XMD: An Expansive Hardware-telemetry based Malware Detector to enhance Endpoint Detection

Harshit Kumar, Biswadeep Chakraborty, Sudarshan Sharma, Nikhil Chawla, Saibal Mukhopadhyay

Dept. of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, USA
{hkumar64, biswadeep, ssharma497, nikhilchawla}@gatech.edu, saibal.mukhopadhyay@ece.gatech.edu

Abstract

Hardware-based Malware Detectors (HMDs) have shown promise in detecting malicious workloads. However, the current HMDs focus solely on the CPU core of a System-on-Chip (SoC) and, therefore, do not exploit the full potential of the hardware telemetry. In this paper, we propose XMD, an HMD that operates on an expansive set of telemetry channels extracted from the different subsystems of SoC. Key innovations in XMD are guided by analytical theorems that leverage the concept of manifold hypothesis. XMD exploits the thread-level profiling power of the CPU-core telemetry, and the global profiling power of non-core telemetry channels, to achieve significantly better detection performance and concept drift robustness than currently used Hardware Performance Counter (HPC) based detectors. We train and evaluate XMD using hardware telemetries collected from 904 benign applications and 1205 malware samples. XMD improves over currently used HPC-based detectors by 32.91% for the in-distribution test data and by 67.57% for the concept drift test data. XMD achieves the best detection performance of 86.54% with a false positive rate of 2.9%, compared to the detection rate of 80%, offered by the best performing software-based Anti-Virus (AV) on VirusTotal, on the same set of malware samples.

1 Introduction

The previous decade has witnessed an explosive growth of malicious applications, compromising the security of modern devices [19]. Consequently, Endpoint security has moved towards behavior analysis that involves continuous monitoring of sensors across the compute stack, and rigorous data analysis [14, 73]. As shown in Figure 1, behavior analysis techniques monitor the program’s execution using semantically rich information sources like registry keys, network endpoints, system calls, and operating system (OS) hooks. However, malicious actors can potentially subvert such protection mechanisms by tampering with the software telemetry [6, 46].

Moreover, such software-level detection approaches result in significant performance overhead. Hence, there has been a recent thrust in data-driven approaches for detecting malicious workloads using low-level hardware telemetry, which promises low overheads and better resilience against tampering compared to software-based telemetry [39].

The application of hardware-level telemetry, like HPC, energy telemetry channels (e.g., Intel’s RAPL), and Dynamic Voltage and Frequency Scaling (DVFS), towards malware detection, has recently gained interest [24, 32, 36, 39, 45, 47, 49, 53–55, 63, 72, 76]. Monitoring the hardware telemetry provides visibility into active threats, even during the presence of anti-evasion techniques like obfuscation [47], or cloaking in virtual machines [6]. As shown in Fig 1, HMD along with software-level detection techniques, form part of the commercially-deployed collaborative defense model, e.g., Endpoint Detection and Response (EDR), that detects and contains threats [6, 10, 11, 17]. In such a collaborative system, HMDs can provide an additional layer of detection capability across the kill chains, complement other detectors, and restrict the ability of the adversary to move in the environment without triggering detection.

Demme et al. demonstrated proof-of-concept of detecting Android malware, Linux Rootkits, and side-channel attacks using HPCs [39]. Thereafter, methodologies for increasing the predictive performance [36, 45, 49, 53, 54], lowering the overheads of the HMDs [63, 72], and alternate CPU-telemetry sources [32] have been proposed. However, prior works focus solely on the hardware telemetry from the CPU of the System-on-Chip (SoC), which captures the partial impact of running workloads on an SoC, and does not exploit the full potential of the extensive hardware telemetry. Moreover, these works do not address the concept drift arising from the behavioral evolution (adversarial or benign) of applications. The resulting change in test data distribution causes severe performance degradation of the deployed ML models, posing a critical challenge in deploying models for security applications.

Approach. In this work, we propose XMD, an HMD that achieves significant improvements in detection performance...
and concept drift robustness compared to prior HPC-based HMDs. To accomplish this, XMD exploits two key innovations. First, to improve detection performance, XMD uses a comprehensive set of telemetry channels that capture a workload’s impact on different SoC sub-systems like GPU, memory, buses, network, and CPU telemetry, the primary focus of prior works. XMD achieves this through the use of DVFS signatures from the non-core devices, and low-level telemetry from select SYSFS nodes, in addition to the existing CPU telemetry channels like CPU-DVFS and HPC. Second, XMD exploits the non-determinism (stochasticity) present in DVFS and SYSFS channels (collectively referred to as GLOBL channels) in the rest of the paper) to achieve significant concept drift robustness compared to prior HPC-based HMDs.

The two key innovations are grounded in the theorems we developed by leveraging the concept of manifold hypothesis (Section 4). Prior works in deep learning have used manifold hypothesis for studying the geometry of manifolds in Deep Neural Networks (DNNs) for vision [35], audition [69], and language modeling [25,60]. Such analytical frameworks have explained the generalization performance of DNNs [70]. In this work, we use manifold hypothesis to connect the stochasticity of the GLOBL channels to concept drift robustness, and an increase in the solution volume of the fusion-classifier [42] to the superior classification performance of XMD.

To evaluate XMD, we create a robust data-collection framework, using a commodity mobile device (Table 5), that incorporates measures to reduce the dataset bias and prevent over-optimistic results. Using this data-collection framework, we create three different datasets, which help us empirically validate our developed theorems. For example, we observe that while the HPCs can profile threads pretty accurately in a biased dataset (F1 score 0.99-1.0), their performance degrades severely under concept-drift scenarios (F1 score 0.46-0.56). On the other hand, the stochasticity present in the DVFS and SYSFS-based telemetry results in concept-drift robustness at the cost of degradation in profiling accuracy. Finally, we compare the detection performance of XMD with commercial AV engines on VirusTotal. To summarize, our main contributions are as follows:

- We develop two analytical theorems, using manifold hypothesis, which proves that the stochasticity in hardware telemetry results in concept drift robustness and using multiple telemetry channels increases the detection performance of the HMD.
- We design a robust bare-metal data collection framework to collect expansive hardware-based telemetry that captures a workload’s impact on different subsystems of the SoC like CPU, GPU, Memory, Buses, and Network.
- Using three kinds of datasets, we perform empirical experiments to validate the proposed theorems. Specifically, we show that: (1) higher stochasticity in the GLOBL channels reduces the performance degradation when tested on a concept drift dataset compared to HPC-based detectors, and (2) using multiple telemetry channels from the different subsystems of an SoC results in better classification performance when compared to classifiers that operate on the telemetry from a single subsystem, e.g., CPU-based telemetry.
- Finally, we develop a fusion-based classifier, called XMD, that combines the strength of the two classes of telemetry to provide better performance than prior HMDs. We benchmark XMD against commercial software-based AV engines present on VirusTotal and find that XMD provides superior detection rates, with an acceptable false-positive rate.

2 Background and Threat Model

2.1 Classes of Telemetry

Thread-level Profiling : Hardware Performance Counters (HPCs). HPCs are dedicated physical registers in modern processors that store the count of the number of microarchitectural events in a CPU core during process execution. They were originally designed to identify performance bottlenecks. To obtain counter information from the registers, they are configured to monitor specific hardware events that are of interest. Although hundreds of hardware events are available for monitoring, a limited number of them can be
monitored simultaneously due to the limited number of physical registers (e.g., 4 in the Snapdragon chipset considered in this paper). HPCs provide the ability to perform thread-level profiling by saving the register values during context switches and, therefore, try to avoid contamination due to events from other processes [38]. With the adoption of ML techniques in security, HPCs have recently been repurposed as a low-level telemetry for identifying malicious workloads in HMDs [24, 36, 39, 45, 47, 49, 54, 55, 63, 72, 76].

Global Profiling: Dynamic Voltage and Frequency Scaling (DVFS). DVFS is an integral part of all power management systems. It reduces the power consumption of an SoC by scaling down the voltage and frequency states of the different sub-systems (e.g., CPU, GPU, buses, caches, and memory) based on the targetted performance requirements of the software workloads. As a result, the DVFS states of a sub-system capture its activity level, providing insight into the impact of running workload on that sub-system. Security implications of the DVFS framework have been studied both from an offensive perspective [52, 61, 71] and to create defenses [31, 32]. Since Android OS (considered in this paper) is based on the Linux kernel, the DVFS states of the CPU and non-CPU devices are accessible through the cpufreq and devfreq framework [5, 7].

It should be noted that the DVFS channels (and other SYSFS nodes used in this work) capture the global state of the device as compared to HPCs that are used for monitoring the specific threads/processes. In mobile devices, numerous system-level threads and user applications are simultaneously contending for hardware resources. This makes the DVFS channels susceptible to noise arising from such background processes. However, in the case of mobile devices, such as the one considered in our work, the foreground applications contribute towards deciding the DVFS states most of the time [32]. The DVFS channels provide better visibility into the impact of running a workload on the entire SoC at the cost of additional non-determinism compared to HPCs that capture a workload’s impact on the CPU core of the SoC.

2.2 Concept Drift (CD)

A central assumption of training a supervised ML algorithm is that the training and the test data points are sampled independently from the same distribution. However, in the case of malware detectors deployed in the wild, the underlying probability distribution of the test data points changes. Such a change may arise from the evolution of tactics, techniques, and procedures (TTPs) employed by the malware authors or can result from the organic behavioral changes in benign applications. This phenomenon of the shift in the true decision boundary is called Concept Drift [44, 53].

Current measures for mitigating the performance degradation arising from CD involve detecting such scenarios and re-training the ML model. The re-trained model is deployed to the client devices via security updates. However, if the underlying telemetry for the ML model is sensitive to CD, then the frequency of periodical re-training increases (expensive). Therefore, CD robustness can reduce the costs associated with maintaining an ML model deployed in a security-sensitive application.

2.3 Threat Model

The threat model of XMD is the same as the HMDs that are commercially deployed, i.e., we assume that the OS kernel is not compromised. Since HMDs are a part of the EDR system, a compromised kernel undermines the process tracking capabilities of the EDR and thwarts the trustworthiness of the different sources of telemetry on which the EDR relies upon [46]. We note that hardware implementations of HMD have been explored for providing detection capabilities under threat models where the OS is compromised [62, 63, 72].

3 Related Works and Motivation

Detection of malicious workloads using hardware telemetry has been extensively studied in the literature [24, 32, 36, 39, 45, 47, 49, 54, 55, 63, 72, 76]. Due to the variabilities in the data collection methodology, we do not empirically compare against prior works, but offer a qualitative comparison against the surveyed works. As summarized in Table 1, we categorize the drawbacks of these works as follows:

Restricted scope of hardware telemetry collection. Prior works on HMD primarily focus on a single modality of data extracted from the CPU of the SoC [24, 32, 36, 39, 45, 47, 49, 54, 55, 63, 72, 76]. These low-level signatures either contain thread-level behavior (e.g., HPC) or global behavior (e.g., CPU-DVFS). While these data-driven approaches have shown decent test accuracy, they do not use the telemetry sources available from the different sub-systems of the SoC. They, therefore, do not realize the full potential of the HMD.

Evaluation on concept drift scenarios. Prior works in HMDs have only been tested on in-distribution malware and benign samples that are from the same period as that of the training data [32, 36, 39, 45, 47, 49, 54, 55, 63, 72, 76]. Demme et al. state in their work that "regardless of how malware writers change their software, its semantics do not change significantly" [39]. A similar argument can be extended to concept drift scenarios, wherein irrespective of the high-level changes in TTPs employed by malware authors; the underlying malicious objective remains the same, e.g., file encryption in crypto-ransomware. Therefore, in principle, the feature space on which HMDs operate should exhibit some robustness to concept drift scenarios. However, none of the prior works have evaluated their models on a concept drift dataset.

Bias in datasets. Creating a dataset for training and evaluating an HMD requires executing the malware and benign
applications in a sandbox and collecting the behavioral signatures in the background. Incorrect assumptions while designing the data collection framework can introduce bias in the dataset. We have identified sources of bias in the data collection frameworks of prior works like failure to verify the execution of the profiled applications and the use of benchmarks as benign applications (Appendix B.1). As we observe in this paper, a non-trivial percentage of applications do not have sufficient runtime during automated telemetry collection (Section 5.4), and benchmarks have the potential to introduce bias in the dataset (Section 6.1.1).

Comparison against Software-based AVs. Prior works present a qualitative comparison of how the behavior-based detection approach of HMDs can outperform the static analysis-based detection techniques of production AV software [24, 32, 36, 39, 45, 54, 55, 63, 72, 76]. However, they do not present a quantitative comparison of how the performance of their proposed HMDs compares against the currently deployed production AV software.

Lack of a unified open-source dataset. Despite the plethora of works on HMDs, a handful of publically available datasets for HMDs restrict the research into developing and testing novel ML agents. Datasets that have been shared are lacking in aspects that have been summarized in Table 1 [32, 39, 76]. Due to the huge variability observed in the behaviors of malware and benign workloads arising from limitations of dynamic analysis studies [28], there is a need for a standardized dataset.

4 Theory of XMD

We describe the main hypotheses behind the design of XMD, followed by the theorems that support our hypotheses.

4.1 Intuition behind XMD

Hypothesis 1: The non-determinism in hardware telemetry results in increased concept drift robustness. Recent works have identified failure scenarios arising from the non-determinism in the hardware telemetry [38, 76]. However, adding synthetic noise to the input of an ML model, forcing the model to learn a more general solution, is a well-known concept in the ML community [67]. Hence, we make the following hypotheses: (1.1) the stochasticity present in the hardware telemetry acts as a regularizer and results in better generalization, (1.2) a direct consequence of learning a general solution will be increased concept drift robustness.

Hypothesis 2: A better classification performance can be achieved with an expansive set of hardware telemetry channels. As pointed out by Zhou et al., it is unclear why the low-level telemetry extracted solely from the CPU of the SoC can help distinguish high-level behavior between benign and malicious applications [76]. For example, the encryption instructions used by a workload can be employed by ransomware for encrypting user files or by benignware like 7zip. Using hardware telemetry from solely the CPU does not provide the complete picture to the ML agents operating on the telemetry. On the other hand, if the ML agent has access to the low-level network telemetry, it can potentially identify the encryption performed by the ransomware since the malware will additionally communicate with its C2 server. This leads us to our hypothesis that an ML classifier will perform better when operating on an expansive set of telemetry channels extracted from the different sub-systems of an SoC. Next, we present the necessary background and definitions based on which we construct the theorems that back our hypotheses.

4.2 Background and Definitions

Manifold. Intuitively, a manifold is a topological space that is locally Euclidean. A topological space is a set of points, with each point having its own set of neighborhoods [59]. We have a corresponding set of features for each of the GLOBL channels and the HPC groups. Using these feature sets, we can visualize an APK sample as a representative point in a higher dimensional vector space. We construct a manifold composed of a set of these representative points for each of the two classes, i.e., benign and malware. Finally, we end up.
A Stochastic Differential Equation (SDE). A SDE is a differential equation that contains terms which are stochastic processes, and is typically of the form

\[ dX_t = \mu(X_t, t) \, dt + \sigma(X_t, t) \, dB_t \]  

(1)

where \( B \) denotes the standard Brownian motion and \( X_t \) (a stochastic process) is the solution to Equation 1. The increments of \( B \) are independent and normally distributed. Hence, in a small time interval \( \delta \), the change in the stochastic process \( \Delta X_t \sim \mathcal{N}(\mu(X_t, t) \, \delta, \sigma(X_t, t)^2 \, \delta) \). The drift coefficient (\( \mu \)) determines the expectation \( \mathbb{E}(X_t) \) and the diffusion coefficient (\( \sigma \)) determines the Variance \( \mathbb{E}((X_t - \mathbb{E}(X_t))^2) \).

4.3 Theorems

Using the formal theory of linear separability of the object manifolds [33], we perform an analytical study that supports our hypotheses presented in Section 4.1. Collectively, the theorems establish that XMD’s superior performance stems from a higher solution volume. We present an intuitive take on the theorems in this section and refer the reader to Appendix A for a mathematically rigorous proof.

**Lemma 1:** The solution volume for a model trained with stochastic inputs (\( \Psi \)) is greater than the solution volume for a model trained with deterministic inputs (\( \Psi' \)), i.e., \( \mathbb{E}[\Psi] \geq \mathbb{E}[\Psi'] \).

Specifically, we model the stochasticity in the telemetry channel by adding Gaussian Noise in the solution volume, i.e., we add a diffusion term in the SDE, which, when solved using an Ito Integral, gives us the solution volume which is a stochastic random variable. Finally, we show that the solution volume calculated by taking the expectation over the stochastic process is lower bounded by the solution volume without stochasticity, resulting in better generalizability for the former. *This Lemma supports Hypothesis 1.1.*

**Theorem 1:** Stochasticity in the input training data improves concept drift robustness of the model.

To prove this, we model concept drift as a change in the drift term of the stochastic random variable representing the input feature space. Next, we show that with concept drift, the volume of the input space increases, which results in a decrease in the corresponding solution volume. Since the solution volume in the case of stochasticity is more than without stochasticity (Lemma 1), this leads to better concept drift robustness. *This theorem supports Hypothesis 1.2.*

**Theorem 2:** Let \( \Psi_i \) be the solution volume corresponding to the classification task of the benign and malware applications using the i-th telemetry channel in the N-dimensional vector space, where the i-th basis corresponds to the i-th telemetry channel \( \forall i \in [1, N] \), and N is the total number of telemetry channels. We show that the solution volume arising from the union of different \( \Psi_i \)'s, i.e. \( \bigcup_i \Psi_i \), is greater than the individual \( \Psi_i \)’s considered independently.

To prove this, we assume an N-dimensional vector space, with one basis for each of the N telemetry channels, and each \( \Psi_i \) is an orthogonal projection of the union of solution volumes \( \bigcup_i \Psi_i \) on the i-th basis. Next, we show that the solution volume arising from \( \bigcup_i \Psi_i \) is lower bounded by the maximum solution volume \( \Psi_{\text{max}} \), where \( \Psi_{\text{max}} = \max_i \Psi_i \).
A higher solution volume results in better classification performance, hence, supports Hypothesis 2.

### 4.4 Approach for Experimental Validation

To validate the hypotheses and theorems presented in Section 4.1 and 4.3 respectively, we design a robust data collection framework that collects the two classes of telemetry (HPC and GLOBL) simultaneously, and collectively captures a workload’s impact on different sub-systems of the SoC like CPU, GPU, memory, buses, and network (Section 5). The framework incorporates measures to reduce the datasets’ bias and prevent over-optimistic results. For example, we devise a Logcat-based activation checker that filters out the runs in which the application doesn’t have sufficient runtime. For both the HPC data logs and the GLOBL data logs, the data collection process produces multi-variate time series data. The feature engineering choices are the same as the ones used in prior works for DVFS [31, 32] and HPC [39, 76], and are summarized in Figure 3.(a) and 3.(b). For completeness, we briefly elaborate on these techniques in Appendix C.2.

**Validating the Hypotheses and Theorems.** We collect three different kinds of datasets using which we establish the concept drift robustness arising from the stochasticity in the hardware telemetry, i.e., GLOBL channels, validating Theorem 1 (Section 6.1 and 6.2). Next, using the fusion-based approach summarized in Figure 3.(c), we demonstrate the superior classification performance of the fusion-based model used in XMD that incorporates the expansive telemetry, validating Theorem 2 (Section 6.3).

### 5 Dataset Collection Framework

#### 5.1 Bare-metal analysis environment

While a Virtual Machine (VM) based sandbox provides benefits like isolation guarantees and checkpointing schemes, prior work has shown that using a VM-based sandbox introduces errors into the HPC measurements [76]. Further, malicious applications do not execute their payload upon detection of VM, aggravating the problem [27, 30, 50, 51]. To perform an automated large-scale collection of hardware-level telemetry, we built a host-client-based bare-metal sandbox environment as shown in Figure 6.

The client is a Google Pixel 3 mobile device (running Android OS) on which the benign and malicious samples are executed (Table 5), and the host is a Linux OS-based PC that orchestrates the data collection. We leveraged Android Debug Bridge (adb) [1] for transferring data and sending commands between the host and the client. The network packets from the client were routed through the host using Gnirehtet [9], providing unfiltered Internet access to the Client, which is crucial for the malware to perform essential functionalities like communicating with their command-and-control (C2) server. Since we were using a bare-metal analysis environment, we developed a custom checkpointing scheme to prevent contamination. Details are elaborated in Appendix B.2.

**Environment assumptions.** During data collection, multiple system workloads were executing in the background and were sending and receiving data packets on the network. Since the experimental Android device has a multi-core CPU, the processes belonging to the foreground application were context switching in and out of the cores and migrating to different cores throughout data collection. Overall, we ensure that the data collection environment is similar to the environments observed in real-world scenarios. Therefore, we believe this work’s observations will apply to a significant portion of mobile devices.

#### 5.2 Selection of Hardware Signals

**Choice of HPC channels.** The HPC events are stated in Table 2. We performed a comprehensive literature survey to identify the performance counter events used in prior work that resulted in the best detection performance. We consider an additional HPC event, not used in prior work, called raw-crypto-spec, which tracks the number of cryptographic operations. Cryptographic operations are used when performing SSL or TLS handshake and for deobfuscating malware payloads [15], and can potentially capture essential malware functionalities (see Appendix C.4). The collection of HPC events is limited by the number of HPC available on the device. On the Snapdragon chipset, there are four available HPCs. However, one HPC is repurposed for monitoring memory latency, so we can collect at most three events simultaneously. Therefore, all the HPC events are divided into groups 1-4. Each HPC group has three events that are collected simultaneously in a single iteration.

**Choice of DVFS channels.** Prior works have primarily focused on the impact of running a workload on the CPU by monitoring the DVFS states of the CPU controller [31, 32]; we expand on this notion by considering an expansive set of DVFS channels that cover both the CPU and the non-CPU devices like GPU, memory, buses, and caches. Channel-1 to Channel-11 in Table 3 elaborates on the selected DVFS channels, their corresponding locations in the Linux device tree, and the corresponding device’s signature they capture.

**Choice of SYSFS channels.** To capture the low-level impact of benign or malicious workload on Network devices, we recorded the number of bytes transmitted and received by the device. These channels are Channel-12 and Channel-13 in the Table 3. Since switching transistors consume energy, there is a correlation between power consumption and the type of workload. Prior works have demonstrated the efficacy of power side channels to detect malicious workloads on x86, IoT, and embedded systems [29, 34, 41, 43]. We collected telemetry from the voltage_now(Channel-14) and current_now(Channel-15) sysfs nodes. While the channels...
and events described earlier capture the impact of workload on a specific subsystem of the SoC, the power channels present a global telemetry channel capturing the impact on the entire SoC. Overall, the combined information from the HPC, DVFS, and SYSFS channels present a comprehensive low-level behavior over all the sub-modules of an SoC like the CPU, caches, GPU, memory, buses, and network.

5.3 Malware and Benign programs used

In this paper, we use a broad definition of malware, i.e., any application that has been flagged as malicious by at least one AV on VirusTotal [56]. While querying the VirusTotal service, we observed inconsistencies within the labels of the malware families, e.g., multiple samples were labeled “a variant of Android/Packed.Jiagu.<some_variant> potentially unsafe,” wherein Jiagu is a packer and not a family name. Such labeling inconsistencies have been identified in literature [65], and prevent us from performing a malware family classification study. Therefore, in this work, we focus on benign vs. malware study where XMD is used as an early-stage detector.

STD-DATASET: The dataset consists of real-world Android benign and malware applications. In particular, we acquired an initial dataset of 1033 samples of malicious and 723 samples of benign Android applications. The samples were downloaded from AndroZoo and were collected in the period from Dec 2019 to June 2021 [26]. There are a total of 54 malware families in the malware dataset, as reported by the ESET-NOD32 AV engine from VirusTotal. The number of applications and the total number of iterations is summarized in Table 4 under STD-DATASET.

BENCH-DATASET: To establish the role of non-determinism in the performance of GLOBL and HPC telemetry, we use 21 android benchmark applications [58]. The benchmark applications comprise CPU-bound benchmarks like floating-point operations and memory-bound benchmarks that test cache and memory. We use the same malware logs used in the STD-DATASET and ensure that we have a balanced dataset. As shown in Section 6.1.1, the benchmarks introduce a bias in the dataset.
5.4 Data Collection Architecture

The data collection architecture consists of three main components: the interaction module, the data-collection module, and the orchestrator module. The interaction module interacts with the application while it executes (see Appendix B.3), the data-collection module collects the data logs in the background while the application is executing in the foreground (see Appendix B.4), and the orchestrator is responsible for synchronizing sub-tasks (Appendix B.5). Figure 6 summarizes the overall flow of profiling and collecting the hardware telemetry logs for a single iteration of data collection.

Method for Running Experiments. For each Android application, we perform eight independent iterations of data collection, where a different sequence of interactions was used for each iteration. For each iteration of data collection, we collected all the GLOBL channels, and one group of HPC events since we are limited by the number of HPC registers. Overall, each Android application has eight iterations of data logs for the GLOBL channels and two iterations of data logs for each HPC group. Considering our limited resources (time and hardware), we ran data collection for a duration of 40 seconds for each iteration.

Ensuring correct execution using Logcat: We designed a Logcat-based activation checker to ensure that the malware is executing in the foreground while the hardware-telemetry logs are collected in the background. In Android OS, logs from applications are collected in a series of circular buffers, which can be filtered and viewed using Logcat [2, 12, 22]. We used logcat from the adb shell to filter the logs using the process ID (PID) of the foreground application executing on the device. As shown in Figure 6, the logs contain the timestamp of the activity, the PID, and the description of the activity. For every iteration of data collection, we collected its corresponding logcat and used it to verify the execution of the foreground application.

Figure 5 shows the distribution of the runtimes of different iterations of the benign and malware applications, calculated using the Logcat logs. We can observe a wide variation in the runtimes of the foreground applications, therefore, for this work, the logcat-based filter rejected all the iterations in which the foreground application executed for less than 15 seconds. Table 4 summarizes the number of files post-filtering for the GLOBL channels and each HPC group. E.g., in the

Table 4: Taxonomy of the dataset

| Dataset | Application type | # apk | % apk executed | % apk post-logcat filter | GLOBL | HPC |
|---------|-----------------|-------|----------------|-------------------------|-------|-----|
| STD     | Malware         | 2143  | 75%            | 75%                     | 417   | 602 |
|         | Benign          | 181   | 35%            | 35%                     | 213   | 327 |
| CD      | Malware         | 181   | 15%            | 15%                     | 213   | 327 |

Table 5: Analysis environment

| Model   | Google Pixel 3 |
|---------|----------------|
| OS      | Android 9.0    |
| Chipset | Qcom Snapdragon 845 |

Figure 4: DS for BENCH-DATASET and STD-DATASET (Metric for Bias).

Figure 5: Runtime for different iterations of malware and benign applications (using Logcat)

Figure 6: Overview of the data-collection framework. (1) Host-Client based malware analysis sandbox, (2) start of data collection process, (3) interaction module and data-collection module running concurrently, (4) system partition integrity checking, (5) Logcat-based filter for verifying execution.
We study the bias introduced by benchmark applications used as building blocks to validate our theorems. Testing the individual classifiers for each of the different HPC-based base-classifiers (Details in Appendix C.1). In the following, we first show that the range of DS calculated for the samples in the dataset using the different GLOBL telemetry channels. We can observe more distinguishability of malware and benign samples. A higher DS implies more distinguishability of malware and benign samples.

We calculated the average DS for the samples present in the BENCH-DATASET and STD-DATASET. Figure 4 shows the range of DS calculated for the samples in the dataset using the different GLOBL telemetry channels. We can observe that the range of DS for the BENCH-DATASET is higher than STD-DATASET, signifying the increased distinguishability and bias arising from the use of benchmarks, considering both datasets use the same set of malware samples.

### 6.1.2 Stochasticity in HPC vs. GLOBL channels

Prior works on HMD have reported their best results using Random Forest (RF) classifiers, hence we only consider the RF classifiers for creating base-classifiers for each of the GLOBL channels and the HPC groups. In the following, we demonstrate the role of stochasticity in the profiling power of the GLOBL and HPC channels, and its corresponding impact on the classification performance for the two different datasets: BENCH-DATASET and STD-DATASET. Figure 7 presents the classification performance of the different HPC-based classifiers and the GLOBL-based classifiers on the two datasets. We provide a brief discussion highlighting the key results and takeaways from these experiments.

**Performance on the BENCH-DATASET.** We observe high F1-scores across the GLOBL channels and the different HPC datasets.
groups. The workload behavior of benchmarks is significantly different from the malicious workloads of the dataset, introducing a strong bias in the dataset. The bias is supported by the F1-score of the different HPC groups, which can perfectly classify the benchmark applications from the malicious applications. HPCs only profile the benchmark applications, and do not have background noise from the other processes impacting their signatures. The GLOBL channels, on the other hand, are susceptible to the background noise arising from the system workloads running in the background. Therefore, their F1-scores are not as high as the different HPC-groups.

**Takeaway-1**

Thread-level profiling performed by HPCs can classify benchmarks vs. malware more accurately as compared to the GLOBL channels whose performance is susceptible to stochasticity arising from background workloads.

**Performance on the STD-DATASET.** Using real-world benign applications results in a more realistic scenario than using benchmark applications, making the classification task difficult. The F1 scores for the HPC groups and CPU-telemetry channels (GLOBL channel-2 and 3) are lower than what has been reported in prior works that have used benchmark applications labeled as benign. Compared to the BENCH-DATASET, the HPC-groups do not have the performance advantage over the GLOBL channels, signifying that CPU-telemetry is not enough to separate malicious applications from real-world benign applications, despite the accurate profiling power resulting from the thread-level profiling.

The GLOBL channels-12 and 13 that capture the number of transmitted and received bytes offer significantly higher F1-score compared to the other channels and HPC groups, demonstrating the role of communication with the C2 server in differentiating a malicious workload from a benign workload. Despite the higher F1 score, relying solely on low-level telemetry from one subsystem (e.g., network) creates a single mode of failure that can be easy to bypass [38]. We can observe that the different channels have different failure modes when identifying malicious workloads (see Appendix C.3).

There is a wide variation in the F1 scores (0.63-0.89) of ML models using the same learning algorithm for the different telemetry channels. This indicates that the solution volume \( \mathcal{V} \) of different telemetry channels are distinct, potentially arising due to the different sources of information captured by the telemetry channels.

**Takeaway-2**

Telemetry channels from different subsystems in an SoC have different classification scores, i.e., the solution volume \( \mathcal{V} \) of different telemetry channels are distinct.

### 6.2 Theorem 1: Stochasticity vs. CD?

**CD-DATASET:** To assess the performance of the HMDs against CD scenarios, we create a CD test dataset in which the samples were collected from July 2021 to Dec 2021, non-overlapping from the time interval in which the STD-DATASET was collected. This dataset is used as a test dataset to test the performance of the models trained on the STD-DATASET. The number of applications and the total number of iterations is summarized in Table 4 under CD-DATASET.

**Results.** As shown in Figure 7, the GLOBL and HPC-based base-classifiers suffered degradation in accuracy when evaluated on the CD-DATASET. The average degradation for the GLOBL channel-based base-classifiers was 5.6%, while it was 24.8% for the HPC-groups-based classifiers. While the degradation in accuracy is insignificant for certain GLOBL channels, it is high for the different HPC groups. The non-determinism in the GLOBL channels (Takeaway-1) acts as a
regularizer and forces the GLOBL channel-based classifiers to have better generalization, evident by their comparatively lesser drop in accuracy compared to the HPC channels.

**Takeaway-3**
The stochasticity in the GLOBL channels results in better concept drift robustness as compared to the HPC groups, and therefore, validates Theorem-1 presented in Section 4.

### 6.3 Expansive telemetry vs. performance

[Figure 8: DS score with respect to number of channels]

In the previous section, we observed that each class of telemetry has its strengths and weaknesses. HPCs can accurately profile the CPU and identify malicious threads but offer poor generalization and concept drift robustness. On the other hand, the GLOBL channels capture a global impact of running a workload on the SoC and offer concept drift robustness but cannot identify the malicious threads. This motivates us to develop a fusion-based approach, called XMD, that complements the thread-level profiling provided by the HPCs with the global profiling provided by the GLOBL channels.

#### 6.3.1 Validating Theorem-2: Is fusion-based model the right approach?

**Theorem-2** in Section 4 guides the design of a fusion-based model where using multiple telemetry channels results in better solution volume than considering a single telemetry source, e.g., the CPU-telemetry-based classifiers. We evaluate the impact of incorporating more telemetry channels on the distinguishability of the benign and malware applications using DS derived from the t-test presented in Section 6.1.1.

For each pair of benign and malware applications, we incorporate more telemetry channels into the feature vector. This is followed by Principal Component Analysis to reduce the augmented feature size to the feature size of a single channel, eliminating the bias arising from increased feature size. We then calculate the DS. As shown in Figure 8, we see an increasing trend in the DS upon increasing the number of telemetry channels, indicating the increased distinguishability of malware and benign samples. An increased distinguishability potentially indicates that the solution volume arising from \((\bigcup_i \Psi_i)\) is higher than the solution volumes of the telemetry channels \((\Psi_i)\) when considered independently. Next, we explore the use of late-stage fusion-based detection to realize the potential performance improvements of incorporating multiple channels.

#### 6.3.2 Approach: Late-stage fusion

As shown in Figure 3.(c), we consider two decision fusion approaches. In the first approach, called ENSEMBLE, we take a majority vote of decisions of the individual base-classifiers, each of them trained on a different telemetry channel. The second approach is a stacked generalization approach where we fuse the decisions of the base-classifiers using a second-stage model, which is a logistic regression model in our case. Logistic Regression was selected as the second-stage model as the learned parameters of the model will provide insight into the importance of each of the GLOBL channels and HPC groups when arriving at the final decision (see Appendix C.4).

**Fusing the decisions of the GLOBL channels.** Table 6 shows the F1 scores from fusion of the different base-classifiers of the GLOBL and DVFS channels. We can observe the following trends from these results: First, for each dataset, the F1 score obtained after fusing the GLOBL channels has a better F1 score compared to the sub-group of DVFS. Therefore, considering the impact of running a workload on all the sub-devices of the SoC is essential. Second, F1-score from fusion is greater than the F1-scores from each of the individual channels which agrees with the statistical analysis in Section 6.3.1 and validates Theorem-2 in Section 4, wherein fusion of distinct solution volumes of the individual telemetry channels (Takeaway-2) results in a higher solution volume. An interesting side-effect of an increased solution volume is the F1 score of 0.98 achieved by the fused model of DVFS channels. Fusing the decisions of different telemetry channels offsets the degradation in profiling accuracy arising from the stochasticity. Moreover, with an F1-score of 0.84 on the CD-DATASET, the enhanced concept drift robustness arising from the stochasticity is preserved.

**Takeaway-4**
The performance of the fused model incorporating all the GLOBL telemetry channels is greater than the performance of each channel independently, validating Theorem-2.

**XMD: Fusing the decisions of the GLOBL channels and HPC.** Table 7 shows the F1 score of the individual HPC groups when they were standalone and when used in conjunction with the DVFS and the GLOBL channels for
Table 6: F1-score of the fused GLOBL channels

| Fused channel | Participating Channels | BENCH Dataset | STD Dataset | CD Dataset |
|---------------|------------------------|---------------|-------------|------------|
| DVFS          | 1-11                   | 0.98          | 0.90        | 0.82       |
| GLOBL         | 1-15                   | 0.99          | 0.92        | 0.84       |

Table 7: F1-scores of different HPC-groups before and after fusion with the DVFS and GLOBL channels.

| HPC          | Group  | Dataset  | Standalone | With DVFS (Ensemble) | With DVFS (SG) | With GLOBL (Ensemble) | With GLOBL (SG) |
|--------------|--------|----------|------------|----------------------|----------------|-----------------------|-----------------|
| group-1      | STD    | 0.66     | 0.89       | 0.88                 | 0.93           | 0.90                  |                 |
|              | CD      | 0.55     | 0.86       | 0.85                 | 0.87           | 0.83                  |                 |
| group-2      | STD    | 0.71     | 0.89       | 0.84                 | 0.89           | 0.92                  |                 |
|              | CD      | 0.56     | 0.83       | 0.81                 | 0.85           | 0.87                  |                 |
| group-3      | STD    | 0.76     | 0.90       | 0.91                 | 0.93           | 0.93                  |                 |
|              | CD      | 0.54     | 0.86       | 0.81                 | 0.87           | 0.83                  |                 |
| group-4      | STD    | 0.67     | 0.91       | 0.90                 | 0.92           | 0.93                  |                 |
|              | CD      | 0.46     | 0.83       | 0.81                 | 0.87           | 0.90                  |                 |

Table 8: Detection rates of XMD and Antivirus Scanners on VirusTotal

| Ensemble | XMD | AV1 | AV2 | AV3 | AV4 | AV5 | AV6 | AV7 | AV8 | AV9 | AV10 |
|----------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|
| group-1  | 84.13 (FPR = 3.70) | 83.87 | 82.76 | 66.12 | 59.64 | 53.44 | 53.22 | 35.48 | 32.25 | 30.50 | 27.42 |
| group-2  | 87.83 (FPR = 6.89) | 80.22 | 81.93 | 53.76 | 60.60 | 57.47 | 52.17 | 35.10 | 38.46 | 23.19 | 23.65 |
| group-3  | 86.54 (FPR = 2.90) | 80.00 | 79.74 | 68.23 | 48.05 | 65.00 | 42.86 | 43.53 | 40.00 | 30.48 | 34.11 |
| group-4  | 85.71 (FPR = 5.23) | 83.56 | 76.81 | 69.86 | 54.54 | 48.48 | 64.28 | 41.09 | 34.78 | 23.88 | 29.17 |

both the techniques of late-stage fusion, i.e., ensemble and stacked generalization (SG). We can observe the following trends from this table: First, the predictive performance of the GLOBL channels, when fused with the HPCs, is higher than the predictive performance of the fused GLOBL models (in Table 6) or the HPC-based base-classifiers. Second, for CD-DATASET, where the performance of the HPC-based base-classifiers suffers significant degradation; upon fusing their decisions with the GLOBL channels, it restores the accuracy to a significant extent. For example, the F1-score of HPC-group-4 dropped to 0.46 when tested on the CD-DATASET; however, it increased to 0.90 when fused with the GLOBL channels. While the performance degradation of the base-classifier when testing on CD-DATASET was 31.3%, the degradation of the HPC-GLOBL fused model was merely 3.2%. The receiver operating characteristic (ROC) curve (on the STD-DATASET) for the Ensemble-fusion models is presented in Figure 9.

Takeaway-5

The HPC-GLOBL fused model inherits the concept-drift robustness from the GLOBL channels and exploits the thread-level profiling power of the HPCs, resulting in a robust malware detector with better performance.

6.4 Comparison with software-based detectors

We compare the performance of the HPC-GLOBL fused models against the software-based AV engines available on VT. For the samples present in our test dataset (in STD-DATASET), we queried VT to get decisions from the AV engines. Not all of the AV engines produced a decision on the samples present in our test dataset, as they may not be specialized in Android malware. Therefore, we selected the top 10 AV engines with the best detection performance on the test samples and had produced decisions for at least 85% of the samples present in our dataset. These AV scanners are ESET-NOD32, Ikarus, K7GW, Microsoft, CAT-QuickHeal, Fortinet, Avira, Cyren, Kaspersky, Lionic, Trustlook, and ZoneAlarm.

Table 8 shows the detection performance of the different fused GLOBL-HPC-groups. For example, while the best AV performance for the test dataset of HPC group-3 is around 80%, the fused model provides a detection rate of 86.54 with an FPR of 2.9%. Such observations clearly show that the HMD can play a significant role in enhancing endpoint detection performance in a system of collaborative detectors.

7 Discussion

Theoretical basis of XMD’s superior performance. XMD performs classification in a higher solution volume than prior HMDs. Theorem-1 indicates that the solution volume of GLOBL channels is higher than the HPCs, resulting in better performance.
concept drift robustness. Theorem-2 guides the design of a fusion-based approach that results in a higher solution volume than the individual telemetry channels. XMD exploits both the theorems to provide increased concept drift robustness compared to HPC-based HMDs (Theorem-1) and better classification performance than prior CPU-telemetry-based HMDs (Theorem-2).

**Non-determinism in Hardware Telemetry.** Recent works have identified failure scenarios in HMDs arising from the non-determinism in the HPC-based telemetry [38, 76]. Non-determinism in HPCs arises due to measurement errors like overcounting, background OS operations, and multiple applications running on the system. On the other hand, the non-determinism in the GLOBL channels arises from global profiling compared to the thread-level profiling performed by the HPCs. This makes the GLOBL channels directly susceptible to the noise arising from background applications, and therefore, they have much higher stochasticity than HPCs (Takeaway-1). Hence, we believe our assumption of non-determinism in GLOBL channels and determinism in HPC for the theorems is valid and is supported by Takeaway-3.

Compared to prior HPC-based defenses, XMD does not eschew the non-determinism in the hardware telemetry. XMD exploits the non-determinism in the hardware telemetry to provide superior concept drift robustness compared to the prior HPC-based approaches. Moreover, XMD relies on an expansive set of telemetry channels that is not restricted to the CPU core of the SoC, potentially making it robust to such proof of concept attacks that are aimed at skewing the CPU telemetry measurements [38, 76]. Finally, we concur with the opinions of Kazdagli et al., i.e., the goal of HMD is not to provide 100% true positives and 0% false positives but to provide an extra layer of protection and complement other software-based detectors in a collaborative defense system.

**Extension to Desktop-class of devices.** While we have considered a mobile-device environment in this work, a similar approach can be extended to desktop-class of devices. Profiling frameworks provided by hardware vendors can monitor the traffic activity to storage and network devices, memory access patterns like stalls on loads from DRAM and caches, and the GPU and CPU usage of the profiled applications [8]. The drivers from such tools can be repurposed to potentially enhance the detection performance of the deployed HMDs [6].

**Limitations and Future Works.** Our work has a few limitations. First, while we have taken measures to make our data-collection environment realistic for mobile devices, there are corner cases that are not covered by the collection environment. Our Interaction Module (Appendix B.3) uses the Monkey tool, which is a stateless interaction, different from real user inputs. A better approach for simulating UI interaction is using a state-based interaction model (e.g., Droidbot [57]). However, in our framework, Droidbot crashed unpredictably and required human intervention. Future work can look into better approaches for simulating human interaction for large-scale telemetry collection. The analysis performed in this paper is based on a single Google Pixel 3 mobile device and is therefore restricted in scope with respect to SoC chipsets and mobile devices. However, the fundamental theorems and the corresponding empirical observations presented in this work do not have architectural or platform dependence and should potentially extend to other mobile devices. We defer the empirical verification of the proposed theorems on different mobile device platforms as future work.

Second, while our data-collection environment is realistic for mobile devices, the same assumptions may not hold for desktop-class machines, where the number of workloads running concurrently is much higher than the mobile-device environment. In Equation 6 (Appendix A.2) representing the solution volume with stochastic telemetry, if the diffusion term (arising from the stochasticity) overpowers the drift term, then classification with a linear hyperplane may not be feasible. In such a case, we need per-process tracking capability for the expansive hardware telemetry, like those provided by commercial profilers [8]. We defer the extension of XMD to desktop-class of machines for future work.

Finally, we have used a late-stage fusion approach to incorporate the power of multiple telemetry channels. However, an intermediate fusion-based approach can potentially result in better detection performance at the cost of increased complexity, e.g., an approach that exploits the interaction between the different telemetry channels. We leave the exploration of such novel ML agents as future work.

### 8 Conclusion

In this paper, we propose XMD, a malware detector that exploits the expansive set of hardware telemetries from the different sub-systems of an SoC used in a mobile device. XMD exploits two key innovations grounded in Theorems that we have developed using the Replica Theory of object manifolds: superior classification performance from expansive telemetry and concept drift robustness from stochasticity in hardware telemetry. To evaluate XMD, we create three different datasets using a robust data collection framework. Our findings suggest that XMD outperforms the current HPC-based detectors and the commercial AV software on VirusTotal with acceptable false-positive rates. Therefore, XMD can complement other software-based detection approaches in a collaborative defense system.

**References**

[1] Android Debug Bridge. https://developer.android.com/studio/command-line/adb.

[2] The Android Logging Service – A dangerous feature for User Privacy? https://blogs.uni-paderborn.de/sse/2013/05/17/privacy-threatened-by-logging/.

[3] Android Monkey. https://developer.android.com/studio/test/other-testing-tools/monkey.
detection of IoT malware using power side channels. In Proceedings of the 15th ACM Asia Conference on Computer and Communications Security, ASIA CCS '20, page 33–46, New York, NY, USA, 2020. Association for Computing Machinery.

[42] Elizabeth Gardner. The space of interactions in neural network models. Journal of physics A: Mathematical and general, 21(1):257, 1988.

[43] Jarilynn Hernandez Jimenez and Katerina Goseva-Popstojanova. Malware detection using power consumption and network traffic data. In 2019 2nd International Conference on Data Intelligence and Security (ICDIS), pages 53–59, 2019.

[44] Roberto Jordaney, Kumar Sharad, Santanu K. Dash, Zhi Wang, Davide Papini, Ilia Noureddinov, and Lorenzo Cavallaro. Transcend: Detecting concept drift in malware classification models. In 26th USENIX Security Symposium (USENIX Security 17), pages 625–642, Vancouver, BC, August 2017. USENIX Association.

[45] Sai Praveen Kadiyala, Pranav Jadhav, Siew-Kei Lam, and Thambipillai Srikantian. Hardware performance counter-based fine-grained malware detection. ACM Trans. Embed. Comput. Syst., 2020.

[46] George Karantzas and Constantinos Patsakis. An empirical assessment of Endpoint Detection and Response Systems against Advanced Persistent Threats Attacker Vectors. Journal of Cybersecurity and Privacy, 2021.

[47] Mikhail Kazdagli, Vijay Janapa Reddi, and Mohit Tiwari. Quantifying the space of interactions in neural network models. Proceedings of the ACM/IEEE International Symposium on Security and Privacy (SP), pages 1466–1482, 2020.

[48] Amin Kharaz, Sajjad Arshad, Collin Mulliner, William Robertson, and Roberto Jordaney. Discovering the space of interactions in neural network models. In 26th USENIX Security Symposium (USENIX Security 17), pages 625–642, Vancouver, BC, August 2017. USENIX Association.

[49] Jie Xue, Yuan Li, and Ravi Janardan. On the expected diameter, width, and thickness of Euclidean unit ball graphs. Computational Geometry, 44(8):583–590, 2011.

[50] Harshit Kumar, Nikhil Chawla, and Saibal Mukhopadhyay. BiasP: A systematic study of malicious code grafting. IEEE Transactions on Information Forensics and Security, 12(6), 2017.

[51] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. Droidbot: a lightweight ui-guided test input generator for android. In 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), pages 23–26, 2017.

[52] Roy Longbottom. Updated android benchmarks for 32 bit and 64 bit cpus from arm and intel contents, 03 2018.

[53] Yunqian Ma and Yun Fu. Manifold learning theory and applications, volume 434. CRC press Boca Raton, 2012.

[54] Jonathan Mamou, Hang Le, Miguel Del Rio, Cory Stephenson, Hanlin Tang, Yoon Kim, and SueYeon Chung. Emergence of separable manifolds in deep language representations. arXiv preprint arXiv:2006.01195, 2020.

[55] Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. Plundervolt: Software-based fault injection attacks against intel sgx. In 2020 IEEE Symposium on Security and Privacy (SP), pages 1466–1482, 2020.

[56] Meltem Ozsoy, Caleb Donovick, Ivor Gorelik, Nael Abu-Ghazaleh, and Dimitry Pomonarev. Malware-aware processors: A framework for efficient online malware detection. In 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), pages 651–661, 2015.

[57] Meltem Ozsoy, Caleb Donovick, Lukov Gorelik, Nael Abu-Ghazaleh, and Dimitry Pomonarev. Malware-aware processors: A framework for efficient online malware detection. In 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), pages 651–661, 2015.

[58] Zhongmin Qian. A gradient estimate on a manifold with convex boundary. Proceedings of the Royal Society of Edinburgh Section A: Mathematics, 127(1):171–179, 1997.

[59] Mohammed Rashed and Guillermo Suarez-Tangil. An analysis of android malware classification services. Sensors, 21(16), 2021.

[60] Irwin Reyes, Primal Wijesekera, Joel Reardon, Amit Elazari Bar On, Abbas Razaghpanah, Narseo Vallina-Rodriguez, and Serge Engelman. “won’t somebody think of the children?” examining COPPA compliance at scale. Proceedings on Privacy Enhancing Technologies, 2018:63 – 83, 2018.

[61] Salah Rifai, Xavier Glorot, Yoshua Bengio, and Pascal Vincent. Adding noise to the input of a model trained with a regularized objective, 2011.

[62] Tobias Schneider and Amir Moradi. Leakage assessment methodology. Cryptographic Hardware and Embedded Systems, 2015.

[63] Cory Stephenson, Jenelle Feather, Suchismita Padhy, Oguz Elibol, Hanlin Tang, Josh McDermott, and SueYeon Chung. Untangling in invariant speech recognition. Advances in neural information processing systems, 32, 2019.

[64] Cory Stephenson, Suchismita Padhy, Abhinav Ganesh, Yue Hui, Hanlin Tang, and SueYeon Chung. On the geometry of generalization and memorization in deep neural networks. arXiv preprint arXiv:2105.14602, 2021.

[65] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfi. CLKSCREW: Exposing the perils of Security-Oblivious energy management. In 26th USENIX Security Symposium (USENIX Security 17), August 2017.

[66] Sanjeev Tannirkulam Chandrasekaran, Abraham Peedikayil Kuruvila, Kanad Basu, and Arindam Sanyal. Real-time hardware-based malware and micro-architectural attack detection utilizing cmos reservoir computing. IEEE Transactions on Circuits and Systems II: Express Briefs, 69(2):349–353, 2022.

[67] R. Vinayakumar, Mamoun Alazab, K. P. Soman, Prabaharan Poor-nachandran, and Sitalakshmi Venkatraman. Robust intelligent malware detection using deep learning. IEEE Access, 7:46717–46738, 2019.

[68] Jie Xue, Yuan Li, and Ravi Janardan. On the expected diameter, width, and complexity of a stochastic convex hull. Computational Geometry, 82:16–31, 2019.
We present the rigorous mathematical proofs in this section, which the space is divided by a line. A convex polytope is an
intersection of a finite number of half-spaces.

Mathematically, the convex hull is given as:

\[
\mathcal{CH}(F^p) = \left\{ \mathbf{x}^\mu(S) \mid \bar{S} \in \mathcal{CH}(S) \right\},
\]

where

\[
\mathcal{CH}(S) = \left\{ \sum_{i=1}^{D+1} \alpha_i \mathbf{S}_i \mid \mathbf{S}_i \in S, \alpha_i \geq 0, \sum_{i=1}^{D+1} \alpha_i = 1 \right\}
\]

Solution Volume. Following Gardner’s replica framework [42], the volume \( \mathcal{V} \) of the solution space is defined as

\[
\mathcal{V} = \int d^N \mathbf{w} \delta \left( \| \mathbf{w} \|^2 - N \right) \prod_{\mu \in F^p} \Theta \left( \mathbf{y}^\mu \cdot \mathbf{x}^\mu - \kappa \right)
\]

where \( \Theta(\cdot) \) is the Heaviside function to enforce the margin constraints in the linear separation constraint \( \mathbf{y}^\mu \cdot \mathbf{x}^\mu \geq \kappa \), along with the delta function to ensure \( \| \mathbf{w} \|^2 = N \).

A Mathematical proofs

We present the rigorous mathematical proofs in this section, complementing the intuitive proofs and definitions presented in Section 4.

A.1 Additional Definitions and Background

Manifold. A point on the manifold consists of the input space \( x^d \in F^p \) given as \( x^d(S) = \sum_{i=1}^{D+1} S_i u^\mu_i \) where \( u^\mu_i \) are a set of orthonormal bases of the \((D+1)\) dimensional linear subspace containing \( F^p \), the \( D+1 \) components \( S_i \) represent the coordinates of the manifold point within this subspace and are constrained to be in the set \( S \subseteq S \). \( S \) denotes the shape of the manifolds and encapsulates the affine constraint.

Convex Hull and Polytopes: The convex hull of a set of points \( S \) is the intersection of all half-spaces that contain \( S \). A half-space is either of the two parts into which a hyperplane divides an affine space. For example, in a two-dimensional Euclidean space, a half-space is either of the two parts into which the space is divided by a line. A convex polytope is an intersection of a finite number of half-spaces.

Mathematically, the convex hull is given as: \( \mathcal{CH}(F^p) = \left\{ x^\mu(S) \mid \bar{S} \in \mathcal{CH}(S) \right\} \), where

\[
\mathcal{CH}(S) = \left\{ \sum_{i=1}^{D+1} \alpha_i \mathbf{S}_i \mid \mathbf{S}_i \in S, \alpha_i \geq 0, \sum_{i=1}^{D+1} \alpha_i = 1 \right\}
\]

A.2 Theorems and Proofs

Lemma 1: The solution volume for a model trained with stochastic inputs (\( \mathcal{V} \)) is greater than the solution volume for a model trained with deterministic inputs (\( \mathcal{V} \)) i.e., \( \mathbb{E}[\mathcal{V}] \geq \mathcal{V} \).

Short Proof: To model the stochasticity in the inputs, we add a stochastic process, which is given by the Gaussian Noise, to the input data \( x \) as a diffusion term. To account for the Gaussian Noise in the input space, we consider the induced stochasticity in the manifolds of the convex polytopes that are obtained from the convex hull of the input data. Hence, the solution volume is also a stochastic random variable \( \mathcal{V} \) and is given by

\[
\mathcal{V} = \int d^N \mathbf{w} \delta \left( \| \mathbf{w} \|^2 - N \right) \prod_{\mu \in F^p} \Theta \left( \mathbf{y}^\mu \cdot \mathbf{x}^\mu - \kappa \right)
\]

where we model the stochasticity by adding a diffusion term \( \int G(w_\alpha, \alpha) dB_\alpha \). The diffusion term consists of Brownian motion \( B_\alpha \) and its coefficient \( G \) is parameterized by unknown variables \( w_\alpha, \alpha \). The diffusion term is an Itô integral and it follows Gaussian distribution. If we set \( G(w_\alpha, \alpha) = \sigma \), the result of integration is \( B_{\alpha+1} - B_\alpha \sim \mathcal{N}(0, \alpha \sigma^2) \), which is consistent with adding Gaussian noise. Next, we show that \( \mathbb{E}[\mathcal{V}] \geq \mathcal{V} \).

For the calculation of the stochastic integral \( \int G(w_\alpha, \alpha) dB_\alpha \), we assume that \( G(w_\alpha, \alpha) \) is changed at discrete time points \( t_i (i \in [1, N]) \), where \( 0 < t_1 < \ldots < t_N < T \). We define the integral

\[
S = \int_0^T G(w_\alpha, \alpha) dB_\alpha
\]

as the Riemann sum

\[
S_N(w) = \lim_{N \to \infty} \sum_{i=1}^N G(\alpha_{i-1}, w) (B_{\alpha_i} - B_{\alpha_{i-1}})
\]

Since the diffusion is on the manifold of the convex polytope \( \mathcal{V} \), \( G(w_\alpha, \alpha) \) is a submartingale, which is a process which increases on average [37, 64]. Hence, \( \mathbb{E}[S_N(w)] \geq 0 \Rightarrow \mathbb{E}[\mathcal{V}] \geq \mathcal{V} \). Therefore, the solution volume with stochastic inputs is lower bounded by the solution volume with deterministic inputs.

Lemma 2: Let us represent the input with drift as \( x' = x + \dot{x} \), where \( x \) is the input data with stochasticity and \( \dot{x} \) represents the drift. The modified convex polytope formed with drift can be written as \( \mathcal{CH}(x') = \mathcal{CH}(x + \dot{x}) \geq \mathcal{CH}(x) \).

Proof: We use the expected diameter of stochastic convex hulls to compare their volumes as discussed by Xue et al. [74]. The diameter of a convex polytope \( A \), denoted by \( \text{diam}(A) \), is defined as the maximum Euclidean distance between any two points in the polytope. Hence, the expected diameter of the stochastic manifold \( x' \) is given as

\[
\text{diam}_{x'} = \sum_{R \subseteq x'} \mathbb{P}(R) \cdot \text{diam}(\mathcal{CH}(R))
\]
where \( \Pr[R] \) denotes the probability that \( R \) occurs as a realization of \( X' \). Since we are considering a Gaussian noise as the diffusion term, without loss of generality, we can consider that the probability of realization remains constant as we dissect the space \( X' \) into \( X, \dot{X} \) respectively. Hence, using triangle inequality for any two points in the convex hulls of the two sets \( X, \dot{X} \), we get

\[
\text{diam}_X = \sum_{R \in X + \dot{X}} \Pr[R] \cdot \text{diam}(\mathcal{CH}(R))
\geq \sum_{S \subseteq X} \Pr[S] \cdot \text{diam}(\mathcal{CH}(S)) + \sum_{T \subseteq \dot{X}} \Pr[T] \cdot \text{diam}(\mathcal{CH}(T))
\geq \sum_{S \subseteq X} \Pr[S] \cdot \text{diam}(\mathcal{CH}(S))
= \text{diam}_X
\]

\[ \tag{8} \]

**Theorem 1:** Stochasticity in the input training data improves concept drift robustness of the model.

**Short Proof:** We consider the manifold obtained from the input space using its convex hull. If \( X \) is the initial manifold of the input space, then the input with the drift is represented as \( X' = X + \dot{X} \), where \( \dot{X} \) represents the drift in the manifold. Hence, the modified convex polytope formed with drift can be written as \( \mathcal{CH}(X') = \mathcal{CH}(X + \dot{X}) \geq \mathcal{CH}(X) \) (by Lemma 2). Now, we consider \( X', \dot{X} \) as the volume of the convex polytopes of the input with and without concept drift, respectively. Now, \( X' \geq X \) due to the added volume of the convex hull formed with the drifted datapoints. Let us denote the solution volumes with and without concept drift as \( \mathcal{V}', \mathcal{V} \) respectively and the solution volume with stochasticity as \( \mathcal{V}' \). \( \mathcal{V}' \) is proportional to the complement of the volume of the convex polytope of the input data \( X \) i.e., \( \mathcal{V}' \propto \frac{1}{X} \). Using Lemma-1, \( E[\mathcal{V}'] \geq \mathcal{V}' \Rightarrow \mathcal{V}' \geq \mathcal{V} \) i.e., solution volume for concept drift with stochasticity is greater than the solution volume without stochasticity. Thus, stochasticity improves the robustness to concept drift. 

**Lemma 3:**

\[ \mathcal{V}'[\mathcal{CH}(\bigcup_i \mathcal{V}'_i)] \geq \max \{ \mathcal{V}'_i \} \tag{9} \]

**Proof:** Let \( \mathcal{V}'_i \) be nonempty, convex sets. We show that \( x \in \mathcal{CH}(\bigcup_i \mathcal{V}'_i) \) if and only if there exist elements \( v_i \in \mathcal{V}'_i \) and \( \lambda_i \geq 0 \) with \( \sum_i \lambda_i = 1 \) such that \( x = \sum_i \lambda_i v_i \). This can be represented as

\[
\mathcal{CH}(\bigcup_i \mathcal{V}'_i) = \left\{ \sum_i \lambda_i v_i \mid \sum_i \lambda_i = 1, \lambda_i \geq 0, v_i \in \mathcal{V}'_i \right\}
\]

Now, let us consider the solution volume \( \mathcal{V}' \) as a measure defined on a vector space \( \Omega \). Then we show that \( \mathcal{V}'(A) \leq \mathcal{V}'(B) \) for all \( A \subseteq B \). Let \( A \subseteq B \), let \( C = A^c \cap B \). Then \( A \cap C = \emptyset \) and \( A \cup C = B \). Thus, \( \mathcal{V}'(A \cup C) = \mathcal{V}'(A) + \mathcal{V}'(C) = \mathcal{V}'(B) \). Therefore \( \mathcal{V}'(A) \leq \mathcal{V}'(B) \) for \( \mathcal{V}'(C) \geq 0 \) by non-negativity. Hence, the solution volume of a convex manifold \( X \), given as \( \mathcal{V}'(X) \), is a monotonic increasing measure. Therefore,

\[
[\mathcal{V}'(\mathcal{CH}(\bigcup_i \mathcal{V}'_i))] = [\mathcal{V}'(\mathcal{CH}(\mathcal{V}'_i \cup \mathcal{V}'_j))]
\geq [\mathcal{V}'(\mathcal{V}'_i \cup \mathcal{V}'_j)]
\geq [\mathcal{V}'(\mathcal{V}'_i)] = \max \{ (\mathcal{V}'_i) \} \forall i \tag{10}
\]

**Theorem 2:** Let \( \mathcal{V}'_i \) be the solution volume corresponding to the telemetry channel-i in the N-dimensional vector space where the i-th basis corresponds to the i-th telemetry channel \( \forall i \in \{1, N\} \). Let \( N \) be the total number of telemetry channels. Then, \( \mathcal{V}'(\mathcal{CH}(\bigcup \mathcal{V}'_i)) \geq \max \{ \mathcal{V}'_i \} \). 

**Short Proof:** We consider a convex polytope for each of the solution volume \( \mathcal{V}' \). Without loss of generality, we assume that each \( \mathcal{V}'_i \) is an orthogonal projection of the union of the solution volumes \( \bigcup \mathcal{V}'_i \), which we refer to as the universal convex polytope. The universal convex polytope is constructed by taking a convex hull over the union of its N orthogonal components \( \mathcal{V}'_i \). From Lemma 3, we get \( \mathcal{V}'(\mathcal{CH}(\bigcup_i \mathcal{V}'_i)) \geq \max \{ \mathcal{V}'_i \} \).

**B Data Collection Framework**

**B.1 Dataset bias in prior works**

*Benchmarks used as benign workloads.* Prior works on HMD primarily use benchmark applications for benign workloads \([24, 32, 39, 45, 54, 55, 63, 72]\). Few works use regular benign applications (e.g., from Play Store), however, they mix these applications with benchmark applications \([49, 76]\). Compared to regular benign applications, which require interaction with the device to explore the different threads of operation, running the benchmark application is straightforward which eases the process of large-scale data collection. These benchmark applications are synthetic workloads designed to test a specific functionality of the SoC and are not representative of real-world benign applications, introducing a bias in the dataset.

*Execution of the workload.* Prior "black-box" approaches of performing large-scale data collection for hardware telemetry do not verify whether the malware is executing \([32, 36, 39, 45, 54, 55, 63, 72]\). The raw hardware signatures do not provide interpretability, making the verification of malware execution more difficult, as compared to software-based behavioral signatures \([48]\). It has been shown that such "black-box" approaches toward collecting hardware-telemetry are ineffective and introduce inconsistencies in the dataset. For example, in the original work using HPC for malware detection \([39]\), 20%
of malware traces are shorter than 1 second, and 56% are <10s [47]. Kazdagli et al. identified this issue in their work and took steps to ensure the correct execution of synthetic malware payloads that they had devised [47]. However, for off-the-shelf malware, they did not elaborate on how they "checked the validity of performance counter readings". Further, prior DVFS works have not tackled this issue in malware detection approaches [32]. For such global signatures, it is important to ensure the successful execution of workloads of interest, because these channels capture signatures irrespective of the execution of the workload of interest. Finally, vital functionalities of malware like communication with the C2 server and subsequent payload activation often rely on network access. Prior works do not specify access to the internet in their data collection setup [32,36,45,54,55,63,72,76].

B.2 Checkpointing scheme

We devised a custom checkpointing scheme to restore the client to its clean state after data collection was performed on a particular sample. Since performing a factory reset on the phone requires manual intervention to set up the phone again, we used a custom recovery tool (TWRP [20]) to create an image of the userdata partition. We assumed that the applications under analysis can read/write to the userdata partition. Since the client was rooted (using Magisk [13]) and we ran applications with full permissions, we used a checksum-based scheme to determine whether the non-userdata partitions are altered or not, handling cases where malware modify system partitions to gain persistence [21, 23]. After the malicious application finishes its execution, we verify the integrity of the non-userdata partitions using the checksum. Upon detecting modifications to the system partitions, we flash all the partitions of the device, otherwise, we only flash the userdata partition. As we can conclude from Table 9, the checksum-based integrity checking saves a lot of data-collection time, considering the fact that we did not detect any malware in our dataset that modified the non-userdata partition.

Table 9: Checkpointing procedure timings

| Step                | Time (s) |
|---------------------|----------|
| Reboot              | 28       |
| Restore userdata partition | 195     |
| Restore all partitions | 435     |

B.3 Interaction Module

The activation of malicious payloads in the Android ecosystem is often contingent on events that require user interactions. This makes performing large-scale automated data collection for Android malware more difficult as compared to desktop-based malware [47]. Therefore, we devised an interaction module to simulate human interaction with the goal of triggering the malware payloads. The Interaction-module consists of two sub-modules: Monkey-based interaction and Broadcast-event-based interaction.

Monkey-based interaction. We use Android’s UI/Application Exerciser Monkey [3], a popular tool in Android SDK, to automate the interaction with applications by simulating user inputs. Monkey simulates random UI interactions by injecting a pseudorandom stream of simulated user input events into the app. Prior works have shown that Monkey matches or exceeds adult human coverage 61% of the time [66]. However, we note that Monkey provides sub-optimal code coverage considering its random nature, and is unable to react to visual UI elements like popup dialog boxes. Therefore, we added functionalities (like Broadcast Event-based interaction) to increase the code coverage. Furthermore, our Logcat based activation checker filters out the iterations in which the Interaction module fails to achieve sufficient activation ensuring the cleanliness of the dataset.

Broadcast-event-based interaction. Malware applications observe system-level broadcast events, an essential component of Inter-Process communication, for activating their payload [75, 77]. For example, the malware explained in the motivating example monitors the broadcast event BOOT_COMPLETED to trigger as soon as the device completes the boot process. Moreover, there are multiple broadcast events that the malicious application registers statically in its manifest file for triggering actions, as described in [77]. Therefore, the Interaction-module uses adb commands to send different broadcast receiver events to the targeted malware application for activating the malicious payload.

B.4 Data Collection Module

GLOBL: We used a script, written in C, that reads the operating states of all the GLOBL channels mentioned in Table 3, logs the timestamp, and saves these values to a file. The script was cross-compiled using the NDK toolchain [4] and runs natively in the client. This binary runs in the background while the profiled application is running in the foreground. The average sampling frequency for collecting the GLOBL data was 3.5 kHz.

HPC: The HPC events are monitored using Android’s simpleperf tool which is a native CPU profiling tool for Android [18]. Using simpleperf, we can repurpose the hardware counters offered by Snapdragon’s Performance Monitoring Unit, to record the HPC events that are stated in Table 2. We used the stat command of simpleperf to sample the accumulated counter data for the profiled application every 100 ms. We monitor the entire application which includes monitoring all the threads of the application. We note that the HPC measurement setup doesn’t account for evasion techniques like process injection, where the malicious application runs in the context of other benign processes [16].
B.5 Orchestrator module: High-level flow

Orchestrator module. The Orchestrator module is a Python script executing on the Host, responsible for invocation and synchronization of the execution of application-under-analysis, the Data-Collection module, and the Interaction module. The orchestrator also performs the checksum extraction to verify the integrity of the userdata and system partitions and pushes the collected data logs to the cloud. A high-level flow of the orchestrator module is as follows:

1. In the host-client sandbox setup, the client starts with a clean image of the rooted OS. (2) The application-under-analysis is transferred to the Android device and the reverse tethering process is started for providing internet access to the device. A pre-infection checksum of all the partitions is obtained for integrity verification. (3) The application is installed on the device followed by the simultaneous invocation of the application execution, Data-collection module, and the Interaction module. The application is run for 40 seconds with the Monkey interacting with the application for the first 30 seconds followed by sending broadcast events to the application for the next 10 seconds. At the end of the execution, the malicious process is killed. (4) Once the hardware telemetry collection is finished, the application is killed and the generated data logs are pulled from the analysis device to the host machine. A post-infection checksum of the partitions is obtained and the entire OS is flashed if there is any violation in the integrity of non-userdata partitions, else only the userdata partition is flashed. (5) Post-filtering is performed using the Logcat-based filter to ensure that we only use the hardware telemetry logs for iterations in which the application is executed. For the benign applications, we uninstalled the application between different iterations and rebooted the device between iterations, considering such applications do not affect non-userdata partitions.

C Analysis

C.1 Dataset Split

(SPLT-case-1) Training and Testing the individual classifiers for each GLOBL channel: The complete dataset is split into Training and Testing with a 70-30 split. (SPLT-case-2) Training and Testing with a 70-30 split. (SPLT-case-2) Training and Testing the individual classifiers and the Interaction module. The orchestrator also performs the checksum extraction to verify the integrity of the userdata and system partitions and pushes the collected data logs to the cloud. A high-level flow of the orchestrator module is as follows:

1. In the host-client sandbox setup, the client starts with a clean image of the rooted OS. (2) The application-under-analysis is transferred to the Android device and the reverse tethering process is started for providing internet access to the device. A pre-infection checksum of all the partitions is obtained for integrity verification. (3) The application is installed on the device followed by the simultaneous invocation of the application execution, Data-collection module, and the Interaction module. The application is run for 40 seconds with the Monkey interacting with the application for the first 30 seconds followed by sending broadcast events to the application for the next 10 seconds. At the end of the execution, the malicious process is killed. (4) Once the hardware telemetry collection is finished, the application is killed and the generated data logs are pulled from the analysis device to the host machine. A post-infection checksum of the partitions is obtained and the entire OS is flashed if there is any violation in the integrity of non-userdata partitions, else only the userdata partition is flashed. (5) Post-filtering is performed using the Logcat-based filter to ensure that we only use the hardware telemetry logs for iterations in which the application is executed. For the benign applications, we uninstalled the application between different iterations and rebooted the device between iterations, considering such applications do not affect non-userdata partitions.

C.2 Feature Engineering

GLOBL. The GLOBL channels consist of multivariate time-series signatures (Figure 3.(a)), with each channel capturing behavioral information from a different sub-device. Unlike the HPC channels, each sample at a particular time stamp is a state instead of event counts. Therefore, while the individual states are important, the variation of states with the passage of time contains valuable information as well. Windowed FFT is performed on each channel, resulting in a spectrogram where the x-axis is time, the y-axis is frequency, with the corresponding entries as the amplitude of the Fourier Transform. This is followed by performing dimensionality reduction on both the time axis and the frequency axis using Principal Component Analysis. Next, we select the PCA components that capture >95% variance in the features. Finally, the features from all the channels are stacked resulting in a tensor of dimension num_channels x feature_size. The high sampling frequency of the GLOBL channels warrants the conversion of the time domain signals into the frequency domain, as windowed FFT captures a compact representation of the highly sampled time series.

HPC. For the HPC channels, each sample is a vector made up of event counts at the time of sampling. Figure 3.(b) shows the feature engineering steps for one HPC-group with three different channels. First, the multivariate time series is divided into 32 equal parts. Next, the raw values are summed over in each interval to form a feature vector of size 32 for each variate of the time series. Finally, the feature vectors are flattened to get a single feature vector of size 1 x feature_size, which represents the information captured by a single HPC group. The feature engineering choices for the HPC are made by considering the jitter and noise introduced due to the limitations in the measurements. The process of binning preserves the raw values, reduces the effects of jitter [76], and evens out the noise [39], resulting in better-trained classifiers.

C.3 Failure modes of different GLOBL channels

Figure 10 presents the decisions of the different GLOBL channels on a set of samples (subset of 25 samples selected for visualization). Each square represents a decision, where a black square is a correct decision, and a white square is an incorrect decision.
C.4 Interpretation of second-stage model parameters

Figure 11 presents the learned weights assigned to the first stage models by the second stage model when the decisions are fused using Stacked Generalization. A higher absolute weight corresponds to higher importance being assigned to the corresponding channel, when arriving at the final decision. We can observe that chn-13 (#received-bytes) and chn-14 (#transmitted-bytes) have a higher weight underscoring the importance of network interaction when identifying malicious workloads. For the fused-group-3, we can also observe that the HPC telemetry has a significantly higher weight as compared to the other fused-groups. This can be attributed to the incorporation of raw-crypto-spec in the HPC-group-3 which is able to potentially capture a subset of the network interaction.