CloudShield: Real-time Anomaly Detection in the Cloud

ABSTRACT

In cloud computing, it is desirable if suspicious activities can be detected by automatic anomaly detection systems. Although anomaly detection has been investigated in the past, it remains unsolved in cloud computing. Challenges are: characterizing the normal behavior of a cloud server, distinguishing between benign and malicious anomalies (attacks), and preventing alert fatigue due to false alarms.

We propose CloudShield, a practical and generalizable real-time anomaly and attack detection system for cloud computing. Cloudshield uses a general, pretrained deep learning model with different cloud workloads, to predict the normal behavior and provide real-time and continuous detection by examining the model reconstruction error distributions. Once an anomaly is detected, to reduce alert fatigue, CloudShield automatically distinguishes between benign programs, known attacks, and zero-day attacks, by examining the reconstruction error distributions. We evaluate the proposed CloudShield on representative cloud benchmarks. Our evaluation shows that CloudShield, using model pretraining, can apply to a wide scope of cloud workloads. Especially, we observe that CloudShield can detect the recently proposed speculative execution attacks, e.g., Spectre and Meltdown attacks, in milliseconds. Furthermore, we show that CloudShield accurately differentiates and prioritizes known attacks, and potential zero-day attacks, from benign programs. Thus, it significantly reduces false alarms by up to 99.0%.

1 INTRODUCTION

The importance of cloud computing has grown significantly in the past years. Cloud customers can lease virtual machines from the cloud providers economically, sharing the physical resources provided by the cloud computing servers. Large cloud providers, like Amazon AWS, Google Cloud Platform, and Microsoft Azure, have proliferated this trend.

There have been various attacks against cloud computing, especially on shared resources. For example, security-critical information, e.g., encryption keys, can be leaked by cache side-channel attacks. Previous work have revealed that many types of cache side-channel attacks can successfully obtain secret or private cryptographic keys [6, 26, 30, 39]. Recently, speculative execution attacks [23–25] exploit performance optimization features of modern processors to breach the user-user, user-kernel or user-hypervisor isolation. Besides, zero-day attacks introduce challenges as they do not have known code nor known behavior.

Anomaly detection techniques are perhaps the only viable solution for detecting unknown zero-day attacks. By its nature, anomaly detection does not look for specifics of an attack but models the normal behavior of a system. Deviation from normal behavior indicates anomalies: either an attack or a benign anomaly.

However, existing anomaly detection systems in the cloud have challenges. First, a model of cloud server behavior is usually scenario-specific and is not easy to extend. Multiple models have to be built to cover various cloud workloads. Second, false alarms in anomaly detection systems are very common in practice. The large volume of false alarms overwhelms the security analysts and causes alert fatigue, potentially causing real attacks to be missed.

In this work, we investigate three questions. First: *How to model the different cloud workloads?* We hypothesize that the normal behavior of a cloud server, although different from workload to workload running on it, consists of a major predictable part, and a minor unpredictable part that follows a certain probability distribution. If we pre-train a general model to predict a cloud system's behavior, an anomaly can be detected by subtracting the predictable part from the original behavior markers and identifying the distribution of the remaining unpredictable part. To this end, we propose that the distribution of the unpredicted part denoted **Reconstruction Error Distribution (RED)**, can capture the characteristics of cloud workload. We show that rather than deploying an individual model for each workload, a general pretrained predictor model is leveraged, and anomalies are identified by statistically comparing the REDs.

The second question we investigate is: *How to select appropriate behavior markers to detect anomalous behavior in the cloud in real-time?* Quick detection of anomalies and attacks can prevent further damage. To support real-time anomaly detection in the cloud, we need an approach to select appropriate behavior markers that can be measured at high frequency and can reliably represent the system's behavior. To this end, we propose a principal component analysis (PCA)-based behavior marker selection method, and leverage the hardware performance counters, which are originally designed to monitor system performance and can be measured at high frequency, as exemplary markers to support real-time protection.

The third question we explore is: *How to distinguish benign anomalies and malicious attacks?* In practice, the "benign anomalous" behavior of a cloud system is quite common. For example, a cloud server used for database applications may be scheduled a different task when its workload is low. The missing piece in the past anomaly detection is the ability to correctly recognize the new tasks as benign. Otherwise, a large number of false alarms are raised, causing the system to be no longer usable. In this work, we refine each detected anomaly with the identification of benign anomalies and known attacks as a second step. This can significantly alleviate the false alarm problem in anomaly detection.

Section 2 describes the background. Section 3 presents the threat model. Section 4 discusses key challenges for anomaly detection in cloud computing. Section 5 describes our CloudShield methodology and Section 6 evaluates our design.

2 BACKGROUND

2.1 Attacks in Cloud Computing

There have been many attacks on cloud computing. We focus on the rapidly growing and representative class of software attacks on shared hardware resources in cloud servers. Two main types are speculative execution attacks and cache-based side-channel attacks, which we use as example attacks in the evaluation of our anomaly detection system. We also include software attacks, e.g., buffer overflow, in our evaluation. Our system is not tailored at all to defeat

these attacks, and the goal of our system is to detect even zero-day attacks, which are attacks that have never been seen before.

2.1.1 Speculative Execution Attacks. Since their first appearance in January 2018, speculative execution attacks [24, 25] have bombarded the world, with new variants continuously popping up. These attacks can leak the entire memory and break the software isolation provided by different virtual machines in the cloud, different virtual address spaces, and even by secure enclaves provided by SGX [38]. These attacks allow transient instructions to execute, illegally access a secret, and change the microarchitectural state based on the secret [16].

2.1.2 Cache Side-channel Attacks. Cache-based side-channel attacks are timing attacks that have traditionally been used to leak the secret key of symmetric-key ciphers or the private key of public-key ciphers, thus nullifying any security provided by such cryptographic protections [17]. Two representative cache side-channel attacks are the flush-reload attack and prime-probe attack. A variant of the flush-reload attack, i.e., the flush-flush attack [13], exploits the early abort if the cacheline to be flushed is not in the cache.

2.1.3 Buffer Overflow Attacks. A buffer overflow attack [37] occurs when the written data exceeds the size of an allocated buffer. Buffer overflow attacks can be exploited by an attacker to insert code and data. A buffer overflow attack is usually triggered by malformed input to write executable code or malicious data to a destination that exceeds the size of the buffer. If the malicious code or wrong data is used in the program, erratic program behavior would occur, e.g., system crash, incorrect results, or incorrect privilege escalation.

2.2 Hardware Performance Counters

Hardware performance counters (HPCs) are special registers that record hardware events. HPCs are widely available in commodity processors, including Intel, ARM, AMD, and PowerPC. Processors have been equipped with a Performance Monitor Unit (PMU) to manage the recording of hardware events. HPCs measure hardware events like the number of cache references, the number of instructions executed, and the number of branch mis-predictions; they also measure system events, like the number of page faults and the number of context switches.

Although the HPCs were designed for system performance monitoring and software debugging, previous work have also shown the feasibility of using hardware performance counters in security, e.g., detecting malware [10, 31], firmware-modification [36] and kernel root-kits [35]. Unlike these existing work, CloudShield leverages the reconstruction error distribution of HPCs, rather than directly using the noisy HPCs for anomaly detection.

3 THREAT MODEL

The target system is a cloud-based Infrastructure-as-a-Service (IaaS) system, where programs share hardware resources. The programs running on the IaaS platform may interfere with each other. As is commonly done, important and frequently used cloud services are scheduled one main task per machine, or per processor core, e.g., machine learning training, database query, MapReduce, or being used as a web or stream server. New tasks can be scheduled on the same core if the workload of the main task is low.

Our threat model covers attacks that breach the confidentiality and integrity of the cloud computing system. Availability attacks are not specifically covered in our threat model. Notably, the side-channel attacks and the recently proposed speculative execution attacks are considered in this threat model. We assume the attacker can launch attack programs in the cloud. We assume that an attack program can hide by switching between running and sleeping.

Our threat model particularly includes zero-day attacks. Unlike signature-based attack detection, we do not make particular assumptions about the attacks. We assume that there is no prior knowledge of attack code and the way the adversary interferes with the system.

Furthermore, once an anomaly is detected, we explicitly consider reducing false alarms caused by other benign programs that concurrently run. These benign programs need to be distinguished from attacks, otherwise, they can cause a large number of false alarms. Consequently, cyber analysts can be overwhelmed by false alarms and miss real attacks, making the detection system ineffective in practice. Therefore, discriminating benign programs, known attacks, and zero-day attacks is an important component in this work.

4 CLOUDSHIELD CHALLENGES

We first identify three challenges of anomaly detection in the cloud, and how they can be handled:

(1) How to model the different cloud workloads?
(2) How to select appropriate behavior markers?
(3) How to Distinguish Benign Anomalies and Malicious Attacks?

4.1 How to Model Different Cloud Workloads?

Our intuition of modeling cloud workloads, which may vary a lot in their functionalities, scales, and required resources, is that ***the behavior of a cloud server running a common cloud workload can be decomposed into two parts: a major predictable component and a minor unpredictable component. The predictable component can be predicted by a pre-trained model. The unpredictable component follows an unknown but fixed distribution.*** We will validate this hypothesis in Section 6.

We present a running example to illustrate this idea in Figure 1. Two sine curves plus subtle perturbations are shown in the top row, marked green and yellow, respectively. By looking at only these two raw measurements, one may not be able to tell the difference. We then subtract the predictable signal (the blue sine curves in the second row) to get the remaining part in the third row and examine the distribution of this remaining unpredictable part (bottom row). It shows that the probability distribution of the remaining part, which we denote the reconstruction error distribution (RED) from a prediction model, amplifies the difference.

With this assumption, rather than an individual model for each workload, we can pre-train a general workload behavior predictor model M for the predictable component, and subtract the prediction from the observed measurement of the system. The distribution of the remaining unpredictable component, i.e., the reconstruction error distribution (RED), can reveal the normal behavior from anomaly. We leverage RED as the key to anomaly detection. Stealthy attacks can be subtle and hide within normal measurements. However, subtracting the major predictable component from the observed

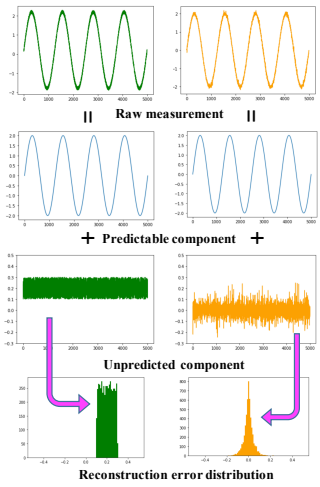

Figure 1: A running example of the reconstruction error distribution. Our intuition is that the behavior of a system can be decomposed into two parts: a major predictable component and a minor unpredictable component. If we can separate the predictable from the unpredictable component using a prediction model, the difference between normal and anomaly is more clearly revealed.

measurements amplifies the anomalous behavior and provides a robust way of detecting sneaky anomalies.

4.2 How to Select Appropriate Behavior Markers?

Modern processors usually provide counts of various events that can be used as behavior markers. Monitoring all of them is inefficient, if not impossible. Therefore, we need a method to choose the appropriate behavior markers from all possible markers that can represent the normal behavior of a system.

Our key idea for selecting behavior markers is to quantify the relative importance of the selected events representing the normal behavior of a system. Given the set of all possible behavior markers $b_1, b_2 ... b_n$, we can define a metric f to evaluate the relative contribution of a marker in representing the normal behavior of the system. Then, the behavior markers are sorted in descending order according to $f(b)$. The markers that exceed a certain threshold of importance are selected as candidate markers. In our implementation, we define f based on principal component analysis (PCA). Other metrics can also be leveraged to automatically select behavior markers.

4.3 How to Distinguish Benign Anomalies and Malicious Attacks?

The ultimate goal of CloudShield is to detect attacks, i.e., malicious anomalies. Once an anomaly is detected, the next step is to determine if it is a benign anomaly or a malicious attack. Without loss of generality, we simplify the discussion by making the assumption that a processor core runs one cloud workload, e.g., a stream server or a web server. A malicious anomaly can be a known attack or a zero-day attack. A benign anomaly can be benign programs that run concurrently with the cloud workload, where their interference could potentially cause false alarms. It could also be a stealthy attack that looks like a benign program.

Note that the key difference between a cloud workload and a benign program is that the cloud workload, as is commonly done, is the one main task per cloud server, or per processor core, while benign programs are relatively small programs that can be scheduled on the same core if the main task (cloud workload) is low.

While anomaly detection systems typically fall short of detecting benign versus malicious anomalies, Cloudshield can detect not just anomalies, but also the subset of anomalies that are attacks. Specifically, CloudShield builds two detectors, one is to identify known benign programs, and the other is to identify attacks. Also, while actual attack detection tends to be very domain-specific, our new contribution is to show that it is possible to use a general framework based on a pre-trained model to do attack detection. We are even able to detect stealthy attacks and potential zero-day attacks.

5 CLOUDSHIELD

5.1 Overview

We show an overview of CloudShield in Figure 2. There are three phases for learning and detecting anomalies and attacks in the cloud: *1) offline training and profiling*, *2) online anomaly detection and mitigation* and *3) online attack versus benign program detection*.

The *offline training and profiling phase* consists of four steps:

① constructing three sets of programs: normal cloud workloads, known attacks, and certified benign programs. The cloud providers select representative workloads. They also must access attack databases, and they probably already have benign program certifications which they can check or bring into a database of their own for this anomaly and attack detection system. A *Certificate Validation Module* is responsible for verifying the certificates of the workload and benign programs. The certificates are generated by trusted entities, e.g., companies that create these programs, and organizations or labs that verify the correctness and security of the programs. The certificate must contain the hash of the program binary and the public key signature of the trusted entity.

② The cloud providers execute the workloads and programs in an offline clean environment and collecting their behavior markers. A *Program Behavior Collection Module* is designed for this.

③ Training a default program-behavior predictor model M in a *Training Module*. The cloud providers generate the normal model for their cloud environment themselves as setup for this anomaly-attack detection system.

④ Calculating the corresponding REDs RD_n, RD_a, and RD_b as the reference Reconstruction Error Distributions (REDs) for normal cloud workloads, known attacks, and benign programs, respectively. We use the distribution RD_n as the normal behavior of the processor core running a cloud workload, while RD_a and RD_b are used to further distinguish between known attacks and benign programs when an anomaly is identified. Note that the cloud workload needs to be paused before collecting HPCs and calculating REDs for attacks and benign programs. The normal cloud workload detector, known attack detector, and benign program detector are also computed.

The *online anomaly detection and mitigation phase* has two steps:

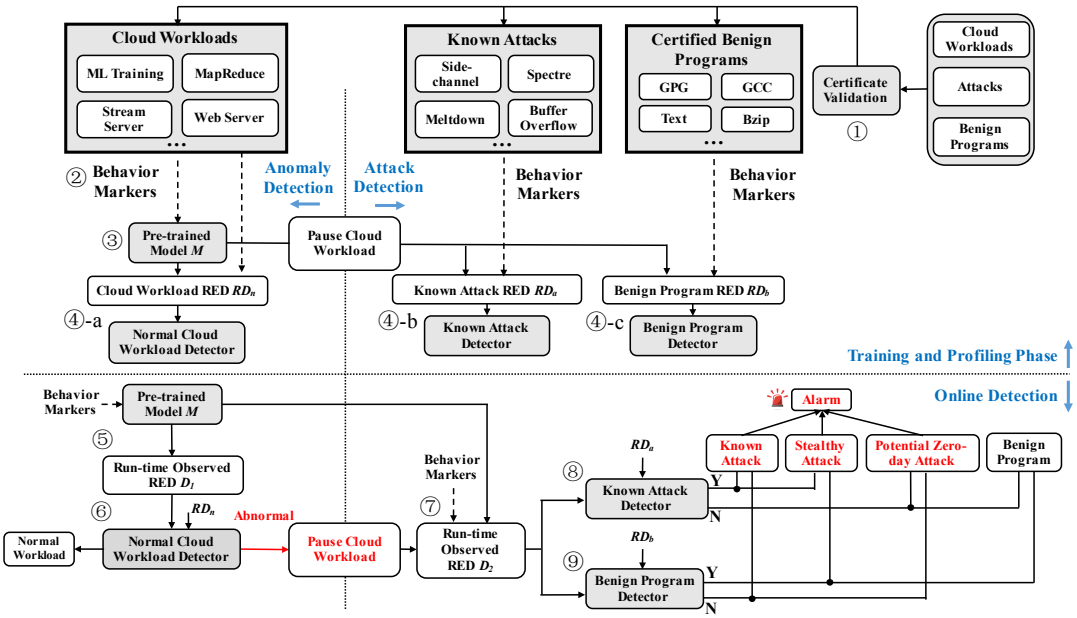

Figure 2: CloudShield methodology for anomaly and attack detection.

⑤ An *Online Detection Module* collects runtime behavior markers of each processor core in a cloud server from the Performance Monitor Unit (PMU) in the host OS. These markers are input into the pre-trained model M for the inference phase, to generate the run-time observed RED D_1.

⑥ Comparing the run-time RED D_1 and the reference RED RD_n. If D_1 does not follow the distribution of normal cloud workloads RD_n, an anomaly is detected and the cloud workload is paused to avoid further security breaches.

Once an anomaly is detected, the *online attack versus benign program detection phase* (step2 detection) is performed to distinguish benign programs from known attacks. This phase has three steps:

⑦ Collecting behavior markers when the cloud workload is no longer running. It is necessary to pause the cloud workload when running the attack detector and the benign program detector to eliminate the interference from the cloud workload, which is usually heavy. We observe that pausing the cloud workload can significantly increase the detection accuracy. The new measurements are inferred through the pre-trained model M and new RED D_2 is gathered.

⑧ Comparing D_2 to the distribution of known attacks RD_a to identify if the anomaly is caused by a known attack.

⑨ Comparing D_2 to the distribution of certified benign programs RD_b to identify if the anomalous behavior is a false alarm. Note that the steps ⑧ and ⑨ can be performed in parallel. As a complementary component, the cloud provider can confirm that benign programs are scheduled on this machine.

In the above discussion, we have assumed that a single pre-trained model of normal cloud workloads is sufficient, and that different known attacks can be detected with a single known attack detector, and that all benign programs added to a cloud workload can be identified with a single benign program detector. This significantly simplifies the implementation of CloudShield, and we will show that this results in excellent anomaly and attack detection in practice. More cloud workloads, attacks, and benign programs can always be added to the three sets of programs to retrain the model M and the three detectors.

The CloudShield implementation consists of four modules: a certificate validation module, a program behavior collection module, a training module, and an online detection module. The servers can share a set of the first three modules, as they are used during training phase. Only the last module needs to run on each cloud server.

5.2 Pre-training Program Behavior Predictor

Feature selection. Modern processors usually provide various events to be monitored by using hardware performance counters. However, due to the limited number of hardware registers in the PMU, only a few of them can be monitored at the same time. While round-robin scheduling of HPC measurements is feasible, it increases overhead. Therefore, it is important to select the appropriate events from all possible events as behavior markers. We propose a principal component analysis (PCA) based selection method to help determine the events to monitor. Our key idea is the selected events should be important to represent normal behavior.

Specifically, the principle component PCA_1 can be represented as a linear combination of all features. The coefficient of the corresponding HPC measurement represents the contribution of that feature in the principal component. Formally,

$$PCA_1 = ||x^T w||^2 \quad (1)$$

$$= \sum_i |w_i|^2 x_i^2 \quad (2)$$

where $x = (x_1, x_2 ... x_n)$ is an HPC reading of n events. $|w_i|$ is the coefficient of x_i in the first principal component. It represents the importance of event x_i in the first principal component.

We collect 34 HPC events from five representative cloud benchmarks, i.e., ML training (PyTorch), stream server (FFserver), database server (Mysql), web server (Nginx), and Hadoop MapReduce. We collect the event measurements for an entire processor core, to provide system-level monitoring, rather than just monitor a specific process or thread. We observe that although the benchmarks are different, they show consistency in the events' importance.

We use $\eta_i = \frac{|w_i|}{\sum_j |w_j|}$ as the event importance for a workload. We average η over the five representative benchmarks as the final importance score $\overline{\eta}$ of the corresponding event. We show the features with $\overline{\eta} \geq 1\%$ in Table 1. We use the thirteen selected events throughout the experiments. In fact, these are also the thirteen distinct events in the top-10 events for the five cloud workloads.

Table 1: HPC features with $\overline{\eta} \geq 1\%$.

Rank	Event	$\overline{\eta}$	Rank	Event	$\overline{\eta}$
1	Instruction	0.267	**8**	BPU read	0.030
2	Stall during issue	0.189	**9**	DTLB write	0.025
3	Stall during retirement	0.178	**10**	Branch	0.023
4	Cycles	0.106	**11**	L1D read miss	0.020
5	Load	0.067	**12**	L1I read miss	0.018
6	DTLB read	0.043	**13**	Context switch	0.015
7	Store	0.037			

Model selection. Recurrent Neural Network (RNN) and its variant, Long Short-Term Memory (LSTM), have become the popular model for sequential data. To balance the model complexity and its prediction power, in the proof-of-concept implementation, we start from a single-cell LSTM as the behavioral model of the system. We show that a simple single-cell LSTM model can already have enough good accuracy. More complicated models can be used to model additional normal workloads. An LSTM cell has three gates that control information flow: the forget gate, the input gate, and the output gate. LSTM automatically determines what information to "remember" and "forget".

Alternative models, e.g., Gated Recurrent Units (GRUs) [9] and BERT [11], can also be used as behavioral models of the system. As the main focus of this work is not to find the best model, but to show the feasibility of using RED of HPCs to detect anomalies in the cloud system, without loss of generality, we just show that LSTM models are enough for this anomaly detection.

Model training. Our goal is to train a model that can capture the predictable component of the behavior of a program. The program behavior markers $\{S_i\}_{i=1}^{N}$ (in our case HPC events of cloud workloads), are obtained from a clean environment. N is the total number of time frames collecting HPCs. In our experiments, each behavior measurement S_i^t is a vector consisting of the thirteen monitored hardware events. At time t, the deep learning model is trained to predict S_i^{t+1} using behavior history $[S_i^1, ..., S_i^t]$. Intuitively, since $\{S_i\}_{i=1}^{N}$ are normal behavior markers collected in the clean environment, the loss penalizes the incorrect prediction of normal behavior. We train this model to minimize the loss function with Stochastic Gradient Descent (SGD).

5.3 RED Profiling

RED generation of cloud workloads. We generate a profile of the normal cloud workloads in terms of reconstruction error distribution (RED), illustrated as RD_n in Figure 2. First, reference sequences of the behavior measurement, $R = [R^1, ..., R^{T'}]$, are collected in a clean environment. For this cloud server setting, each time frame R^i is a vector of thirteen dimensions (the number of monitored events) in our experiment. Second, at time frame t, we use the trained model to predict $t+1$ using the corresponding history behavior. We denote the prediction as P^{t+1}. The reconstruction error is defined as:

$$E(t) = R^{t+1} - P^{t+1} \tag{3}$$

Each reconstruction error sample $E(t)$ is a vector of dimension n, where n is the number of monitored events. We gather the prediction errors of each cloud workload and define the overall distribution of {E(1), E(2), E(3)...} from *all* workloads as RD_n.

KDE profiling of cloud workload. We use Kernel Density Estimation (KDE), a non-parametric estimation approach that better handles high-dimensional data, to profile the high-dimensional distribution of reconstruction errors from reference samples, denoted ④-a in Figure 2. We use non-parametric estimation because the formula of the RED of normal workloads is unknown, and its formula can be too complex to assume. KDE represents the distribution from elementary kernels. It assumes a small high probability area (Gaussian in our implementation) within a bandwidth around the observed samples, and sums them up as the probability distribution. Formally, KDE is defined as:

$$\hat{f}(x) = \frac{1}{nb} \sum_{i=1}^{n} K(\frac{x - x_i}{b}) \tag{4}$$

where $\hat{f}(x)$ is the estimated probability density. $K(\cdot)$ is a kernel function, whose value drops rapidly outside a bandwidth b. x_is are the samples from the distribution, i.e., E(t) in our case. n is the total number of samples.

In Figure 3, we show examples of reconstruction error distribution (RED) of normal cloud workloads (first five in green), benign programs (next six in blue), and attacks (last nine in red). To illustrate the high-dimensional distribution, we calculate the magnitude of REDs in Eq. 3 and observe that the normal cloud workloads, in general, have the smallest REDs (distributions to the left). Figure 3 shows clear difference between the cloud workloads, the benign programs, and the attacks. The cloud workloads have the smallest REDs (leftmost). The REDs of different benign programs are distinct. Most of the benign programs have larger REDs than cloud workloads, except the gpg-rsa program whose RED is similar to the cloud workloads. Moreover, the REDs of all evaluated attacks are to the right side, meaning larger reconstruction errors than cloud workloads and benign programs.

Profiling for benign programs and known attacks. Similarly, we profile the RED of the benign programs and known attacks. We collect their behavior data in a clean execution environment from the Program Behavior Collection Module. Interestingly, we observe that it is not necessary to train another program behavior predictor model for benign programs and attacks. The pre-trained one on cloud workloads can be reused to profile the benign programs and known attacks. We hypothesize that it is because pre-training on different workloads improves the generalizability of the model, by suppressing

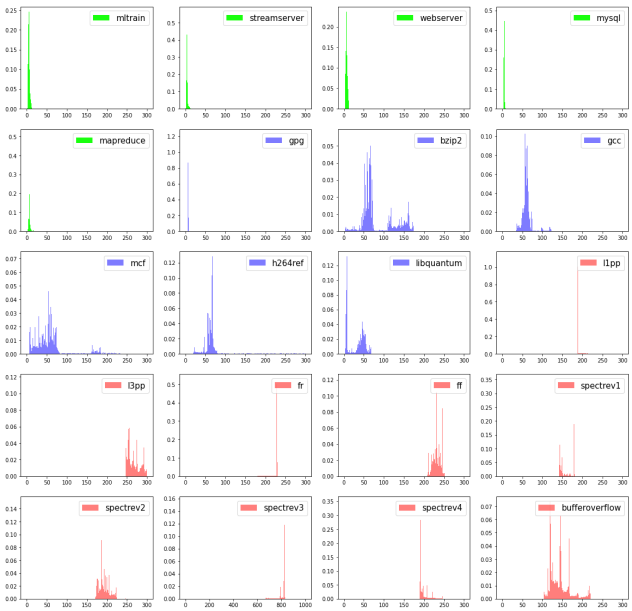

Figure 3: Reconstruction error distribution (RED) of normal cloud workloads (first five in green), benign programs (next six in blue) and attacks (last nine in red).

potential overfitting. At last, two KDE estimations are performed on the RED of known attacks and benign programs, shown as ④-b and ④-c in Figure 2, respectively.

We illustrate an example of kernel density estimation of benign programs in Figure 4. To illustrate, we first use t-SNE [20] to map the thirteen HPCs to a 2-D plane and build a KDE estimator of benign programs (gcc, gpg, and libquantum) using the REDs from the pre-trained model. The high-density regions (likely to be benign programs) are colored red while the low-density areas (unlikely to be benign programs) are colored blue. We plot three benign programs, i.e., gcc (green square), gpg (green diamond) and libquantum (green triangle) in Figure 4. We also depict four attacks, i.e., l3pp (red cross), fr (red square), spectre v1 (red diamond), and buffer overflow (red triangle), in Figure 4 and observe that they are all in the low-density area, where the benign program detector can identify them as non-benign programs. Figure 4 explains why KDE works, specifically the benign programs form high-density clusters while the attacks are outside the clusters.

5.4 Runtime Anomaly Detection and Mitigation

The online detection module is responsible for detecting anomalies and distinguishing attacks and benign programs at runtime. A processor core's behavior, in terms of hardware event measurements, is dynamically monitored at runtime.

Anomaly detection based on RED. Similar to the offline profiling phase, the runtime gathered HPC sequences are sent through the pre-trained model (⑤ in Figure 2) to obtain the runtime observed RED D_1. The likelihood of the observed reconstruction error following the RED of normal cloud workloads (RD_n) is computed using the

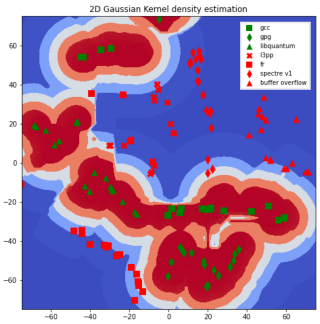

Figure 4: Illustration of kernel density estimation of benign programs. The high-density regions (likely to be benign programs) are colored red while the low-density areas (unlikely to be benign programs) are marked blue.

KDE normal workload detector ($\hat{f}(x)$ in Eq. 4) [1]. If the likelihood $\hat{f}(x)$ is lower than a pre-defined threshold, i.e., the prediction error does not follow the distribution of RD_n, an anomaly is detected.

Based on the results of the anomaly detection, different response actions can be taken. If no anomaly is detected, no further actions are required. Once an anomaly is detected, CloudShield triggers different responses (⑥ in Figure 2). First, the cloud workload running on the machine is temporarily paused to avoid further damage. This also eliminates the interference between the cloud workload and other tasks that concurrently run (attacks or benign programs). Second, access to the most security-critical data and resources is temporarily turned off. Attacks against data confidentiality, e.g., side-channels, can target these secret data. Thus, cutting access to the security-critical data prevents these data from being leaked out. Third, the known attack detector and benign program detector are woken up, to identify if the anomaly is malicious (an attack) or benign (a false alarm). This can further reduce false-alarm fatigue in practice, as discussed below.

5.5 Distinguishing Benign Programs and Attacks

A detected anomaly can be caused by benign programs. Thus, CloudShield attempts to distinguish "benign anomalies" caused by benign programs versus real attacks. As discussed in Section 5.4, the cloud workload is paused once an anomaly is detected (⑥ in Figure 2). Now the monitored core is possibly running attacks. Moreover, other benign programs (can be a victim program) that concurrently run with the attack may hide the attack and make identifying attacks even harder. We will show CloudShield can detect an attack in both scenarios, with and without benign programs running.

Attacks and benign programs identification. To distinguish the attacks and benign programs, firstly, hardware events' measurements are monitored through the PMU after the main cloud workload is switched off. Then the PMU sends the newly measured data (without cloud workload) to the same pre-trained program behavior predictor M for inference. Similar to anomaly detection, we compute the RED D_2 in the form of Eq. 3. The KDE attack detector (④-b) and the KDE benign program detector (④-c) were loaded into the online detection

[1] Tree-based structures, e.g., KD tree, can be used to find the x_is close to x and accelerate the computation because the effect of x_is outside the bandwidth b is negligible.

module from the training module [2]. The attack detector computes the likelihood of the observed prediction errors following the RED of known attacks (RD_a), using Eq. 4. If a high likelihood is observed, the attack detector reports an attack. Similarly, the benign program detector computes the likelihood of the observed prediction error following the RED of benign programs (RD_b). If a high likelihood is observed, the benign program detector reports a benign program.

We catagorize four classes of definite (known) attacks, benign programs, stealthy attacks, and potential zero day attacks. The definite (known) attacks and benign programs have known patterns to the defender, and they are used in the training of the attack and benign program detector, respectively. The stealthy attacks are attacks that have similar behavior to the benign programs, possibly by mimicing the workflow of the benign programs. The zero-day attacks have unknown patterns which may not be similar to the benign programs. We map these four classes to the decisions of the two detectors, we list the four possible final decisions in Table 2.

Table 2: Benign program and attack decisions and responses.

	Known Attack Detector	Benign Program Detector	Decision	Response
Case 1	Y	Y	Stealthy attack	Alarm (high priority)
Case 2	Y	N	Attack	Alarm (high priority)
Case 3	N	Y	Benign program	Resume cloud workload
Case 4	N	N	Zero-day attack or new benign programs	Alarm (medium priority)

Case 1: The attack detector recognizes it as a known attack, and the benign program detector recognizes it as a benign program. In this case, CloudShield reports it as a stealthy attack where the attack program hides by mimicking the behavior of a benign program. Another possible scenario of this case is that a benign program, which could be a victim program, is concurrently running with the attack program. We will show in the experiments that attacks can still be detected even when they run together with benign programs. A high-priority alarm is raised.

Case 2: The attack detector recognizes it as an attack, and the benign program detector does not report it as a benign program. This case indicates clear attacks and a high-priority alarm is raised and a detailed report is sent for inspection.

Case 3: The attack detector does not report it as an attack, and the benign program detector recognizes it as a benign program. In this case, the previously detected anomaly is caused by a benign program. The cloud workload is resumed to execute and no alarm is raised.

Case 4: The attack detector does not report it as a known attack, and the benign program detector does not report it as a benign program. In this case, a potential zero-day attack or an unknown benign program is possible. A medium-priority alarm is raised by CloudShield. The cyber analysts can handle these alarms after the high-priority alarms. In fact, in our experiments, we show that case 4 is very unlikely.

Response. Once an anomaly is detected (step 1), CloudShield has already paused the normal cloud workload to shield it from the attacks. Access to highly sensitive data, code, and resources can also be denied, depending on the server's security response policy. If in the second step, an attack is detected, an alarm will be raised. Further responses can be taken to protect the system, and the code and data on it. CloudShield can also stop all processes running on the core. Meanwhile, CloudShield records the relative information into logs for further investigation.

System update. We discuss possible system updates of CloudShield. Specifically, CloudShield can update itself if new types of cloud workloads are added, new attacks are discovered or new benign programs are certified. A new model has to be trained only if new cloud workloads are added. For new attacks and benign programs, only the KDE detectors for attacks and benign programs need to be updated. Detailed discussions can be found in Appendix A.1.

[2] Note that here we only need two KDE estimators, one for attacks and the other for benign programs, rather than an individual detector for each attack or benign program.

6 EVALUATION

6.1 Experimental Settings

Platform. We perform our evaluation of CloudShield on a server equipped with 2 Intel Xeon E5-2667 CPUs, each with 6 physical processor cores. Each core has a 32KB L1D (Level-1 Data) cache and a 32KB L1I (Level-1 Instruction) cache. Each package of six cores shares a 256KB L2 (Level-2) cache and a distributed last-level cache of 15MB (2.5MB*6). The server has 64GB memory and a 2TB hard disk. The machine is also equipped with an Nvidia 1080Ti GPU. The HPC values are collected every 10 milliseconds using *Perf* [4] supplied by the Ubuntu 14.04.6.

Cloud workload benchmarks. We choose five representative cloud benchmarks, as shown in Table 3.

Table 3: Cloud workload benchmarks.

Cloud workload	Description
Web server (Nginx)	Serving 1000 remote connections to request webpages using WRK benchmark [2]
Database server (Mysql)	Performing 128 concurrent queries using SysBench [3]
Stream server (FFserver)	Streaming a MPEG video in real-time to a remote user with FFserver and FFmpeg
ML training (Pytorch)	Training an LSTM model using an Nvidia 1080Ti GPU
Hadoop	Perform Terasort [5] using MapReduce

Evaluated attacks. We select nine representative runtime attacks against cloud computing systems for evaluation (Table 4). The evaluated attacks are cache side-channel attacks, speculative execution attacks, and buffer overflow attacks. The cache side-channel attacks silently leak information. The four recently discovered speculative execution attacks represent the main hardware resources exploited by the different speculative attack variants. We also evaluate a representative software attack, i.e., buffer overflow attack.

Table 4: Evaluated attacks.

Catagory	Attack
Cache side-channel attacks	L1 cache prime-probe attack (l1pp) [14]
	L3 cache prime-probe attack (l3pp) [26]
	Flush-reload (fr) [39]
	Flush-flush (ff) [13]
Speculative execution attacks	Speculative boundary bypass (spectre v1) [24]
	Indirect branch mis-prediction (spectre v2) [24]
	Meltdown (spectre v3) [25]
	Speculative store bypass (spectre v4) [1]
Buffer overflow	Stack overflow attack [37]

Benign programs. We choose representative benign programs from the SPEC2006 benchmark suite [19]. The evaluated benign programs cover a large scope of programs: crypto software (gpg-rsa), compiler (gcc), file and video compression tools (bzip2, h264ref), scientific computation (mcf, milc, namd, libquantum), statistics, and machine learning (soplex, hmmer) and gaming (gobmk).

Data collection. Data were collected in different scenarios. To evaluate the first step, i.e., for detection, we collected data when ① only the cloud workload is running; ② the cloud workload is running with benign programs listed above; ③ the cloud workload is running with the attacks listed above; and ④ the cloud workload is running with both benign programs and attacks. To evaluate the second step, i.e., for detection of attacks and benign programs, which we do when the cloud workload is not running, we collected data when ① only an attack is running; ② only a benign program is running; and ③ an attack is running together with a benign program. Due to the large number of combinations of cloud workloads, attacks and benign programs, we run each combination for six minutes on a server, and split the data equally into training, validation and testing sets.

Metrics We first compute an anomaly score for each behavior measurement and then use a threshold to determine False Positive Rate (FPR) and False Negative Rate (FNR). An anomaly score is $-log(\hat{f}(x))$, where $\hat{f}(x)$ is the KDE density in Eq. 4. Low density $f(x)$ indicates a high anomaly score. The threshold of the cloud workload detector is obtained such that 80% of the validation normal measurements during the training phase are correctly classified as normal (to avoid leakage, no attack data are used to construct the normal cloud workload detector nor to determine the threshold). For the benign program detector and attack detector, the threshold is obtained such that the equal error rate (EER) is achieved, on the validation set.

6.2 Overall: Anomaly+Attack Evaluation

As CloudShield first detects anomalies (step 1) and then identifies attacks and benign programs (step 2), we first illustrate the end-to-end (anomaly detection + attack detection) results in Table 5. Separated results and analysis of each step are discussed in Section 6.3-Section 6.5.

We evaluate different window sizes: if the window size is w, in step 1, w contiguous anomalous behavior marker measurements are identified as an anomaly. Similarly, w contiguous behavior marker measurements are collected before an attack or benign program can be identified. For a specific cloud workload, we report the average FPR for that cloud workload + each benign program. We report the average FNR for that cloud workload + each attack + each benign program we evaluated. A higher FPR increases the number of false alarms, while a higher FNR increases the chance that an attack will go undetected. Low rates of both are desired. We observe that the CloudShield indeed has very low FNRs for all 5 workloads for all window sizes - less than 0.3%, indicating excellent detection accuracy and hence, excellent security. FPRs are slightly higher but also less than 0.6%. When w=1, the webserver workload has the highest FPR (0.51%), while stream server achieves the lowest FPR (0.26%). For all five cloud workloads, the FPR decreases as w becomes larger, however, the FNR increases accordingly. When w = 100, FPR decreases to 0.13% (for stream server) and 0.24% (for webserver). FNR increases to 0.09% and 0.19%. When w = 200, FNR tends to exceed FPR for all five cloud workloads. Note that a larger window size can increase the detection delays (evaluated in Section 6.6). A window size of 5-10 should be sufficient.

Table 5: Quantitative end-to-end (anomaly detection + attack detection) evaluation results.

	ML training (Pytorch)		Database (Mysql)		Stream server (FFserver)		Webserver (Nginx)		MapReduce (Hadoop)	
w	FPR	FNR	FPR	FNR	FPR	FNR	FPR	FNR	FPR	FNR
1	0.0034	0.0005	0.0033	0.0005	0.0026	0.0011	0.0051	0.0005	0.0032	0.0005
3	0.0034	0.0005	0.0033	0.0005	0.0025	0.0012	0.0050	0.0005	0.0031	0.0005
5	0.0033	0.0005	0.0032	0.0005	0.0025	0.0012	0.0049	0.0005	0.0031	0.0005
10	0.0032	0.0005	0.0031	0.0005	0.0024	0.0013	0.0048	0.0005	0.0030	0.0005
20	0.0030	0.0006	0.0029	0.0006	0.0023	0.0014	0.0045	0.0006	0.0028	0.0006
50	0.0025	0.0007	0.0024	0.0007	0.0019	0.0017	0.0036	0.0007	0.0023	0.0007
100	0.0016	0.0009	0.0016	0.0009	0.0013	0.0019	0.0024	0.0009	0.0015	0.0009
200	0.0005	0.0011	0.0005	0.0011	0.0004	0.0025	0.0008	0.0011	0.0005	0.0011

We compare the proposed CloudShield to four representative anomaly detection methods in the literature, i.e., Isolation Forrest (IF) [27], One-class SVM (OCSVM) [33], Local Outlier Factor (LOF) [7], and Principal Component Analysis (PCA) [21]. We show the end-to-end results in Table 6. For the existing anomaly detection methods, we replace the pretrained model + KDE in steps 1 and 2 of CloudShield with the corresponding method. We average the FPR and FNR across each combination of cloud workload, benign program, and attack. We observe that, with w=5 or w=10, CloudShield achieves lower FPR and FNR compared to other methods. Specifically, when w=5, the best FPR and FNR of existing methods are 1.41% (OCSVM) and 6.95% (PCA), respectively, while CloudShiled has much lower (better) FPR of 0.34% and FNR of 0.06%. Similar results are shown when w=10.

Table 6: Compare CloudShield to existing anomaly detection methods.

		False Positive Rate (FPR)	False Negative Rate (FNR)
w=5	Isolation Forrest (IF)	0.1728	0.442
	One-class SVM (OCSVM)	0.0141	0.1011
	Local Outlier Factor (LOF)	0.0518	0.0956
	PCA	0.0587	0.0695
	CloudShield	0.0034	0.0006
w=10	Isolation Forrest (IF)	0.1539	0.416
	One-class SVM (OCSVM)	0.01571	0.1031
	Local Outlier Factor (LOF)	0.0516	0.0990
	PCA	0.0519	0.1150
	CloudShield	0.0033	0.0007

6.3 Step 1: Can CloudShield Detect Anomalous Behavior in Realtime?

A key challenge for real-time anomaly detection is short or stealthy attacks. Attacks can hide by switching between running and sleeping. A good anomaly detection system should be able to capture the attack once it is running. We evaluate CloudShield against such attacks and show it can detect them almost immediately. We schedule each of the nine attacks to run and then sleep for a random period (10s-40s) before the next attack runs. The experiment is performed when the ML training workload is running.

We show the attack scheduling and the anomaly scores output ($-log(\hat{f}(x))$ in Eq. 4) by CloudShield in Figure 5. It is clear that once an attack is running, possibly after sleeping, CloudShield captures

it (indicated by a large anomaly score). Once the attack program's behavior is suspended, the anomaly score quickly goes back to a low value. Therefore, the proposed CloudShield can detect anomalies in real-time. We also observe that two last-level cache attacks, i.e., the flush-flush attack (ff) and the LLC prime-probe attack (l3pp), and two speculative execution attack variants, i.e., the spectre v2 and the spectre v3, result in higher anomaly scores than the other attacks, indicating their distinctive behavior.

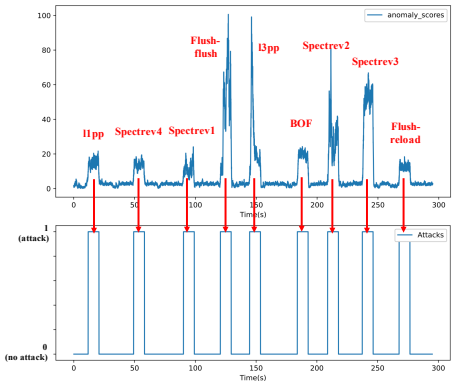

Figure 5: Real-time detection of anomalies. We schedule each of the nine attacks to run for 10 seconds and sleep for a random period of time (10-40s).

6.4 Step 2: Can CloudShield Detect Zero-day Attacks?

We evaluate the anomaly detection (step 1) on the nine attacks, including the four recently proposed speculative execution attacks. *Note that in the anomaly detection step, CloudShield is only trained on the normal behavior of the cloud workloads, and has not seen code or data of any of the nine attacks, so they are like zero-day attacks to CloudShield in this experiment.*

We consider the model predictions for the four scenarios:

(1) Normal workload
(2) Normal workload and a benign program running
(3) Normal workload and an attack running
(4) Normal workload, a victim program, and an attack running

We show quantitative results of anomaly detection (step 1) in Table 7. We find that when only the normal workload is running (scenario 1), CloudShield almost always correctly recognizes it as normal (the first line) for the ML training, database, stream server, and web server benchmarks with a 0.1%-0.5% false positive rate (predict abnormal column). When MapReduce is running, CloudShield misrecognizes 1.7% of normal workloads as anomalous – still a small level.

When a benign program is running concurrently with the cloud benchmark (scenario 2), we observe that the results highly depend on the cloud workload and the benign program. For example, the GPG-RSA is recognized as normal with less than 1% false positive rate in the database, web server, and MapReduce workloads. However, large false positives, i.e., 6.4% and 47.6% of the GPG-RSA,

Table 7: Results of anomaly detection (step 1) with different cloud workloads.

		ML training (Pytorch)		Database (Mysql)		Stream server (FFserver)		Webserver (Nginx)		MapReduce (Hadoop)	
		FPR	FNR	FPR	FNR	FPR	FNR	FPR	FNR	FPR	FNR
	Cloud workload only	0.001	--	0.001	--	0.005	--	0.001	--	0.017	--
Cloud workload + Benign program	gpg-rsa	0.476	--	0.001	--	0.064	--	0.003	--	0.003	--
	gcc	0.780	--	0.619	--	0.791	--	0.822	--	0.382	--
	mcf	0.549	--	0.745	--	0.485	--	0.964	--	0.877	--
	libquantum	0.001	--	0.399	--	0.078	--	0.909	--	0.401	--
Cloud workload + Attack	l1pp	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	l3pp	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	fr	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	ff	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	spectrev1	--	0.012	--	0.000	--	0.016	--	0.000	--	0.000
	spectrev2	--	0.005	--	0.000	--	0.003	--	0.000	--	0.000
	spectrev3	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	spectrev4	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	bufferoverflow	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
Cloud workload + Benign program + Attack	l1pp + gpg-rsa	--	0.000	--	0.000	--	0.004	--	0.000	--	0.000
	l3pp + gpg-rsa	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	fr + gpg-rsa	--	0.020	--	0.000	--	0.045	--	0.000	--	0.000
	ff + gpg-rsa	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	spectrev 1 + gpg-rsa	--	0.000	--	0.000	--	0.005	--	0.000	--	0.000
	spectrev 2 + gpg-rsa	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	spectrev 3 + gpg-rsa	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	spectrev 4 + gpg-rsa	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	bof+ gpg-rsa	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	l1 pp + gcc	--	0.001	--	0.000	--	0.002	--	0.000	--	0.000
	l3 pp + gcc	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	fr+ gcc	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	ff+ gcc	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	spectrev 1 + gcc	--	0.000	--	0.000	--	0.001	--	0.000	--	0.000
	spectrev 2 + gcc	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	spectrev 3 + gcc	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	spectrev 4 +gcc	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	bof + gcc	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	l1 pp + libquantum	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	l3 pp + libquantum	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	fr+ libquantum	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	ff+ libquantum	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	spectrev 1+ libquantum	--	0.031	--	0.000	--	0.028	--	0.000	--	0.000
	spectrev 2 + libquantum	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	spectrev 3 + libquantum	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	spectrev 4 +libquantum	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000
	bof + libquantum	--	0.000	--	0.000	--	0.000	--	0.000	--	0.000

are observed in the stream server and ML training workloads, respectively. These false alarms can cause false alarm fatigue. Thus it requires the next step to further distinguish benign programs versus malicious anomalies, and reduce the number of false alarms. Note that CloudShield distinguishes certified benign programs from attacks (step 2) to reduce false alarms after an anomaly is identified (results discussed in Section 6.5).

In scenario 3, once an attack is running with the cloud workload, it can be detected with zero false negatives in the database, web server, and MapReduce workloads. For the ML training workload, the Spectre v1 and v2 attacks cause 1.2% and 0.5% false-negative rates, respectively. For the stream server workload, the Spectre v1 and v2 attacks introduce 1.6% and 0.3% false-negative rates, respectively. These results show that CloudShield is capable of detecting zero-day attacks, since the normal cloud workload detector in step1 has not been trained with any attack.

We also evaluate scenario 4 where an attack program runs concurrently with a benign or victim program and the cloud workload. Similar to scenario 3, we observe that the attacks can be detected with zero false negatives with the database, web server, and MapReduce workloads. For the ML training workload, the worst case is when spectre v1 and libquantum are running concurrently, the attack is missed by 3.1%, slightly higher than scenario 3 where the spectre v1 attack is running alone (1.2%). For the stream server workload, the highest false-negative rate is 4.5% when a flush-reload attack is executed with gpg-rsa. Next is when Spectre v1 and libquantum are running with the stream server, the FNR is 2.8%. Although these results from just step 1 for anomaly detection are very good for not missing attacks (low FNRs), the FPRs in scenario 2 is not low enough. Hence, we propose step 2, to detect benign anomalies from real attacks.

Consider all four scenarios, in our experiment, 95.4% of the detected anomalous behavior turns out to be actual attacks. This leaves little room for an attacker to perform a DoS attack.

6.5 Can CloudShield Distinguish Benign Anomalies from Attacks?

Anomalies can be caused by benign programs, i.e., benign anomalies. Therefore, once an anomaly is detected, CloudShield takes the next step to figure out whether it is a benign anomaly or an attack. As shown earlier, CloudShield implements two detectors to identify known attacks and certified benign programs, respectively. These two detectors can reduce false alarms by 99.0%.

We show a real example of CloudShield reducing false alarms by distinguishing known attacks and certified benign programs in Figure 6. We run an attack (spectre v3) and a benign program (gcc), both with the ML training workload. The periods ① and ③ indicate that the attack is running, and the period ② means the benign program is running. Figure 6 (a) illustrates the anomaly scores in the anomaly detection step. We observe that while both attacks are correctly identified (periods ① and ③), the beginning of gcc execution is incorrectly recognized as attacks (false alarms). Then the ML training workload is paused and the behavior measurements are re-collected as input to the two step 2 detectors. Figure 6 (b) shows the result of the attack detector. High values indicate an attack and low values mean no attack. It correctly identifies periods ① and ③ as attacks, while ② is not an attack. Figure 6 (c) shows the result of the benign program detector. High values represent a benign program and low values indicate a program that is not in the set of certified benign programs. We find that the certified benign program detector reports high values in period ② (and idle periods), while the values in periods ① and ③ are low (not certified benign programs). Jointly considering the two detectors, CloudShield correctly determines that ② is a certified benign program, while ① and ③ are real attacks.

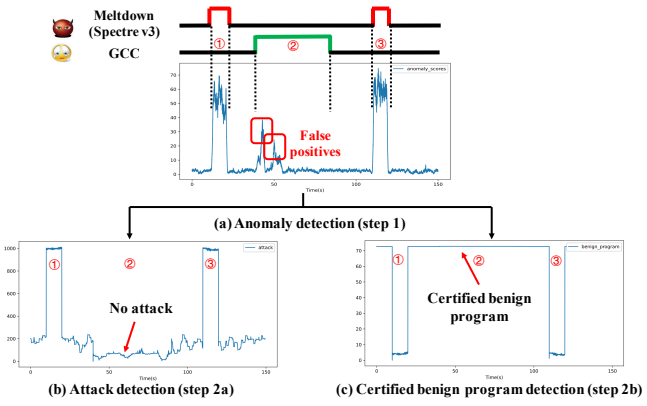

Figure 6: An example of reducing false alarms by identifying attacks and certified benign programs.

We show quantitative results of attacks and certified benign program detection (step 2) in Table 8. We select eleven representative benign programs from the SPEC benchmark suite and the same nine attacks as in previous sections for evaluation. For the benign program detection, we observe that six benign programs (gpg-rsa, bzip2, namd, soplex, hmmer, and libquantum) can be recognized correctly with no false alarms. The milc program introduces the highest but acceptable FPR of 3.5%. Of this, 2.6% were identified as stealthy attacks and 0.9% as zero-day attacks or unknown benign programs. On average, 99.0% of the benign programs can be identified correctly, i.e., the false alarms raised by benign programs in the anomaly detection is suppressed by 99.0%. Within the remaining false alarms (1.0%), we observe that 0.6% are recognized as case 4 (zero-day attacks or unknown benign programs) which results in a medium-priority alarm, and 0.4% are recognized as high-priority attacks (case 1 and 2). For attack detection, we observe that all attacks are correctly identified. A detailed analysis shows that 99.8% of attacks are identified as high-priority attacks (case 2) and 0.2% attacks are recognized as stealthy attacks.

We also consider a more difficult scenario where an attack is running concurrently with a certified benign program. We show that even if attack is spread out in an application, it can still be detected by its behavior via HPCs. 99.9% of attacks are correcly identified while 96.4% are identified as high-priority attacks (case 2). Detailed analysis are shown in Appendix .

Table 8: Results of benign programs/attacks detection (step 2).

		Pred benign (Case 3)	Pred attack (Case 1,2,4)	Case 1 (stealthy attack)	Case 2 (attack)	Case 3 (benign)	Case 4 (0-day)
None	None	1.000	0.000	0.000	0.000	1.000	0.000
Benign programs	gpg-rsa	1.000	0.000	0.000	0.000	1.000	0.000
	bzip2	1.000	0.000	0.000	0.000	1.000	0.000
	gcc	0.971	0.029	0.000	0.000	0.971	0.029
	mcf	0.987	0.013	0.000	0.000	0.987	0.013
	milc	0.965	0.035	0.026	0.000	0.965	0.009
	namd	1.000	0.000	0.000	0.000	1.000	0.000
	gobmk	0.983	0.017	0.017	0.000	0.983	0.000
	soplex	1.000	0.000	0.000	0.000	1.000	0.000
	hmmer	1.000	0.000	0.000	0.000	1.000	0.000
	libquantum	1.000	0.000	0.000	0.000	1.000	0.000
	h264ref	0.980	0.020	0.000	0.000	0.980	0.020
	Average	**0.990**	**0.010**	**0.004**	**0.000**	**0.990**	**0.006**
Attacks	l1pp	0.000	1.000	0.000	1.000	0.000	0.000
	l3pp	0.000	1.000	0.000	1.000	0.000	0.000
	fr	0.000	1.000	0.000	1.000	0.000	0.000
	ff	0.000	1.000	0.000	1.000	0.000	0.000
	spectrev1	0.000	1.000	0.000	1.000	0.000	0.000
	spectrev2	0.000	1.000	0.000	1.000	0.000	0.000
	spectrev3	0.000	1.000	0.000	1.000	0.000	0.000
	spectrev4	0.000	1.000	0.000	1.000	0.000	0.000
	bufferoverflow	0.000	1.000	0.021	0.979	0.000	0.000
	Average	**0.000**	**1.000**	**0.002**	**0.998**	**0.000**	**0.000**

Table 9: Results of zero-day (unknown) attacks in step 2.

		Pred benign (Case 3)	Pred attack (Case 1,2,4)	Case 1 (stealthy attack)	Case 2 (attack)	Case 3 (benign)	Case 4 (0-day)
Known Attack	l1pp	0.000	1.000	0.000	1.000	0.000	0.000
	l3pp	0.000	1.000	0.000	1.000	0.000	0.000
	spectrev1	0.000	1.000	0.000	1.000	0.000	0.000
	spectrev2	0.000	1.000	0.000	1.000	0.000	0.000
	bufferoverflow	0.000	1.000	0.021	0.979	0.000	0.000
Unknown Attack	fr	0.000	1.000	0.000	0.999	0.000	0.001
	ff	0.000	1.000	0.000	0.000	0.000	1.000
	spectrev3	0.000	1.000	0.001	0.000	0.000	1.000
	spectrev4	0.000	1.000	0.000	0.999	0.000	0.001

Zero-day attack detection in step 2. We conduct another experiment by putting only L1 prime-probe (l1pp), LLC prime-probe (l3pp), spectre v1, spectre v2, and buffer overflow attacks in the set of known attacks. This means that the flush-reload (fr), flush-flush (ff), spectre v3, and spectre v4 attacks are unknown zero-day attacks. We show the known and zero-day attack detection results in Table 9. We observe that CloudShield can still correctly recognize unknown attacks. The flush-flush and spectre v3 attacks are classified as case 4 (zero-day attacks). The other two attacks, i.e., the flush-reload and spectre v4 attacks, are detected as known attacks probably because their behavior is similar to the known attacks.

Necessity of the two-step method. We have also investigated detecting attacks together with detecting anomalies in the first step, when the cloud workloads are running. The benefit of doing this is that the attacks can be identified more quickly. However, the downside of detecting attacks in the first step is that the attacks and cloud workloads interfere with each other, making the behavior markers collected in the first step not capable enough to identify the attacks. Hence our two-steps method is much better.

6.6 Detection Latency and Overhead

Detection latency. The detection latency is defined as the period from the time the attack starts running, to the time an attack alarm is raised. We present the overhead of robust detection using more than one set of behavior marker measurements, e.g., with a sequence of $w = 5$ sets of measurements. The timeline for detecting an attack is shown in Figure 7 (similar for attack and benign program detection). t_B denotes the time interval for collecting w behavior marker measurement. t_{RED} represents the time needed for computing the RED by inferencing the pre-trained model. t_{KDE} is the time to infer the KDE detector. The computation of RED and KDE can overlap with the HPC collection if $w > 1$ (Figure 7).

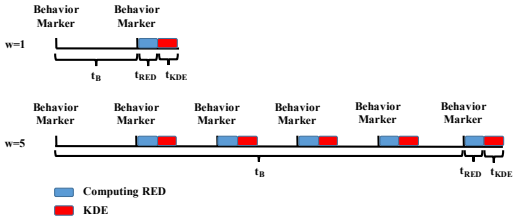

Figure 7: Illustration of the timeline for anomaly detection.

Table 10 presents the detection latency when w =1, 5, 10, 50 and 100. As the HPCs are sampled every 10ms, t_B=10ms when w=1. We measure t_{RED} and t_{KDE} on the server. Specifically, the calculation of RED (t_{RED}) is performed on the GPU and the calculation of KDE (t_{KDE}) is performed on the CPU of the server. The two overall numbers in the parenthesis are detection time when there is no anomaly (thus no step 2) and there is an attack, respectively. We show that CloudShield can detect anomalies and identify the attacks and benign programs in 32 to 112 milliseconds if $w = 1$ or $w = 5$. Considering the attack usually takes seconds to succeed, e.g., several encryption operations for side-channel attacks, this latency can achieve our design goal of *real-time* detection. We suggest w=5 is sufficient.

Table 10: Detection latency (ms) versus window sizes.

(ms)	Anomaly Detection			Benign program/Attack detection			Overall (no anomaly, attack)
	t_B	t_{RED}	t_{KDE}	t_B	t_{RED}	t_{KDE}	
w=1	10.0	0.02	0.76	10.0	0.02	1.58	(10.78, 32.38)
w=5	50.0	0.02	0.76	50.0	0.02	1.58	(50.78, 112.38)
w=10	100.0	0.02	0.77	100.0	0.02	1.60	(100.79, 212.41)
w=50	500.0	0.02	0.78	500.0	0.02	1.62	(500.80, 1012.44)
w=100	1000.0	0.02	0.79	1000.0	0.02	1.65	(1000.81, 2012.48)

Performance overhead. We evaluate the performance overhead of CloudShield. We use the benchmarks in Table 3. We use completion time as the metric for ML training and MapReduce, average time per query for Database and Webserver, and processing time per frame for Stream Server. All the metrics are normalized to the cloud workload running without Cloudshield. Figure 8 reports the normalized metrics without CloudShield (blue solid) and with CloudShield running (orange dashed). Results are averaged over five runs). We see that CloudShield only introduces a small performance overhead. The maximum overhead is 6.3% for MapReduce and the minimum is 0.5% for database. In our experiments, we observe that on average CloudShield consumes 17.1% CPU time on the server.

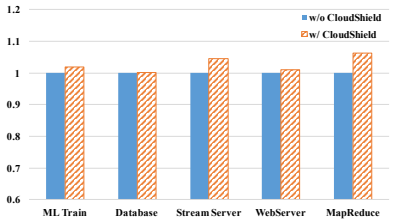

Figure 8: Performance overhead of CloudShield with different cloud workloads.

6.7 Discussion: Evasion Attacks

Previous work [32] revealed that attackers can effectively generate adversarial examples in the black-box setting to evade deep learning based intrusion detection systems. However, generating adversarial examples against our system is generally harder. First, our system monitors the **dynamic** behavior of a program. Generating dynamic adversarial examples that can both interact with other programs and escape detection in the black-box setting remains challenging. Second, the behavior markers monitored in our system are HPC measurements. As HPC measurements highly depend on the context of the executing environment, this introduces an extra obstacle for the attacker to construct the same execution environment when generating evasion adversarial examples. How to design and develop efficient evasive attacks and how to detect these attacks are worth exploring as future work.

7 PAST WORK

Recent work used deep learning for anomaly detection. Sucheta *et al.*[8] and Malhotra *et al.*[28] proposed LSTM for sequential anomaly detection. However, their methods only examined single prediction errors, rather than the distribution. We show that the reconstruction error distribution is more effective and robust in anomaly detection. He *et al.*[18] leveraged LSTM for anomaly detection in critical infrastructures. Different from this work, we detect not only anomalous behavior, but also which of these anomalous behavior are real or potential attacks (and which are benign or false alarms). Du *et al.*[12] updated anomaly detection model through unlearning. As stated in their work, unlearning may introduce a higher false-positive rate. In contrast, CloudShield significantly reduces false positives by distinguishing benign and malicious anomalies.

Another line of research detected specific attacks in the cloud. For example, Zhang *et al.*[40] developed CloudRadar for side-channel attack detection in the cloud using hardware performance counters. Guo *et al.*[15] detected cache side-channel leakage with symbolic

execution. Wang *et al.*[34] leveraged symbolic execution to detect speculative execution attacks. However, each of these detected a specific type of attack, unlike our work, which covers a broader scope of attacks, including zero-day attacks.

8 CONCLUSION

In this paper, we proposed CloudShield, a real-time anomaly and attack detection system for cloud computing. CloudShield leverages a single pre-trained deep learning model and leverages the reconstruction error distribution (RED) of hardware performance counters to model the normal behavior of a system using kernel density estimation (KDE). It is worth noting that CloudShield explicitly takes false-alarm reduction into account, a critical problem in anomaly detection systems. Once an anomaly is detected, CloudShield automatically distinguishes benign programs, known attacks, and zero-day attacks by investigating the different attack and benign program reconstruction error distributions, using the pre-trained model and kernel density estimators.

We evaluate CloudShield on various cloud workloads, attacks, and benign programs. Experimental results show that CloudShield can reliably detect various attacks in real-time with high accuracy and very low FNR and FPR. Moreover, experiments show that it can correctly identify unknown zero-day attacks and stealthy attacks that are running concurrently with benign programs. CloudShield achieves very low 0.3% FNR and 0.6% FPR for overall anomaly-attack detection. Especially, we find that CloudShield can detect the recently proposed speculative execution attacks in 32-112ms, and it can reduce false alarms by up to 99.0%.

REFERENCES

[1] 2018. https://msrc-blog.microsoft.com/2018/05/21/analysis-and-mitigation-of-speculative-store-bypass-cve-2018-3639/.
[2] 2019. https://github.com/wg/wrk.
[3] 2019. https://github.com/akopytov/sysbench.
[4] 2020. https://perf.wiki.kernel.org/index.php/Main_Page.
[5] 2020. https://hadoop.apache.org/docs/current/api/org/apache/hadoop/examples/terasort/package-summary.html.
[6] Joseph Bonneau and Ilya Mironov. 2006. Cache-collision timing attacks against AES. In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*.
[7] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. In *ACM SIGMOD International Conference on Management of Data (SIGKDD)*.
[8] Sucheta Chauhan and Lovekesh Vig. 2015. Anomaly detection in ECG time signals via deep long short-term memory networks. In *IEEE International Conference on Data Science and Advanced Analytics*.
[9] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).
[10] John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, and Salvatore Stolfo. 2013. On the feasibility of online malware detection with performance counters. *ACM SIGARCH Computer Architecture News* (2013).
[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina N. Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805* (2018).
[12] Min Du, Zhi Chen, Chang Liu, Rajvardhan Oak, and Dawn Song. 2019. Lifelong anomaly detection through unlearning. In *ACM Conference on Computer and Communications Security (CCS)*.
[13] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. 2016. Flush+Flush: a fast and stealthy cache attack. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*.
[14] David Gullasch, Endre Bangerter, and Stephan Krenn. 2011. Cache Games–Bringing Access-Based Cache Attacks on AES to Practice. In *IEEE Symposium on Security and Privacy (S&P)*.
[15] Shengjian Guo, Yueqi Chen, Peng Li, Yueqiang Cheng, Huibo Wang, Meng Wu, and Zhiqiang Zuo. 2020. Specusym: Speculative symbolic execution for cache timing leak detection. In *International Conference on Software Engineering*.
[16] Zecheng He, Guangyuan Hu, and Ruby B Lee. 2021. New Models for Understanding and Reasoning about Speculative Execution Attacks. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*.
[17] Zecheng He and Ruby B Lee. 2017. How secure is your cache against side-channel attacks?. In *Annual IEEE/ACM International Symposium on Microarchitecture*.
[18] Zecheng He, Aswin Raghavan, Guangyuan Hu, Sek Chai, and Ruby Lee. 2019. Power-Grid Controller Anomaly Detection with Enhanced Temporal Deep Learning. In *IEEE International Conference On Trust, Security And Privacy In Computing (TrustCom)*.
[19] John L Henning. 2006. SPEC CPU2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News* (2006).
[20] Geoffrey E Hinton and Sam Roweis. 2002. Stochastic neighbor embedding. *Advances in Neural Information Processing Systems (NeurIPS)* (2002).
[21] Harold Hotelling. [n. d.]. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology* ([n. d.]).
[22] Arijit Khan, Xifeng Yan, Shu Tao, and Nikos Anerousis. 2012. Workload characterization and prediction in the cloud: A multiple time series approach. In *IEEE Network Operations and Management Symposium*.
[23] Vladimir Kiriansky and Carl Waldspurger. 2018. Speculative buffer overflows: Attacks and defenses. *arXiv preprint arXiv:1807.03757* (2018).
[24] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. 2019. Spectre attacks: Exploiting speculative execution. In *IEEE Symposium on Security and Privacy (S&P)*.
[25] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, et al. 2018. Meltdown: Reading kernel memory from user space. In *USENIX Security Symposium*.
[26] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. 2015. Last-level cache side-channel attacks are practical. In *IEEE Symposium on Security and Privacy (S&P)*.
[27] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *IEEE International Conference on Data Mining (ICDM)*.
[28] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. 2015. Long short term memory networks for anomaly detection in time series. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*.
[29] Asit K Mishra, Joseph L Hellerstein, Walfredo Cirne, and Chita R Das. 2010. Towards characterizing cloud backend workloads: insights from google compute clusters. *ACM SIGMETRICS Performance Evaluation Review* (2010).
[30] Dag Arne Osvik, Adi Shamir, and Eran Tromer. 2006. Cache attacks and countermeasures: the case of AES. In *CryptographersâĂŹ Track at the RSA conference*.
[31] Meltem Ozsoy, Khaled N Khasawneh, Caleb Donovick, Iakov Gorelik, Nael Abu-Ghazaleh, and Dmitry Ponomarev. 2016. Hardware-based malware detection using low-level architectural features. *IEEE Trans. Comput.* (2016).
[32] Han Qiu, Tian Dong, Tianwei Zhang, Jialiang Lu, Gerard Memmi, and Meikang Qiu. 2020. Adversarial Attacks against Network Intrusion Detection in IoT Systems. *IEEE Internet of Things Journal* (2020).
[33] Bernhard Schölkopf, Robert C Williamson, Alex J Smola, John Shawe-Taylor, and John C Platt. 2000. Support vector method for novelty detection. In *Advances in Neural Information Processing Systems (NeurIPS)*.
[34] Guanhua Wang, Sudipta Chattopadhyay, Arnab Kumar Biswas, Tulika Mitra, and Abhik Roychoudhury. 2020. Kleespectre: Detecting information leakage through speculative cache attacks via symbolic execution. *ACM Transactions on Software Engineering and Methodology* (2020).
[35] Xueyang Wang and Ramesh Karri. 2014. Detecting kernel control-flow modifying Rootkits. In *Network Science and Cybersecurity*.
[36] Xueyang Wang, Charalambos Konstantinou, Michail Maniatakos, and Ramesh Karri. 2015. Confirm: Detecting firmware modifications in embedded systems using hardware performance counters. In *International Conference on Computer-Aided Design (ICCAD)*.
[37] Xinran Wang, Chi-Chun Pan, Peng Liu, and Sencun Zhu. 2008. Sigfree: A signature-free buffer overflow attack blocker. *IEEE Transactions on Dependable and Secure Computing* (2008).
[38] Ofir Weisse, Jo Van Bulck, Marina Minkin, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Raoul Strackx, Thomas F Wenisch, and Yuval Yarom. 2018. *Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution*. Technical Report.
[39] Yuval Yarom and Katrina Falkner. 2014. FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack. In *USENIX Security Symposium*.
[40] Tianwei Zhang, Yinqian Zhang, and Ruby B Lee. 2016. Cloudradar: A real-time side-channel attack detection system in clouds. In *International Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*.

A APPENDIX

A.1 Detect attacks concurrently running with benign programs

We consider a more difficult scenario where an attack is running concurrently with a certified benign program. We show that even if attack is spread out in an application, it can still be detected by its behavior via HPCs. We run three benign programs: gpg-rsa, gcc, and libquantum with the nine evaluated attacks in Table 11. First, on average, 99.9% of attacks when they are concurrently running with benign programs, are correctly recognized as attacks. A detailed analysis shows that, when an attack program is running concurrently with a benign program, 96.4% are identified as high-priority attacks (case 2), 3.1% are recognized as high-priority stealthy attacks (case 1), only 0.4% are classified as medium-priority zero-day attacks (case 4). These results show that CloudShield can still detect attacks even if they hide in benign programs.

Table 11: Results of benign programs/attacks detection (step 2), when attacks and benign programs run concurrently.

	Pred benign (Case 3)	Pred attack (Case 1,2,4)	Case 1 (stealthy attack)	Case 2 (attack)	Case 3 (benign)	Case 4 (0-day)
l1pp + gpg	0.000	1.000	0.241	0.746	0.000	0.013
l3pp + gpg	0.000	1.000	0.026	0.974	0.000	0.000
fr + gpg	0.000	1.000	0.117	0.883	0.000	0.000
ff + gpg	0.000	1.000	0.000	1.000	0.000	0.000
spectrev1 + gpg	0.000	1.000	0.000	0.999	0.000	0.000
spectrev2 + gpg	0.000	1.000	0.000	1.000	0.000	0.000
spectrev3 + gpg	0.000	1.000	0.000	1.000	0.000	0.000
spectrev4 + gpg	0.000	1.000	0.000	0.955	0.000	0.045
bufferoverflow + gpg	0.000	1.000	0.000	1.000	0.000	0.000
l1pp + gcc	0.000	1.000	0.000	1.000	0.000	0.000
l3pp + gcc	0.013	0.987	0.003	0.970	0.013	0.014
fr + gcc	0.000	1.000	0.044	0.956	0.000	0.000
ff + gcc	0.000	1.000	0.000	1.000	0.000	0.000
spectrev1 + gcc	0.000	1.000	0.017	0.983	0.000	0.000
spectrev2 + gcc	0.000	1.000	0.031	0.969	0.000	0.000
spectrev3 + gcc	0.000	1.000	0.000	0.973	0.000	0.027
spectrev4 + gcc	0.000	1.000	0.042	0.958	0.000	0.000
bufferoverflow + gcc	0.000	1.000	0.049	0.951	0.000	0.000
l1pp + libquantum	0.000	1.000	0.000	1.000	0.000	0.000
l3pp + libquantum	0.000	1.000	0.088	0.908	0.000	0.004
fr + libquantum	0.000	1.000	0.051	0.949	0.000	0.000
ff + libquantum	0.000	1.000	0.000	1.000	0.000	0.000
spectrev1 + libquantum	0.005	0.995	0.037	0.954	0.005	0.004
spectrev2 + ibquantum	0.000	1.000	0.000	1.000	0.000	0.000
spectrev3 + libquantum	0.000	1.000	0.000	0.992	0.000	0.008
spectrev4 + libquantum	0.000	1.000	0.035	0.965	0.000	0.000
bufferoverflow + libquantum	0.000	1.000	0.059	0.940	0.000	0.001
Average	**0.001**	**0.999**	**0.031**	**0.964**	**0.001**	**0.004**

A.2 Detailed discussion about system update

New types of cloud workloads. The commonly used cloud workloads in practice share common characteristics [22, 29], thus this re-training process only needs to be performed when a new type of cloud workload is added. This kind of update is not frequent. Moreover, the whole update procedure can be performed during low usage time. CloudShield loads the updated models and detectors to the processor cores.

New certified benign programs. Update of new certified benign programs is relatively lightweight, compared to cloud workload update, because the pre-trained model does not need to change. CloudShield then executes the new benign program, collects its behavior measurements in a clean execution environment, and calculate the REDs. As shown in the formula of KDE estimator (Eq. 4), the estimated likelihood $\hat{f}(x)$ is summed over all reference prediction errors x_i. Therefore, CloudShield only needs to append the new prediction errors of the new certified program to the existing prediction errors to form the new RED.

New discovered attacks. This follows the same procedure of updating certified benign programs. It is also lightweight as the pretrained model does not need to be updated.