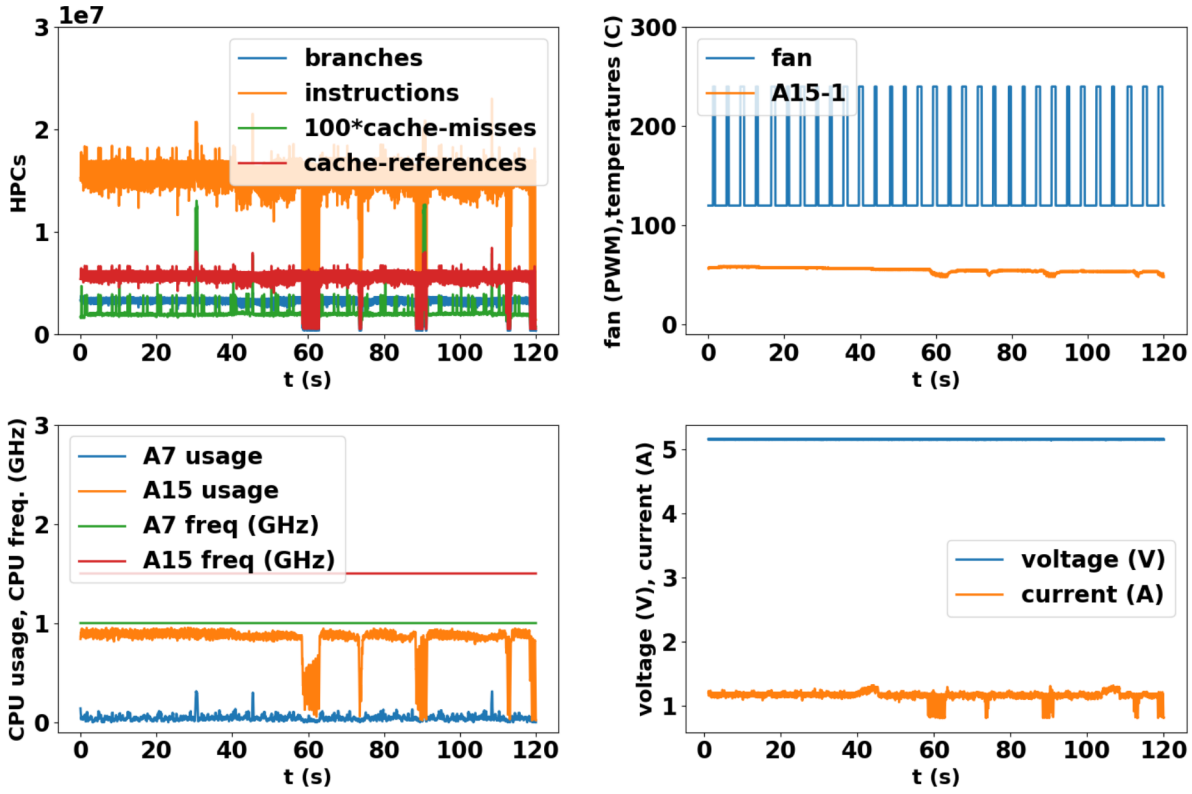


Figure 43: Data collection on XU4 (4 Test Processes on 4 A15 Cores, UART Info Leak Trojan at Fixed Frequency).

which the Trojan impacts that side channel.

We summarize the Trojan detection performance for multi-PCB Trojans in Table 14. For multi-PCB studies, side channel measurements from both PCBs (time synchronization by simply starting measurement code manually on each PCB as close to simultaneous as possible) are concatenated

Trojan	Fuzz	Accuracy	Precision	Recall	F1-Score
LEAK.SCPU	No	0.4846	0.4009	0.0623	0.1079
LEAK.SCPU	Yes	0.9982	0.9964	1.0000	0.9982
LEAK.SFAN	No	0.4846	0.4009	0.0623	0.1079
LEAK.SFAN	Yes	0.9834	0.9888	0.9779	0.9833
LEAK.SHYB	No	0.4946	0.4691	0.0823	0.1400
LEAK.SHYB	Yes	0.9927	1.0000	0.9854	0.9927
PD.DYN	No	0.5023	0.5119	0.0977	0.1641
PD.DYN	Yes	0.9988	0.9976	1.0000	0.9988
PD.STA	No	0.4822	0.4579	0.0932	0.1549
PD.STA	Yes	0.9941	0.9792	0.9951	0.9871
HWR.A15	Yes	0.9944	0.9890	1.0000	0.9945

Table 12: Trojan detection performance for stealthy, performance degradation, and hardware replacement Trojans when considering all side channels with and without fuzzing.

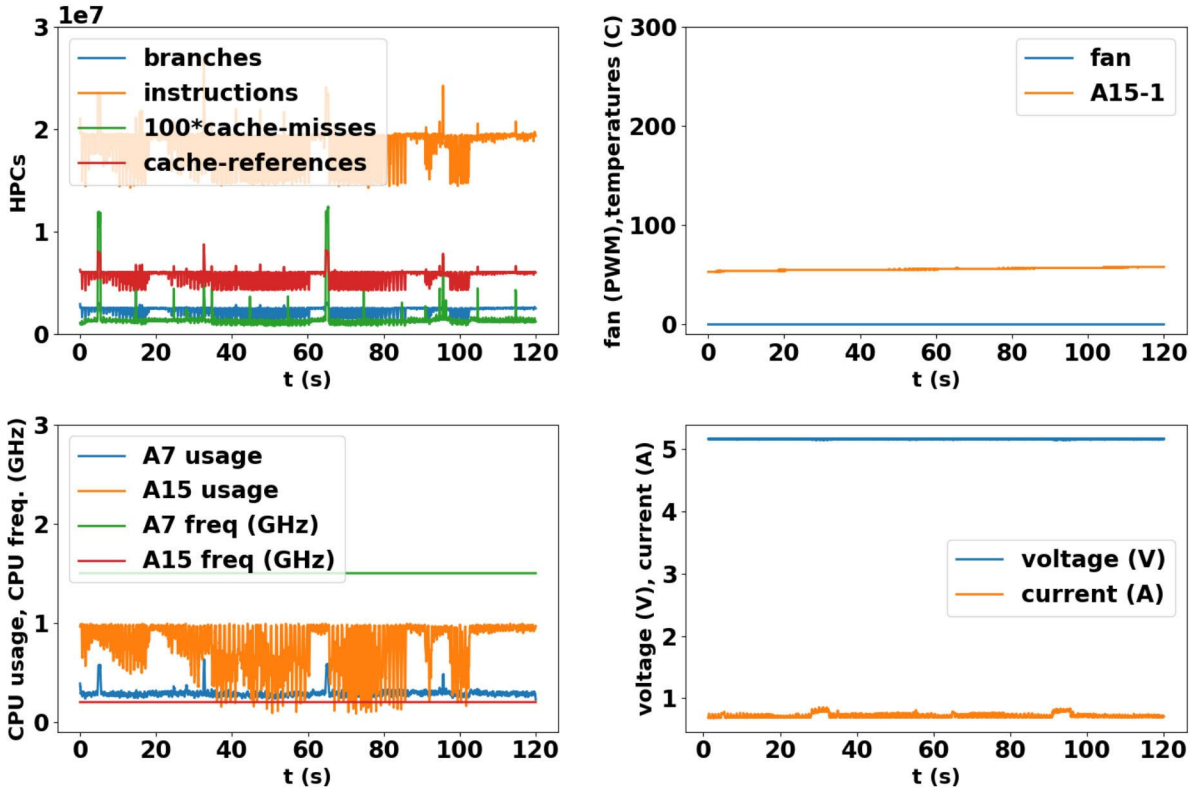


Figure 44: Data collection on XU4 (4 Test Processes on 4 A15 Cores, Performance Degradation Trojan).

for anomaly detection.

Figures 49(a), 49(b), 49(c), and 49(d) show the ROC curves for the Trojan detector with the Trojans LEAK.FAN, LEAK.CPU, LEAK.HYB, and LEAK.GPIO, respectively, when no fuzzing is utilized (i.e., Table 11). Figures 50(a), 50(b), 50(c), 50(d), 50(e), 50(f) show the ROC curves for the Trojans LEAK.GPIO, PD.DYN, LEAK.SCPU, LEAK.SFAN, LEAK.SHYB, and HWR.A15 respectively, when fuzzing is applied (four test processes, one pinned to each of the four cores of the A15 processor), i.e., for the test cases summarized in Table 12.

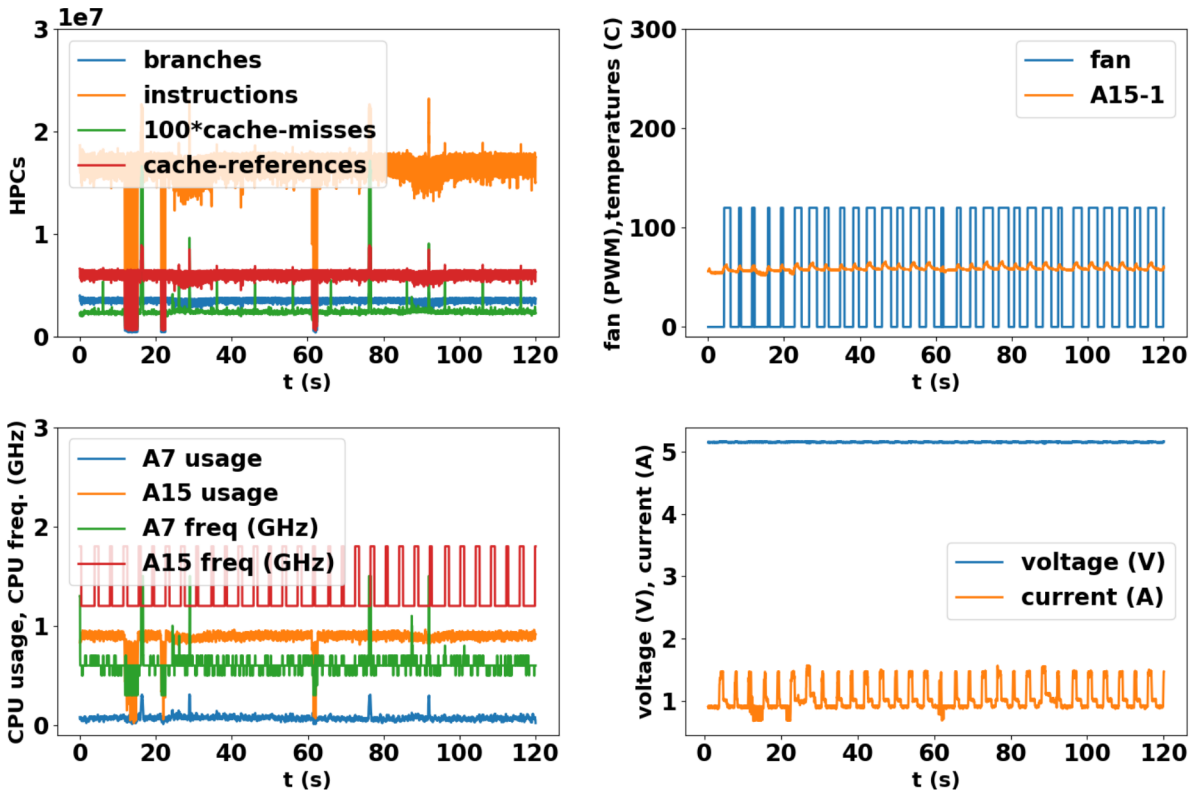


Figure 45: Data collection on XU4 (Test Processes on 4 A15 Cores, Stealthy CPU Info Leak Trojan).

# side channels	Side Channel	LEAK.CPU	LEAK.FAN	LEAK.HYB
One	Power	0.9316, 0.8371, 0.9317, 0.8819	0.8391, 0.8223, 0.8391, 0.8307	0.8936, 0.8326, 0.8933, 0.8618
	Thermal	0.9391, 0.9638, 0.9392, 0.9513	0.9261, 0.9638, 0.9262, 0.9446	0.9347, 0.9621, 0.9343, 0.9479
	HPCs	0.9721, 0.9977, 0.9722, 0.9848	0.9391, 0.9628, 0.9392, 0.9513	0.9549, 0.9743, 0.9542, 0.9641
	CPU Freq.	0.9723, 0.9664, 0.9724, 0.9694	0.8536, 0.8976, 0.8537, 0.8751	0.9127, 0.9213, 0.9128, 0.9170
	Load Avg.	0.6796, 0.6885, 0.6796, 0.6841	0.8659, 0.8776, 0.8658, 0.8717	0.7589, 0.7724, 0.7590, 0.7656
	Fan Speed	0.6671, 0.6598, 0.6672, 0.6635	0.9906, 0.9989, 0.9905, 0.9947	0.7323, 0.7620, 0.7334, 0.7474
Two	Power + Thermal	0.9794, 0.7568, 0.9793, 0.8614	0.9602, 0.7551, 0.9601, 0.8569	0.9720, 0.7543, 0.9712, 0.8491
	Fan Speed + Thermal	0.9798, 0.9652, 0.9799, 0.9725	0.9910, 1.0000, 0.9911, 0.9955	0.9811, 0.9725, 0.9813, 0.9768
	Load Avg. + CPU Freq.	0.9902, 0.9850, 0.9901, 0.9875	0.9173, 0.9352, 0.9172, 0.9259	0.9356, 0.9479, 0.9354, 0.9416
Three	Load Avg. + Thermal + Power	0.9564, 0.9621, 0.9562, 0.9591	0.9656, 0.7833, 0.9657, 0.8649	0.9613, 0.8313, 0.9616, 0.8917
	Load Avg. + CPU Freq. + HPCs	0.9930, 0.9862, 1.0000, 0.9931	0.9529, 0.9763, 0.9525, 0.9642	0.9617, 0.9772, 0.9614, 0.9692
All	Power + Thermal + Load Avg. + CPU Freq. + HPCs + Fan Speed	0.9944, 0.9890, 1.0000, 0.9945	1.0000, 1.0000, 1.0000, 1.0000	0.9976, 0.9953, 1.0000, 0.9976

Table 13: Trojan detection performance by combining side channels and no fuzzing. Cells have accuracy, precision, recall, F1-scores.

Trojan	Accuracy	Precision	Recall	F1-Score
DISTR.BAD	1.0000	1.0000	1.0000	1.0000
DISTR.BIG	1.0000	1.0000	1.0000	1.0000
DISTR.SLW	0.9861	1.0000	0.9723	0.9860

Table 14: Trojan detection performance on multi-PCB Trojans using all side channels and no fuzzing.

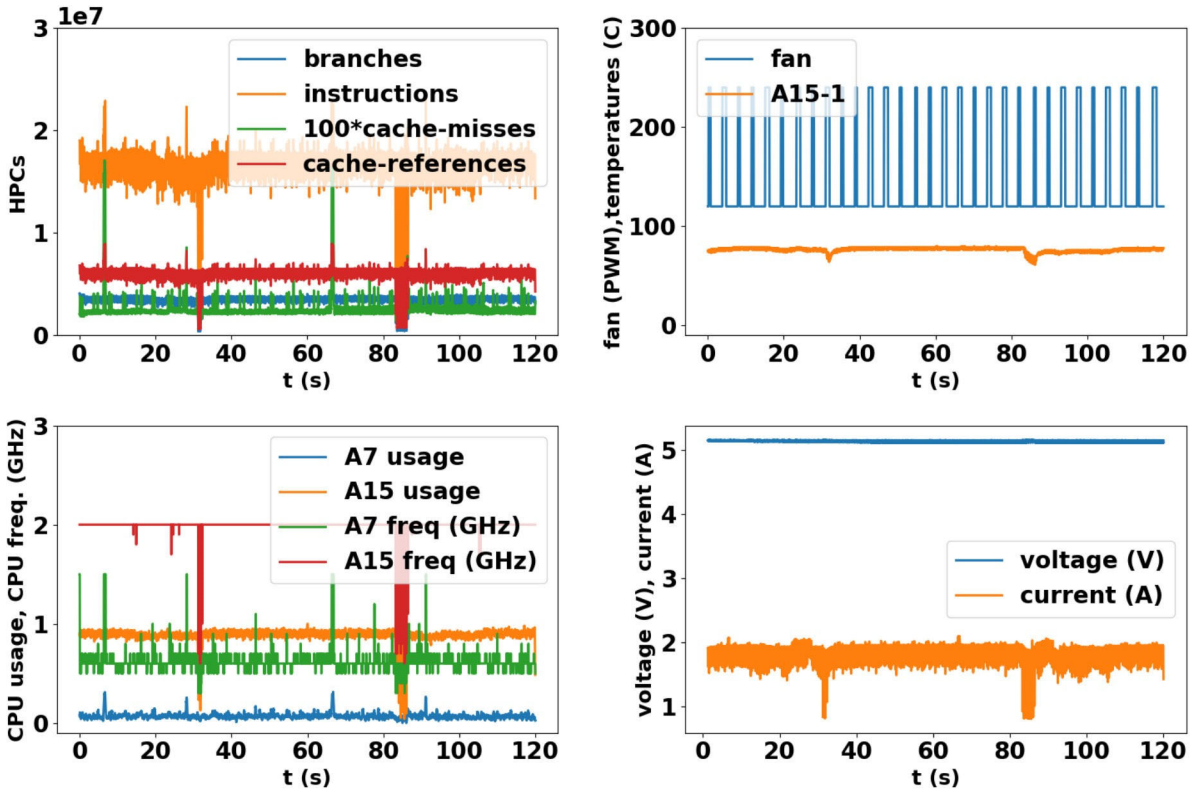


Figure 46: Data collection on XU4 (Test Processes on 4 A15s, Stealthy Fan Info Leak Trojan).

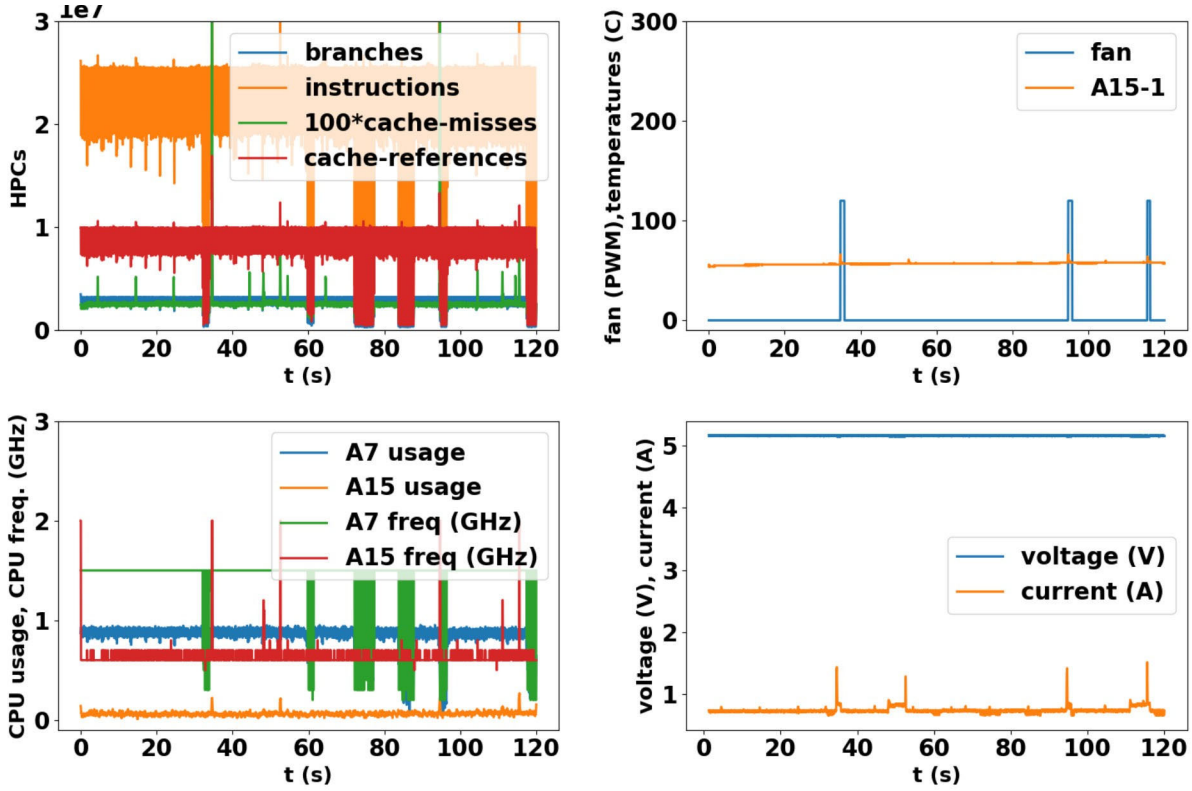


Figure 47: Data collection on XU4 (Test Processes on 4 A15s, A15 → A7 replacement Trojan).

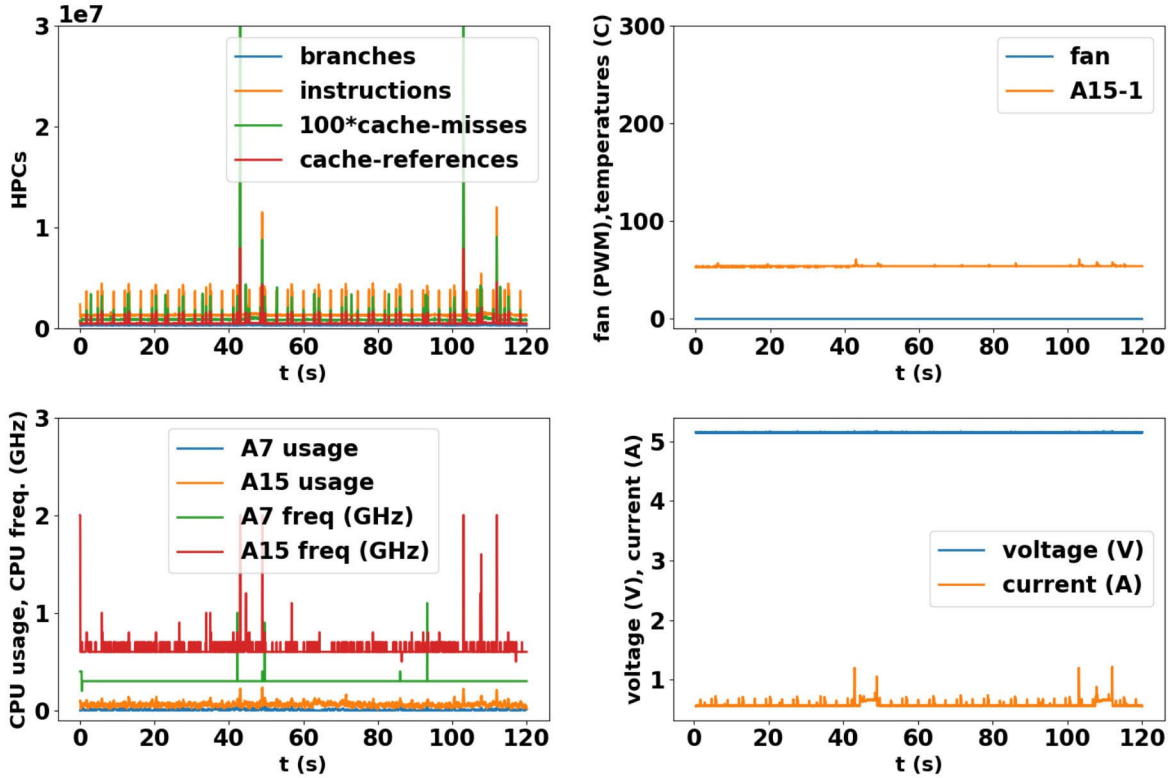
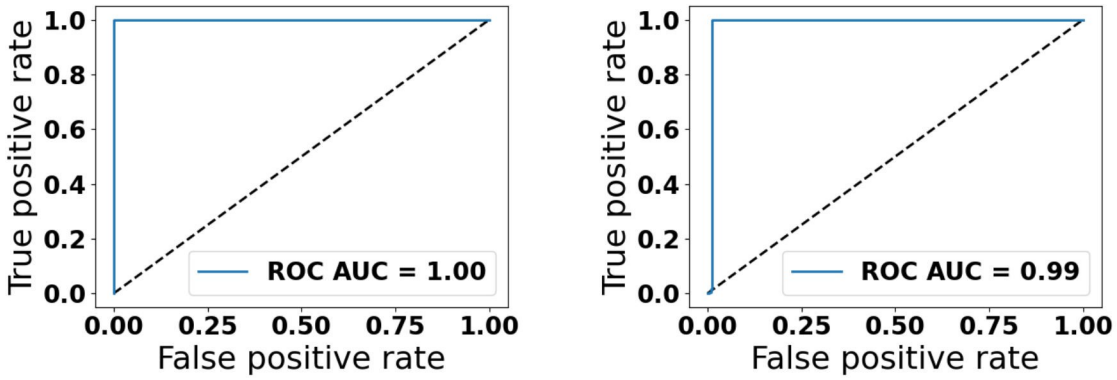


Figure 48: Data Collection on XU4: Quiescent, Stealthy Fan Info Leak Trojan.



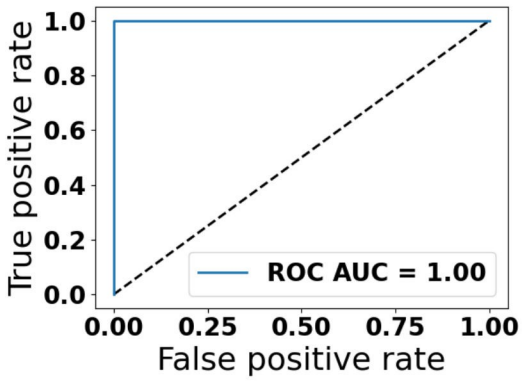
(a) LEAK.FAN

(b) LEAK.CPU

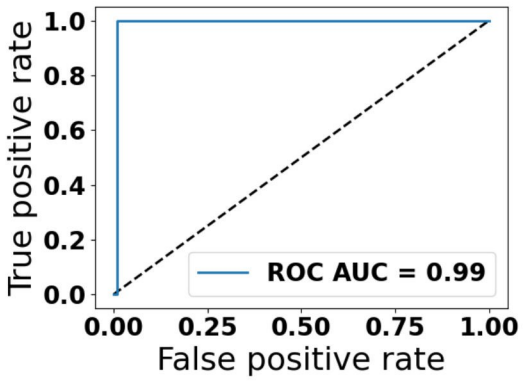
(c) LEAK.HYB

(d) LEAK.GPIO

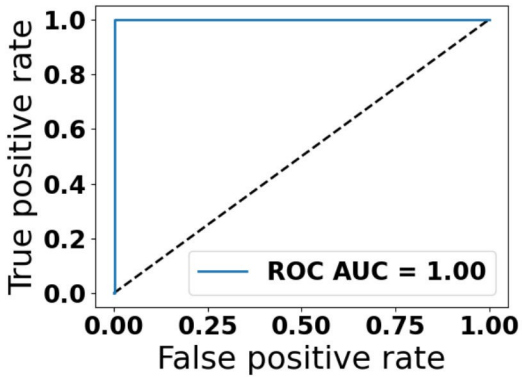
Figure 49: ROC curves for Trojan detection without test processes.



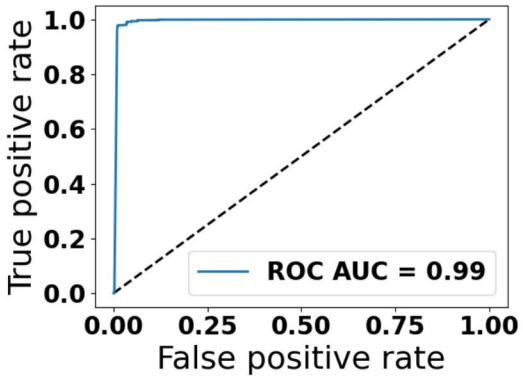
(a) LEAK.GPIO



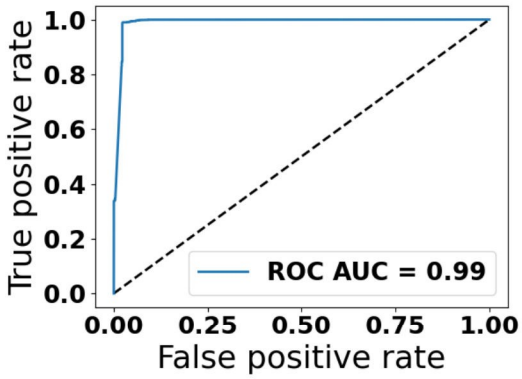
(b) PD.DYN



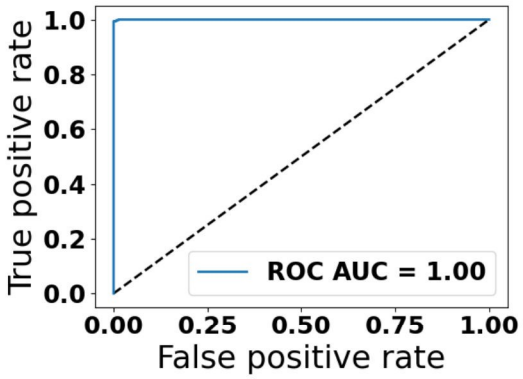
(c) LEAK.SCPU



(d) LEAK.SFAN



(e) LEAK.SHYB



(f) HWR.A15

Figure 50: ROC curves for Trojan detection when test processes run on the four cores.

5.0 Conclusions

In this project, we developed methods for golden-free detection of Trojans in a multi-PCB COTS system. The methodologies developed in this project synergistically combine graph-theoretic analysis of equivalence/invariance properties, behavioral specifications, and ML classifier-based anomaly likelihood estimation. Multi-modal side channel signals are collected under controlled excitations by hardware and software fuzzing. The multi-modal side channel signal time series measurements are fused based on the multiple anomaly detection techniques developed in this project to detect presence of Trojans. A simulator-based approach is used to generate training data for the ML anomaly detector. Expected signal interrelationships (behavioral rules) inferred from design information are used by a behavioral analyzer together with the ML-based anomaly detector in an ensemble structure. Equivalence/invariance properties expected from identical/similar devices and interconnected groups of devices are analyzed to detect statistical dissimilarities indicating presence of Trojans. Experimental studies on a wide range of Trojans in single-PCB and multi-PCB scenarios show that the golden-free Trojan detection methodology is a promis- ing direction for side channel based detection of anomalies without requiring a golden PCB. The experimental results demonstrate the efficacy of fusing multiple side channels and the utility of fuzzing in amplifying the Trojan visibility. Directions for future work include:

- Validation of the approach on additional PCBs
- Leveraging more sources of design information for implementation of the Trojan detection techniques
- Further red-team testing on wider classes of Trojans – iterative blue-team/red-team exercises including wider variety of hardware platforms
- Refinements of the behavioral simulator; generalizations to distributed simulator with heterogeneous inter-board communications; mixed-mode simulators combining circuit models, component characterizations, processor simulators, and communication models
- Refinements of the ML-based anomaly detectors, behavioral analyzers, ensemble models, and on-demand and on-line continuous integrity verification systems
- Exploration of additional degrees of freedom in fuzzing, refinement of sequential fuzzing methodology for Trojan illumination

6.0 References

[1] J. Rajendran, E. Gavas, J. Jimenez, V. Padman, and R. Karri, "Towards a Comprehensive and Systematic Classification of Hardware Trojans," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, May 2010, pp. 1871–1874, iSSN: 2158-1525.

[2] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Trustworthy Hardware: Identifying and Classifying Hardware Trojans," *Computer*, vol. 43, no. 10, pp. 39–46, Oct. 2010, conference Name: Computer.

[3] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware Trojans: Lessons Learned after One Decade of Research," *ACM Transactions on Design Automation of Electronic Systems*, vol. 22, no. 1, pp. 6:1–6:23, May 2016. [Online]. Available: https://doi.org/10.1145/2906147

[4] Defense Advanced Research Projects Agency (DARPA), "Microsystems Exploration: Safeguards against Hidden effects and Anomalous Trojans in Hardware (SHEATH)," 2019. [Online]. Available: https://sam.gov/opp/25d32ec9a35117576b04d52896267319/view

[5] J. Robertson and M. Riley, "The Big Hack: How China Used a Tiny Chip to Infiltrate U.S. Companies," *Bloomberg*, Oct. 2018. [Online]. Available: https://www.bloomberg.com/news/features/2018-10-04/the-big-hack-how-china-used-a-tiny-chip-to-infiltrate-america-s-top-companies

[6] ——, "The Long Hack: How China Exploited a U.S. Tech Supplier," *Bloomberg*, Feb. 2021. [Online]. Available: https://www.bloomberg.com/features/2021-supermicro/

[7] S. Paley, T. Hoque, and S. Bhunia, "Active Protection against PCB Physical Tampering," in *2016 17th International Symposium on Quality Electronic Design (ISQED)*, Mar. 2016, pp. 356–361, iSSN: 1948-3295.

[8] M. McGuire, U. Ogras, and S. Ozev, "PCB Hardware Trojans: Attack Modes and Detection Strategies," in *2019 IEEE 37th VLSI Test Symposium (VTS)*, Apr. 2019, pp. 1–6, iSSN: 2375-1053.

[9] G. Piliposyan, S. Khursheed, and D. Rossi, "Hardware Trojan Detection on a PCB Through Differential Power Monitoring," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2020, conference Name: IEEE Transactions on Emerging Topics in Computing.

[10] D. Mehta, H. Lu, O. P. Paradis, M. A. M. S., M. T. Rahman, Y. Iskander, P. Chawla, D. L. Woodard, M. Tehranipoor, and N. Asadizanjani, "The Big Hack Explained: Detection and Prevention of PCB Supply Chain Implants," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 16, no. 4, pp. 42:1–42:25, Aug. 2020. [Online]. Available: https://doi.org/10.1145/3401980

[11] T. Hudson, "Modchips of the State," 35th Chaos Communication Congress (35C3), Dec. 2018. [Online]. Available: https://media.ccc.de/v/35c3-9597-modchips_of_the_state

[12] H. Zhu, X. Guo, Y. Jin, and X. Zhang, "PCBench: Benchmarking of Board-Level Hardware Attacks and Trojans," in *Proceedings of 2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan. 2021, pp. 396–401, iSSN: 2153-697X.

[13] S. Ghosh, A. Basak, and S. Bhunia, "How Secure Are Printed Circuit Boards Against Trojan Attacks?" *IEEE Design Test*, vol. 32, no. 2, pp. 7–16, Apr. 2015, conference Name: IEEE Design Test.

[14] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, "Trojan Detection using IC Fingerprinting," in *Proceedings of 2007 IEEE Symposium on Security and Privacy (SP '07)*, 2007, pp. 296–310.

[15] Y. Liu, K. Huang, and Y. Makris, "Hardware Trojan Detection through Golden Chip-Free Statistical Side-Channel Fingerprinting," in *DAC*, 2014, pp. 1–6.

[16] Y. Jin and Y. Makris, "Hardware Trojan Detection using Path Delay Fingerprint," in *IEEE HOST*, 2008, pp. 51–57.

[17] N. K. Patel, P. Krishnamurthy, H. Amrouch, J. Henkel, M. Shamouilian, R. Karri, and F. Khorrami, "Towards a New Thermal Monitoring Based Framework for Embedded CPS Device Security," *IEEE TODASC*, p. (Early access), 2020.

[18] D. Paul-Pena, P. Krishnamurthy, R. Karri, and F. Khorrami, "Process-Aware Side Channel Monitoring for Embedded Control System Security," in *Proceedings of IEEE VLSI-SoC*, 2017, pp. 1–6.

[19] P. Krishnamurthy, R. Karri, and F. Khorrami, "Anomaly Detection in Real-Time Multi-Threaded Processes using Hardware Performance Counters," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 666–680, June 2020.

[20] Y. Zhang, H. Quan, X. Li, and K. Chen, "Golden-Free Processor Hardware Trojan Detection Using Bit Power Consistency Analysis," *Springer JETTA*, vol. 34, no. 3, pp. 305–312, 2018.

[21] F. Karabacak, U. Y. Ogras, and S. Ozev, "Detection of Malicious Hardware Components in Mobile Platforms," in *IEEE ISQED*, 2016, pp. 179–184.

[22] B. Shakya, T. He, H. Salmani, D. Forte, S. Bhunia, and M. Tehranipoor, "Benchmarking of Hardware Trojans and Maliciously Affected Circuits," *Journal of Hardware and Systems Security*, vol. 1, no. 1, pp. 85–102, 2017.

[23] T. Hoque, S. Yang, A. Bhattacharyay, J. Cruz, and S. Bhunia, "An Automated Framework for Board-level Trojan Benchmarking," *arXiv:2003.12632 [cs]*, Mar. 2020, arXiv: 2003.12632. [Online]. Available: http://arxiv.org/abs/2003.12632

[24] C. Dong, Y. Xu, X. Liu, F. Zhang, G. He, and Y. Chen, "Hardware Trojans in Chips: A Survey for Detection and Prevention," *Sensors*, vol. 20, no. 18, p. 5165, 2020.

[25] J. Harrison, N. Asadizanjani, and M. Tehranipoor, "On Malicious Implants in PCBs Throughout the Supply Chain," *Integration*, vol. 79, pp. 12–22, Jul. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167926021000304

[26] S. Wakabayashi, S. Maruyama, T. Mori, S. Goto, M. Kinugawa, Y.-i. Hayashi, and M. Smith, "A Feasibility Study of Radio-Frequency Retroreflector Attack," in *USENIX Workshop on Offensive Technologies (WOOT 18)*, 2018.

[27] "NSA ANT Catalog," 2013, Electronic Frontier Foundation. [Online]. Available: https://www.eff.org/document/20131230-appelbaum-nsa-ant-catalog

[28] J. Winter and K. Dietrich, "A Hijacker's Guide to Communication Interfaces of the Trusted Platform Module," *Computers & mathematics with applications*, vol. 65, no. 5, pp. 748–761, 2013.

[29] J. Frazelle, "Opening up the Baseboard Management Controller," *Communications of the ACM*, vol. 63, no. 2, pp. 38–40, 2020.

[30] NCC Group *et al.*, "TPM Genie: Interposer Attacks against the Trusted Platform Module Serial Bus," 2018.

[31] R. Gonggrijp and W.-J. Hengeveld, "Studying the Nedap/Groenendaal ES3B Voting Computer: A Computer Security Perspective," in *USENIX Workshop on Accurate Electronic Voting Technology*, 2007.

[32] A. Greenberg, "Planting Tiny Spy Chips in Hardware Can Cost as Little as $200," *Wired*, Oct. 2019. [Online]. Available: https://www.wired.com/story/plant-spy-chips-hardware-supermicro-cheap-proof-of-concept/

[33] B. Blaxill and J. Sandin, "PicoDMA: DMA Attacks at your Fingertips," 2019. [Online]. Available: https://i.blackhat.com/USA-19/Wednesday/us-19-Sandin-PicoDMA-DMA-Attacks-At-Your-Fingertips.pdf

[34] S. Bhunia and M. M. Tehranipoor, Eds., *The Hardware Trojan War: Attacks, Myths, and Defenses*, 1st ed. Cham: Springer International Publishing : Imprint: Springer, 2018.

[35] T. Roth, J. Datko, and D. Nedospazov, "Chip.fail: Glitching the Silicon of the Connected World," 2019. [Online]. Available: https://www.blackhat.com/us-19/briefings/schedule/index.html#chipfail---glitching-the-silicon-of-the-connected-world-15192

[36] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware Trojan Attacks: Threat Analysis and Countermeasures," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1229–1247, 2014.

[37] A. Hennessy, Y. Zheng, and S. Bhunia, "JTAG-based Robust PCB Authentication for Protection against Counterfeiting Attacks," in *Asia and South Pacific Design Automation Conference*, 2016, pp. 56–61.

[38] J. R. Hamlet, M. T. Martin, and N. J. Edwards, "Unique Signatures from Printed Circuit Board Design Patterns and Surface Mount Passives," in *2017 International Carnahan Conference on Security Technology (ICCST)*, 2017, pp. 1–6.

[39] Z. Guo, X. Xu, M. M. Tehranipoor, and D. Forte, "MPA: Model-Assisted PCB Attestation via Board-Level RO and Temperature Compensation," in *Proceedings of Asian Hardware Oriented Security and Trust Symp.*, 2017, pp. 25–30.

[40] D. Zhang, Y. Han, and Q. Ren, "A Novel Authorization Methodology to Prevent Counterfeit PCB/Equipment through Supply Chain," in *2019 IEEE 4th International Conference on Integrated Circuits and Microsystems (ICICM)*, 2019, pp. 128–132.

[41] X. Wang, Y. Han, and M. Tehranipoor, "System-Level Counterfeit Detection using On-Chip Ring Oscillator Array," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 12, pp. 2884–2896, 2019.

[42] C. Sun, L. Y. Cheng, L. W. Wang, Q. Huang, Y. Huang, and G. L. Feng, "A Machine Learning Method for Hardware Trojan Detection on Real Chips," *AIP Advances*, vol. 11, no. 5, p. 055006, May 2021, publisher: American Institute of Physics. [Online]. Available: https://aip.scitation.org/doi/full/10.1063/5.0038773

[43] N. Shang, A. Wang, Y. Ding, K. Gai, L. Zhu, and G. Zhang, "A Machine Learning Based Golden-Free Detection Method for Command-Activated Hardware Trojan," *Information Sciences*, vol. 540, pp. 292–307, Nov. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025520304679

[44] V. R. Surabhi, P. Krishnamurthy, H. Amrouch, K. Basu, J. Henkel, R. Karri, and F. Khorrami, "Hardware Trojan Detection using Controlled Circuit Aging," *IEEE Access*, vol. 8, pp. 77 415–77 434, 2020.

[45] V. R. Surabhi, P. Krishnamurthy, H. Amrouch, J. Henkel, R. Karri, and F. Khorrami, "Exposing Hardware Trojans in Embedded Platforms via Short-Term Aging," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3519–3530, 2020.

[46] A. C. De Melo, "The New Linux 'perf' Tools," in *Slides from Linux Kongress*, vol. 18, 2010, pp. 1–42.

[47] P. J. Mucci, S. Browne, C. Deane, and G. Ho, "PAPI: A Portable Interface to Hardware Performance Counters," in *Proceedings of the Department of Defense HPCMP users group conference*, vol. 710, 1999.

[48] P. Bridges, N. Doss, W. Gropp, E. Karrels, E. Lusk, and A. Skjellum, “Users’ Guide to MPICH, a Portable Implementation of MPI,” *ANL*, vol. 9700, pp. 60 439–4801, 1995.

7.0 List of Symbols, Abbreviations, and Acronyms

COTS	Commercial Off The Shelf
DTW	Dynamic Time Warping
DVFS	Dynamic Voltage and Frequency Switching
HPC	Hardware Performance Counter
IC	Integrated Circuit
MITM	Machine In The Middle
ML	Machine Learning
PCB	Printed Circuit Board
SBC	Single Board Computer
VLAN	Virtual Local Area Network