CloudRadar: A Real-Time Side-Channel Attack Detection System in Clouds

Tianwei Zhang[1(✉)], Yinqian Zhang[2], and Ruby B. Lee[1]

[1] Princeton University, Princeton, NJ, USA
{tianweiz,rblee}@princeton.edu
[2] The Ohio State University, Columbus, OH, USA
yinqian@cse.ohio-state.edu

Abstract. We present *CloudRadar*, a system to detect, and hence mitigate, cache-based side-channel attacks in multi-tenant cloud systems. *CloudRadar* operates by correlating two events: first, it exploits signature-based detection to identify when the protected virtual machine (VM) executes a cryptographic application; at the same time, it uses anomaly-based detection techniques to monitor the co-located VMs to identify abnormal cache behaviors that are typical during cache-based side-channel attacks. We show that correlation in the occurrence of these two events offer strong evidence of side-channel attacks. Compared to other work on side-channel defenses, *CloudRadar* has the following advantages: first, *CloudRadar* focuses on the root causes of cache-based side-channel attacks and hence is hard to evade using metamorphic attack code, while maintaining a low false positive rate. Second, *CloudRadar* is designed as a lightweight patch to existing cloud systems, which does not require new hardware support, or any hypervisor, operating system, application modifications. Third, *CloudRadar* provides real-time protection and can detect side-channel attacks within the order of milliseconds. We demonstrate a prototype implementation of *CloudRadar* in the OpenStack cloud framework. Our evaluation suggests *CloudRadar* achieves negligible performance overhead with high detection accuracy.

Keywords: Attack detection · Side-channel attacks · Performance counters · Cloud computing · Mitigation

1 Introduction

Infrastructure-as-a-Service (IaaS) cloud systems usually adopt the multi-tenancy feature to maximize resource utilization by consolidating virtual machines (VMs) leased by different tenants on the same physical machine. Virtualization technology is used to provide strong resource isolation between different VMs so each VM's memory content is not accessible to other co-tenant VMs. However, confidentiality breaches due to cross-VM side-channel attacks become a major concern. These attacks often operate on shared hardware resources and extract

F. Monrose et al. (Eds.): RAID 2016, LNCS 9854, pp. 118–140, 2016.
DOI: 10.1007/978-3-319-45719-2_6

sensitive information, such as cryptographic keys, by making inferences on the observed side-channel events due to resource sharing. CPU caches are popular attack surfaces that lead to cross-VM side-channel attacks. Several prior work have shown the possibilities of cross-VM secret leakage via different levels of CPU caches [10,14,15,21,42,45,46].

Mitigating side-channel attacks in clouds is challenging. Past work on defeating side-channel attacks have some practical drawbacks: they mostly require significant changes to the hardware [6,20,39,40], hypervisors [17,18,31,33,35,48] or guest OSes [48], making them impractical to be deployed in current cloud datacenters. Other work have proposed to mitigate these attacks in cloud contexts by periodic VM migrations to reduce the co-location possibility between victim VMs and potential malicious VMs [25,47]. These heavy-weight approaches cannot effectively prevent side-channel leakage unless performed very frequently, making them less practical as VM co-location takes on the order of minutes [34] while side-channel attacks can be done on the order of milliseconds [21,42].

In this paper, we propose to detect side-channel attacks as they occur and prevent information leakage by triggering VM migration upon attack detection. However, side-channel attack detection is non-trivial. To do so, we must overcome several technical challenges in the application of traditional detection techniques, like signature-based detection and anomaly-based detection, to side-channel attacks. Signature-based side-channel detection exploits pattern recognition to detect known attack methods [4,5,13]. While low in false negatives for existing attacks, it fails to recognize new attacks; anomaly-based detection flags behaviors that deviate significantly from the established normal behaviors as attacks, which can potentially identify new attacks in addition to known ones. However, differentiating side-channel attacks from normal applications is difficult as these attacks just perform normal memory accesses which resemble some memory intensive applications.

To overcome these challenges, we design *CloudRadar*, a real-time system to detect the existence of cross-VM side-channel attacks in clouds. There are two key ideas behind *CloudRadar*: first, *the victim has unique micro-architectural behaviors when executing cryptographic applications that need protection from side-channel attacks.* So the cloud provider is able to identify the occurrence of such events using a signature-based detection method. Second, *the attacker VM creates an anomalous cache behavior when it is stealing information from the victim.* Such anomaly is inherent in all side-channel attacks due to the intentional cache contention with the victim to induce side-channel observations. By correlating these two types of events, *CloudRadar* is able to detect the stealthy cache side-channel attacks with high fidelity.

We implement *CloudRadar* as a lightweight extension to the virtual machine monitors. Specially, it (1) utilizes the existing host system facilities to collect micro-architectural features from hardware performance counters that are available in all modern commodity processors, and (2) non-intrusively interacts with the existing virtualization framework to monitor the VM's cache activities while

inducing little performance penalty. Our evaluations show that it effectively detects side-channel attacks with high true positives and low false positives.

Compared to past work, *CloudRadar* has several advantages. First, *CloudRadar* focuses on the root causes of cache-based side-channel attacks and hence is hard to evade using different attack code, while maintaining a low false positive rate. Our approach is able to detect different types of side-channel attacks and their variants with a simple method. Second, *CloudRadar* is designed as a lightweight patch to existing cloud systems, which does not require new hardware support or hypervisor/OS modifications. Therefore *CloudRadar* can be immediately integrated into modern cloud fabric without making drastic changes to the underlying infrastructure. Third, *CloudRadar* exploits hardware performance counters to monitor VM activities, which detects side-channel attacks within the order of milliseconds with negligible performance overhead. Finally, *CloudRadar* requires no changes to the guest VM or the applications running in it, and thus is transparent to cloud customers.

To summarize, *CloudRadar* achieves the following contributions:

- The first approach to detect cache side-channel attacks using techniques that combine both signature-based and anomaly-based detection techniques.
- A novel technique to identify the execution of cryptographic applications, which are of interest in its own right.
- A non-intrusive system design that requires no changes to the hardware, hypervisor and guest VM and applications, which shows potential of immediate adoption in modern clouds.
- Full prototype implementation and extensive evaluation of the proposed approach and detection techniques.

The rest of this paper is organized as follows: Sect. 2 presents the background of cache side-channel attacks and defenses, and other detection systems based on performance counters. Section 3 presents the design challenges and system overview. Section 4 discusses the signature-based methods to detect cryptographic applications. Section 5 shows the anomaly-based method to detect side-channel activities. Section 6 presents the architecture and implementation of *CloudRadar*. Section 7 evaluates its performance and security. We discuss the limitations of *CloudRadar* and potential evasive attacks against it in Sect. 8. Section 9 concludes.

2 Background and Related Work

2.1 Cache Side-Channel Attacks

In cache-based side-channel attacks, the adversary exfiltrates sensitive information from the victim via shared CPU caches. The sensitive information are usually associated with cryptographic operations (*e.g.*, signing or decryption), but may also be extended to other applications [46]. Such sensitive information are

leaked through secret-dependent control flows or data flows that lead to attacker-observable cache use patterns. The adversary, on the other hand, may exploit several techniques to manipulate data in the shared cache to deduce the victim's cache use patterns, and thereby make inference on the sensitive information that dictates these patterns. Two cache manipulation techniques are well-known for side-channel attacks:

Prime-Probe Attacks: The adversary allocates an array of cacheline-sized, cacheline-aligned memory blocks so that these memory blocks can exactly fill up a set of targeted cache sets. Then the adversary repeatedly performs two attack stages: in the PRIME stage, the adversary reads each memory block in the array to evict all the victim's data in these cache sets. The adversary waits for some time interval before performing the PROBE stage, in which he reads each memory block in the array again, and measures the time of memory accesses. Longer access time indicates one or more cache misses, which means this cache set has been accessed by the victim between the PRIME and PROBE stages. The adversary will repeat these two steps for significant amount of times to collect traces that, hopefully, overlap with the victim's execution of cryptographic operations, for offline analysis. This technique was first proposed by Percival [27], and then applied to the cloud environment in [14,21,28,45].

Flush-Reload Attacks: This type of attacks assumes identical memory pages can be shared among different VMs, so that the adversary and victim VMs may share the same pages containing cryptographic code or data. The adversary carefully selects a set of cacheline-sized, -aligned memory blocks from these shared pages. Then he also conducts two stages repeatedly: in the FLUSH stage, the adversary flushes the selected blocks out of the entire cache hierarchy (*e.g.*, using the *clflush* instruction). Then it waits for a fixed interval in which the victim might issue the critical instructions and fetch them back to the caches. In the RELOAD stage, the adversary reloads these memory blocks into the caches and measures the access time. A short access time for one memory block indicates a cache hit, so this block has been accessed by the victim during the interval. By repeating these two stages the adversary can obtain traces of the victim's memory accesses and deduce the confidential data. This FLUSH-RELOAD technique was first proposed in [11], and further demonstrated in different virtualized platforms with different variants [9,10,15,46].

2.2 Defenses Against Side-Channel Attacks

Previous studies propose to defeat cache-based Side-channel attacks in one of these three ways:

– **Partitioning caches:** One straightforward approach is to prevent the cache sharing by dividing the cache into different zones by sers or ways for different VMs. This can be achieved by hardware [6,19,40] or software methods [17,31].

- **Randomization:** This idea is to add randomization to the attacker's measurements, making it hard for him to get accurate information based on his observations. This includes random memory-to-cache mappings [39,40], cache prefetches [20], timers [18,35] and cache states [48].
- **Avoiding co-location:** New VM placement policies were designed [1,12] to reduce the co-location probability between victim and attacker VMs. Zhang et al. [47] and Moon et al. [25] frequently migrated the VMs to add difficulty of VM co-location for the attackers.

These approaches, when applied in the cloud setting, require significant modification of computing infrastructure, and thus are less attractive to cloud providers for practical adoption. In our study, we aim to build atop existing cloud framework a lightweight side-channel attack detection system to detect, and then mitigate, the attacks as they take place, while doing so without modifying guest OS, hypervisor or hardware.

2.3 Intrusion Detection Using Hardware Performance Counters

Hardware performance counters are a set of special-purpose registers built into ×86 (*e.g.*, Intel and AMD) and ARM processors. They work along with event selectors which specify certain hardware events, and update a counter after a hardware event occurs. Most modern processors provide a Performance Monitor Unit (PMU) that enables applications to control performance counters. One of the basic working modes of PMUs is the interrupt-based mode. Under this working mode, an interrupt is generated when the occurrences of a given event exceed a predefined threshold or a predefined amount of time has elapsed. Therefore, it makes both event-based sampling and time-based sampling possible.

Performance counters were originally designed for software debugging and system performance tuning. Recently, researchers exploited performance counters to detect security breaches and vulnerabilities [2,5,23,32,36,37,41,43]. The intuition is that the performance counters can reveal programs' execution characteristics, which can further reflect the programs' security states. Besides, performance counter detection introduces negligible performance overhead to the programs. Related to ours were signature-based side-channel attack detection using performance counters [4,5,13], which, unfortunately, could be easily evaded by smarter attackers by slightly changing cache probing pattern.

3 Design Challenges and Overview

In this paper, we explore an oft-discussed, but never successfully implemented, idea: exploiting hardware performance counters available in commodity processors to detect side-channel attacks that abuse processor caches. We first systematically explore the design challenges and then sketch our design of *CloudRadar*.

Threat Model and Assumptions. We focus on cross-VM side-channel threats in public IaaS clouds based on Last Level Caches (LLC) that are shared between processor cores. We assume the adversary is a legitimate user of the cloud service who is able to launch VMs in the cloud and has complete control of his VMs. We further assume the attacker is able to co-locate one of his VMs on the same server as the victim VM, and the two VMs will share the same processor package, thus the LLC, with non-negligible probability. We consider both Prime-Probe side-channel attacks and Flush-Reload side-channel attacks, which represent all known LLC side channels in modern computer systems.

3.1 Design Challenges

Signature-Based Detection. Signature-based detection approaches are widely used techniques in detecting network intrusion and malware, by comparing monitored application or network characteristics with pre-identified attack signatures. Similarly, to detect side-channel attacks, signatures of side-channel attacks must be generated from all known side-channel attack techniques and used to compare with events collected from production systems. Prior work [4,5] has preliminarily explored such ideas. Particularly, Demme et al. demonstrated in a simplified experiment setting that classification algorithms could successfully differentiate normal programs from Prime-Probe attack programs. The advantage of this approach is that they have high true positive rate in detecting known attacks. However, such detection method is very fragile and easy to evade by clever attackers. It also fails to recognize unknown attacks even with only subtle changes from existing ones. For instance, the attacker can change the memory access pattern (*e.g.*, sequential order, access frequency) in a Prime-Probe attack to evade signature-based detection.

Anomaly-Based Detection. In anomaly-based detection, the normal behaviors of benign applications are modeled and any substantial deviation from such models are detected as attacks. To detect side-channel attacks using such techniques, one can build models for benign application behaviors. Then for each VM to be monitored, we check if its behaviors conform to the models in the database. Compared to signature-based detection, anomaly-based detection can potentially identify "zero-day" attacks in addition to known ones. However, the difficulty of applying the anomaly-based approach to side-channel attacks stems from the challenge of precisely modeling benign application activities using performance counters. Cache side-channel attacks resemble benign memory intensive applications (*e.g.*, memory streaming application [24]), and therefore they are difficult to be differentiated using only hardware performance counters. False positive or false negative rates can be extremely high due to imprecise application behavior modeling. We are not aware of successful side-channel detection methods that are based on anomaly detection.

3.2 Design Overview

We design a side-channel attack detection system, *CloudRadar*, that combines both anomaly-based and signature-based techniques. The only features used by *CloudRadar* are hardware event values read from the performance counters available in commercial processors. The key insight that motivates *CloudRadar* is derived from our prior research in side-channel attacks: in cache side-channel attacks, to effectively exfiltrate secret information from the victim's sensitive execution, the attacker needs to repeatedly conduct side-channel activities (*e.g.*, Prime-Probe or Flush-Reload) and deduce his own cache uses based on the execution time of his own memory activities. Then he can make inferences on the victim's cache use pattern by looking at the statistics of his use of caches (*e.g.*, cache hits and cache misses). As such, the attacker's cache use patterns must be different when the victim executes sensitive operations so that the attacker can differentiate them in his own analysis. Our intuition is that *if such distinction can be detected by the attacker using timing channels, it can be detected by the cloud provider using performance counters.*

We design *CloudRadar* to monitor all VMs running on a cloud server and collect their cache use patterns using hardware performance counters. Once anomaly in cache use patterns are detected by *CloudRadar*, these anomalies will be correlated with the sensitive operations (usually cryptographic operations) in the co-located protected VM (*i.e.*, VMs owned by customers paying for such services). Strong correlation will serve as a good indicator of cache-based side-channel attacks.

Two key *technical challenges* in our design are (1) identifying the execution of the protected VM's sensitive operations without asking the customers to modify their applications and (2) detecting untrusted VM's abnormal cache use patterns. We aim to achieve both by using only values read from performance counters. To do so, we *first* propose to use signature-based techniques to detect sensitive applications of the protected VM, because they are conducted by honest parties and will not attempt to evade detection intentionally—a perfect target of signature-based detection techniques. *Second*, we propose to use anomaly-based detection techniques to detect abnormal cache patterns due to side-channel activities, as they are expected to vary due to different attack techniques and intensity. As side-channel attack detection is done via correlation with sensitive operations, false positives that are common challenges to anomaly detection techniques can be ruled out. We will highlight our design of these two components in Sects. 4 and 5.

4 Signature Detection of Cryptographic Applications

As sensitive operations that are targeted by side-channel attacks are usually cryptographic operations, we consider detection of cryptographic applications in this paper. Our working hypothesis here is that all cryptographic applications have unique signatures that can be easily identified by performance counters. In this section, we validate our hypothesis by a set of preliminary experiments.

4.1 Cryptographic Signature Generation

To generate signatures for detecting cryptographic applications, we need to select a proper hardware performance feature that uniquely characterizes a certain execution phase [30] of such applications.

Feature Selection. Modern processors allow a large number of events to be measured and reported by performance counters. The signature generated from a proper hardware event should satisfy two requirements: (1) *uniqueness*: the signatures of different applications should be highly distinguishable; (2) *repeatability*: the signature of a cryptographic application should be identical each time it is generated, regardless of the platform's configurations and the inputs.

We consider different events from three main categories: CPU events, cache events and kernel software events. We use the Fisher Score [7] to test the *repeatability* and *uniqueness* of these events in identifying cryptographic applications. Fisher Score is one of the most widely used methods to select features quickly. It finds the optimal feature so that the distances between data points in the feature space of different classes are maximized, while the distances between data points in the same class are minimized.

To test the *uniqueness* of an event, we use performance counters to measure the number of this event every 100 µs during the execution of six representative cryptographic applications (*i.e.*, asymmetric cryptography: ElGamal and DSA from GnuPG; symmetric cryptography: AES and 3DES from OpenSSL; hash: HMAC from OpenSSL and SHA512 from GnuPG). We select 10 consecutive counter values (collected from $10 \times 100\,\mu s$) from the beginning of each application to form a timing sequence as one training data point. We repeat this 100 times for each cryptographic application. For each hardware event we considered, we calculate the Fisher Score using 600 training data points from the six cryptographic applications to test the uniqueness of this event in distinguishing different applications. Table 1 (Inter-class F-Score column) shows the results. Note a larger inter-class F-Score indicates a better *uniqueness* of this event. We can see some CPU events (instructions, branches and mispredicted branch instructions) and cache events (L1I load misses) are better candidates for signature generation. They vary significantly for different cryptographic applications. The events that rarely happen during the cryptographic execution (*e.g.*, context switches and page faults), or remain identical for different cryptographic applications (*e.g.*, CPU cycles or clock) fail to satisfy the *uniqueness* requirement.

To test the *repeatability* of an event, we repeat the above experiments on three servers with different hardware and software configurations. For each cryptographic application, we calculate the Fisher Score from 300 training data points collected from three servers. Table 1 (Intra-class F-Score column) shows the average Fisher Score of the six cryptographic programs. A smaller Intra-class F-Score indicates the signature with this event is more repeatable. We are able to find some events with good repeatability (*e.g.*, instructions, branches and mispredicted branch instructions).

Table 1. Fisher scores for different events.

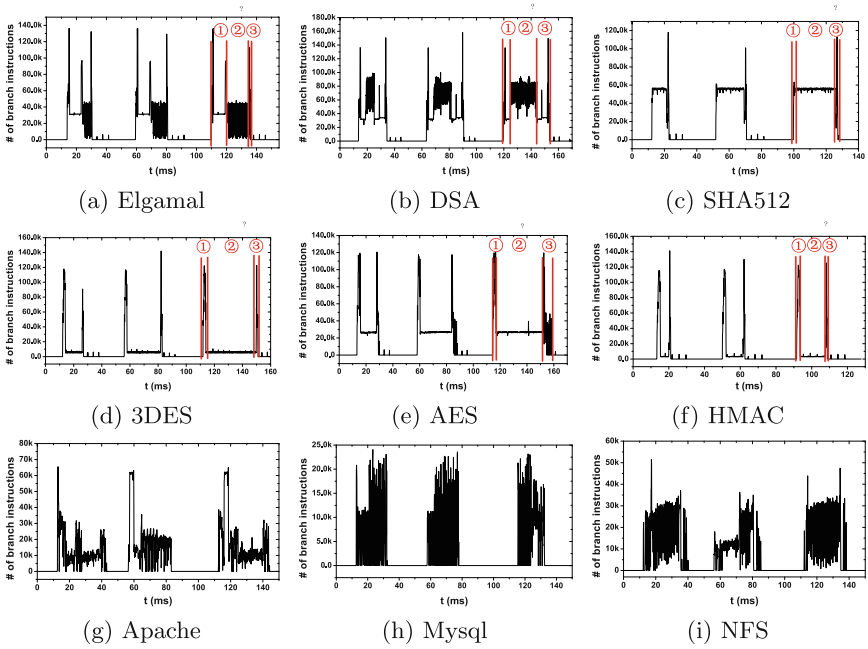
Category	Events	Inter-class F-Score	Intra-class F-Score
CPU events	Instructions	1.49	0.13
	Branch instructions	1.55	0.14
	Mispredicted branch instructions	1.11	0.15
	CPU cycles	0.01	0.30
Cache events	L1D load accesses	0.37	0.72
	L1D load misses	0.69	0.42
	L1I load misses	1.14	0.20
	LLC load accesses	0.79	0.31
	LLC load misses	0.05	0.36
	iTLB load accesses	0.55	0.27
	iTLB load misses	0.23	0.21
	dTLB load accesses	0.22	0.63
	dTLB load misses	0.36	0.62
Software events	Context switches	0.00	0.00
	Page faults	0.00	0.00
	CPU clock	0.01	0.50

Based on the inter-class and intra-class Fisher Scores, we can choose the features with both good uniqueness and repeatability for signature matching. For instance, we can use *instructions* and *branch instructions* to conduct multi-feature classification. Further evaluations in Sect. 7 show one single feature (*i.e.*, *branch instructions*) is already enough to give good accuracy. So we will collect the number of *branch instructions* as the feature to generate signatures in the following sections.

Phase Selection. It has been shown in prior studies that programs run in different phases [30]. Therefore, another question we need to solve is which phase of the cryptographic application we should use to generate the signature. The selected phase should be able to distinguish cryptographic applications from non-cryptographic applications. It should also be independent of the inputs.

We conducted the following experiments: we ran the same six cryptographic applications as above. For each cryptographic application, the cryptographic keys and input message (for signing or encryption) are randomly chosen each time the applications are executed. We exploit the performance counters to record the number of *branch instructions* taking place in the program within 100 µs windows. Figure 1 shows the profiling results for each cryptographic application. For comparison, we also show the profiling results for three non-cryptographic applications: Apache, Mysql and the Network File System (NFS).

We observe that the cryptographic applications have different behaviors from the non-cryptographic ones. Each cryptographic application exhibits three distinguishable stages, labeled in Fig. 1. (1) The first stage initializes the program



(a) Elgamal (b) DSA (c) SHA512

(d) 3DES (e) AES (f) HMAC

(g) Apache (h) Mysql (i) NFS

Fig. 1. Signatures of different applications based on the number of branches

and variables. Specifically, it analyzes the application's parameters, allocates buffers for the input and output messages, retrieves keys from passphrase or salts, and sets up the cipher context. This stage does not depend on the inputs. (2) The second stage computes the cryptographic operations (*e.g.*, multiply or square operations, checking lookup tables, *etc.*), the characteristics of which are input dependent: the duration of this stage is linearly related to the length of the plaintext/ciphertext, and the pattern depends on the values of the cryptographic key and the plaintext/ciphertext blocks. (3) The last stage ends the application, frees the memory buffer and reports the results. We chose the first stage as the signature to represent a crypto application, because it is input independent. The Fisher Score in Table 1 were also generated for this stage.

4.2 Cryptographic Application Detection

To detect the execution of the sensitive applications, *CloudRadar* only requires the customers to provide the signature generated offline using performance counters (not necessarily on the same hardware) or simply the executables for the service provider to generate the signature. At runtime, *CloudRadar* keeps monitoring the protected VM using the same set of performance counters. It then compares the data points collected at runtime with the signature of the cryptographic application. If a signature match is found, *CloudRadar* will assume the

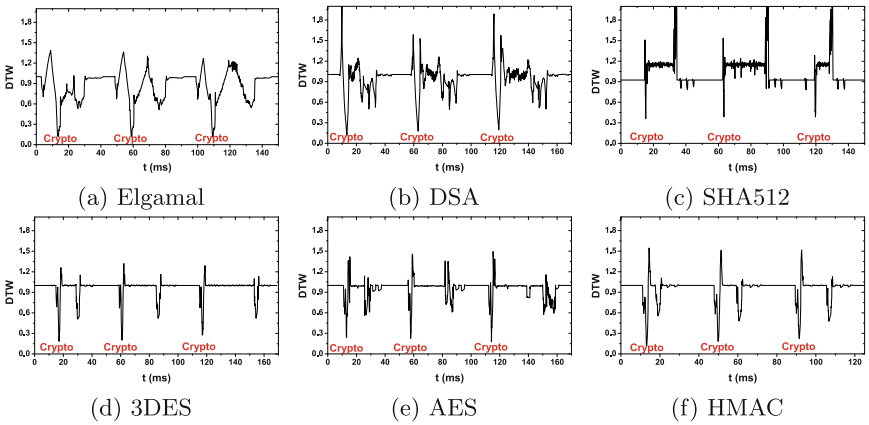


Fig. 2. DTW distances of different cryptographic programs. The lowest distance indicates a signature match.

cryptographic application is being executed by the protected VM (In fact, our evaluation in Sect. 7 shows high fidelity of this approach).

Because the cryptographic signatures and runtime measurements are temporal sequences of performance counter values, we cast the signature detection problem as a time series analysis problem: *i.e.*, measuring the similarity between the two sequences that represent the signature and the runtime measurement, respectively. We adopt the Dynamic Time Warping (DTW) algorithm [29] to calculate the distance between the two sequences. DTW is able to measure the similarity between temporal sequences which may vary in speed: it tries different alignments between these sequences and finds the optimal one that has the shortest distance. This distance is called the DTW distance. We chose the DTW algorithm because the runtime sequence may be slightly stretched or shrunk due to the difference of the computing environment (*e.g.*, CPU models, running speed, interruption, *etc.*). DTW is powerful enough to find the similarity between two temporal sequences even with distortion.

We normalize the DTW distance to the magnitude of the signature sequence, which is used as the metric for pattern matching. Figure 2 shows the normalized DTW distance of different cryptographic programs. We observe that occurrence of cryptographic programs yields very small DTW distances, which indicates a signature match. We defer a more systematic evaluation of the signature-based cryptographic program detection technique to Sect. 7.

5 Anomaly Detection of Side-Channel Activities

The cache use patterns that *CloudRadar* monitors for anomaly detection are characterized by the *cache hit* count and the *cache miss* count measured by the performance counters: In Prime-Probe side-channel attacks, the attacker

PROBEs certain cache sets and measures if there are cache miss via timing the accesses to this set after the victim executes. It is expected that cache misses will be higher than normal when the protected VM executes the cryptographic operations, since cache misses will be the tell-tale signal for the attacker to detect these operations in the first place. In FLUSH-RELOAD side-channel attacks, the attacker RELOADs certain cache lines and tries to detect a cache hit. Cache hits should occur more frequently during the protected VM's sensitive operations.

To validate this hypothesis, we conducted a set of experiments to show that abnormal cache activities in the untrusted VM can be correlated with the protected VM's sensitive operations. We first consider a PRIME-PROBE attack against the ElGamal cipher [21]. Figure 3 shows the DTW distance (low distance indicates a signature match) between the runtime sequence and the signature sequence observed on the protected VM (top figure), correlates with the attacker VM's high cache miss counts (bottom figure). We next consider a FLUSH-RELOAD attack against the RSA cipher [42]. Figure 4 shows the low DTW distance of the protected VM correlates with the high cache hit counts of the attacker VM. We align the top figures and the bottom figures according to timestamps. Strong correlation can be observed in both set of experiments, which suggest that this method can be used for side-channel attack detection.

To describe our detection algorithm more precisely, when *CloudRadar* detects that the victim VM starts executing crypto applications (a low DTW distance), two short sub-sequences are selected from the entire monitored runtime sequences in the untrusted VMs: $\mathbb{S}$, data points of size w before the DTW distance reaches its minimum, and $\mathbb{S}'$, data points of size w after the minimum points of DTW distance, where w is a parameter of the detection system. If *CloudRadar* detects that the difference between any value in $\mathbb{S}'$ and any value in $\mathbb{S}$ is larger than a pre-determined threshold T, *CloudRadar* will raise an alarm of a possible side-channel attack. This rule can be formally expressed in Eq. 1. We will further evaluate this side-channel detection method in Sect. 7.

$$\text{Alarm: } v' - v > T, \quad \forall v \in \mathbb{S},\ v' \in \mathbb{S}' \tag{1}$$

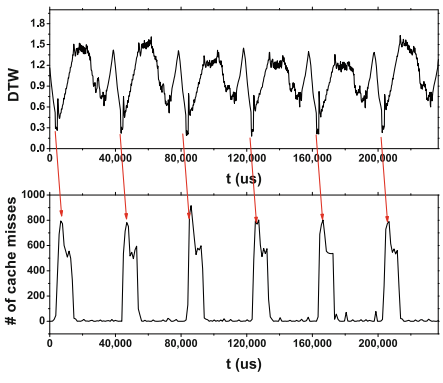


Fig. 3. PRIME-PROBE attack

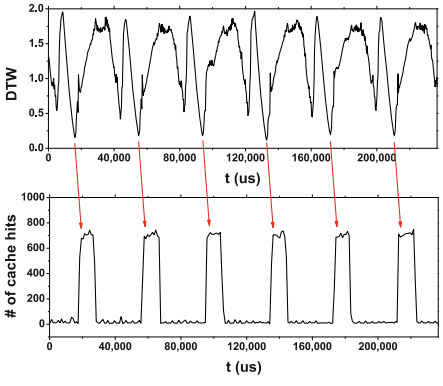


Fig. 4. FLUSH-RELOAD attack

6 Implementation

6.1 System Architecture Overview

CloudRadar is provided by the cloud operator as a security service to the customers who are willing to pay extra cost for better security, as in Security-on-Demand cloud frameworks [16,44]. Figure 5 shows the architecture of *CloudRadar*, and the workflow of detecting side-channel attacks. We implement *CloudRadar* in the opensource cloud software OpenStack platform. Two types of servers, the Cloud Controller and regular Cloud Servers, are relevant to our discussion.

The Cloud Controller is a dedicated server to manage the provided security services and coordinate the interaction between service users (cloud customers paying to use the side-channel detection service) and the Cloud Servers. The `Signature Database` is used to store signatures of crypto programs. The Controller Server is built upon the OpenStack Nova module. We modified the Nova API to enable the customers to request for the side-channel detection services.

CloudRadar's functionality within a Cloud Server is tightly integrated with the host OS. As shown in Fig. 5, *CloudRadar* consists of three modules, with each one running on a dedicated core. The `Victim Monitor` is responsible for collecting the protected VM's runtime events, which will be fed to `Signature Detector` to detect the cryptographic programs using our signature-based technique; The `Attacker Monitor` is responsible for collecting cache activities of the other VMs, using anomaly-based detection approach to identify side-channel attackers. We used the Linux *perf_event* kernel API for the PMU to manage the performance counters, therefore no change is needed to the hypervisor itself.

6.2 Operations

CloudRadar includes four steps, as shown in Fig. 5 with different paths. Each step is described below:

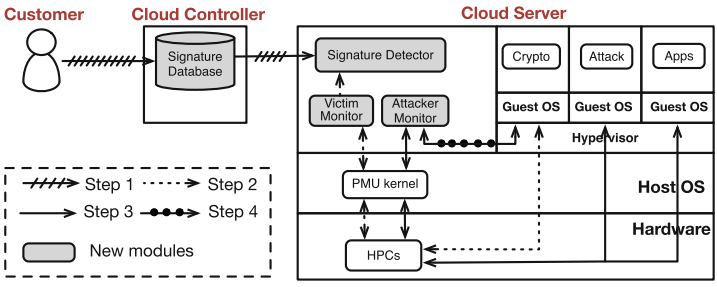


Fig. 5. Architecture overview of *CloudRadar*

Step 1: Generating Cryptographic Signature. In this step, the customer who seeks side-channel detection services for his protected VM can indicate to the Cloud Controller what sensitive applications to be protected, by providing the signatures generated offline using performance counters (not necessarily on the same hardware) or simply the executables. Then the Cloud Controller will run these crypto programs on a dedicated server with the same configuration as the Cloud Server that hosts the protected VM, and use performance counters to generate the signatures for the customer. The signatures will be stored in the `Signature Database` for future reference. They will also be sent to the Cloud Server that hosts this VM.

Step 2: Detecting Cryptographic Applications. This step takes place at runtime. In this step, the `Victim Monitor` monitors the protected VM using performance counters. It periodically (*e.g.*, every 100 µs) records the event counts (*e.g.*, branch instructions) as a time sequence, while the `Signature Detector` keeps comparing the most recent window of data points in the sequence with the signature. If a signature match is found, the `Signature Detector` can identify the protected VM is performing a cryptographic application, and signal this result to the `Attacker Monitor`.

Step 3: Monitoring Cache Activities. This step happens concurrently with Step 2. The `Attacker Monitor` exploits performance counters to monitor all untrusted VMs simultaneously. One challenge is that not enough performance counters are available on the servers to monitor all VMs, if this number is large: most of the Intel and AMD processors support up to six counters, and the number of counters does not scale with the number of cores. So when there are a lot of VMs on the server, the `Attacker Monitor` cannot monitor them concurrently.

To solve this problem, we use a time-domain multiplexing method: `Attacker Monitor` identifies active vCPUs that share LLC with the protected VM as the *monitored* vCPUs, and then measures each of them in turn. Specifically, in each period, the `Attacker Monitor` uses a kernel module to check the state and CPU affinity of each vCPU of each VM from its *task_struct* in the kernel. The `Attacker Monitor` marks the vCPUs in the *running* state that are sharing the same LLC with the protected VM as *monitored*. Then it sets up performance counters to measure each *monitored* vCPU's cache misses and hits in turn. When the `Attacker Monitor` is notified that a cryptographic application is happening in the protected VM, it will compare each *monitored* vCPU's cache misses and hits before and during the cryptographic application, as specified in Sect. 5. If one vCPU has an abrupt increase in the number of cache misses or hits during the cryptographic application, the `Attacker Monitor` will flag an alarm.

Step 4: Eliminating Side Channels. Once the `Attack Monitor` notices that one co-tenant VM has abnormal cache behavior exactly when the protected VM executes cryptographic applications, it will raise alarm for side-channel attacks.

It will migrate this malicious VM to a different processor socket which does not share the Last Level Cache (LLC), or another cloud server (*i.e.*, via VM migration [25,47], to cut off the cache side channels. In addition, the Cloud Controller will report this incident to the cloud provider for further processing, such as shut down the malicious VM or eventually block the attacker's account.

7 Evaluation

We used four servers to evaluate the security and performance of *CloudRadar*. A Dell R210II Server (equipped with one quad-core, 3.30 GHZ, Intel Xeon E3-1230v2 processor with 8 GB LLC) is configured as the Controller Server. Two Dell PowerEdge R720 Servers are deployed as the host cloud servers: one is equipped with one eight-core, 2.90 GHz Intel Xeon E5-2690 processor with 20 GB LLC; one is equipped with two six-core, 2.90 GHz Intel Xeon E5-2667 processors with 15 GB LLC. We also use another Dell 210II server as the client machine outside of the cloud system to communicate with cloud applications. Each VM in our experiments has one virtual CPU, 4 GB memory and 30 GB disk size. We choose Ubuntu 14.04 Linux, with 3.13 kernel as the guest OS.

7.1 Detection Accuracy

We measure the detection accuracy of cryptographic signature detection and cache anomaly detection.

Accuracy of Cryptographic Operation Detection. To detect a cryptographic operation, we used the branch instruction counts as the signature. We consider the detection of a cryptographic application as a binary classification, and measure its true positive rate and false positive rate. True positive happens when a cryptographic application is correctly identified as such. We used the same six cryptographic applications from Sect. 4.1. *CloudRadar* first generates a signature for each application. In the detection phase, the victim VM generates a random memory block and feeds it to the crypto application. We run the experiment 100 times, and measure the number of times *CloudRadar* can correctly identify the cryptographic under different thresholds. False positive is defined as non-cryptographic applications identified as cryptographic. We select 30 common linux commands and utilities [26] which do not contain cryptographic operations. In each experiment the victim VM run these commands in a random order. We repeated the experiment 100 times and measure the number of times false positives take place under different thresholds. We plot the ROC (Receiver Operating Characteristic) curves to show the relations between the true positive rate and false positive rate.

We explored the effect of changing performance counter sampling granularities (*i.e.*, period with which performance counter value is taken) on detection accuracy. We choose two different sampling granularities: 100 µs and 1 ms. Figure 6 shows the ROC curves of the six cryptographic applications under these

two granularities. From this figure we can see 100 µs gives better accuracy than 1 ms: *CloudRadar* can achieve close to 100 % true positive rate with zero false positive rate when the DTW threshold is set between 0.3 and 0.4. For 1 ms, Elgamal and DSA application can be detected with less accuracy, while SHA512, AES, HMAC and 3DES cannot be differentiated from non-cryptographic applications with reasonable false positive and false negative at the same time.

The optimal sampling granularity depends on the length of the cryptographic application's initialization stage: if the sampling period is much shorter than the initialization stage, the signature will contain more data points, thus yielding more accurate results. In our experiments, the initialization stages of Elgamal, DSA, SHA512, AES, HMAC and 3DES last for 10 ms, 5 ms, 1.6 ms, 2 ms, 2 ms and 2 ms respectively. So a granularity of 100 µs can give good results for all the six applications, while 1 ms granularity performs worse, especially for SHA512, AES, HMAC and 3DES whose signatures only contain two data points.

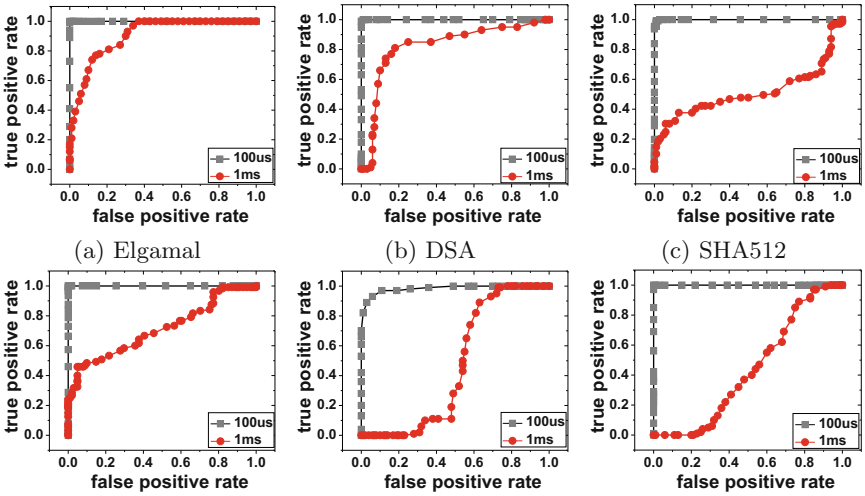
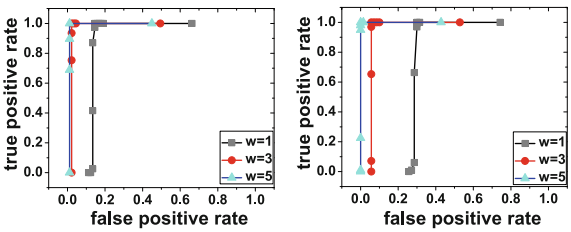


Fig. 6. ROC curve of crypto detection under two sampling intervals.

Accuracy of Cache Side-Channel Attack Detection. We measure the true positive rate and false positive rate of side-channel attack detection. True positive is the cases where side-channel attacks that are correctly identified. We test the Prime-Probe attack [21] and Flush-Reload attack [42]. False positive is defined as benign programs that are falsely identified as an attack. We select different common linux commands and utilities as benign applications. We change the threshold and draw the ROC curves to show the relations between true positive and false positive rate.

We first considered different window sizes w for $\mathbb{S}$ and $\mathbb{S}'$ (Sect. 5). Figure 7 shows the attack detection accuracy under three window size: $w = 1$, 3 and 5. In these experiments, we set the sampling granularity as 1 ms (this sampling rate is different from that of signature detection). From these results we see that *CloudRadar* has an excellent true positive rate: with appropriate thresholds (100–300 events per 1 ms), the true positive rate can be 100 %. However, it also has false positives. When $w = 1$, the false positive rate can be as high as 20 %–30 %. False positives are caused by the coincidence that a benign application experiences a phase transition at exactly the same time as the victim application executes a crypto operation. *CloudRadar* will observe changes in the benign application's cache behavior and think it is due to interference with the victim. Then it will flag this benign VM as malicious. We can increase w to reduce the false positive rate without affecting the true positive rate: when $w = 5$, the false positive rate is close to 0 while true positive rate is 100 %.



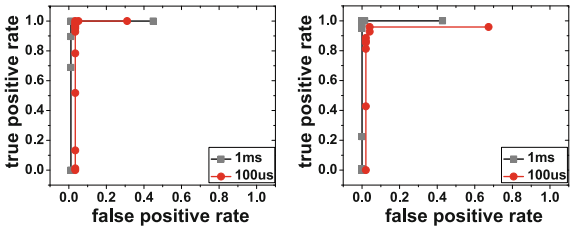
(a) Prime-Probe attack (b) Flush-Reload attack

Fig. 7. ROC curve of attack detection under different window lengths.

We also tested different sampling granularities. Figure 8 shows the ROC curves of detecting two attacks under two different sampling intervals: 1 ms and 100 µs. The window size is 5 data points. We can see the 1 ms interval is better than the 100 µs. This is because when the sampling interval is small, the number of cache events occurring within a sampling period is comparable to the measurement noise. So the measurements under this sampling granularity are not very accurate. It is interesting to note that we need different granularities to sample the victim's CPU events (100 µs) and attacker's LLC events (1 ms). This is because victim's CPU events occur more frequently than the attacker's LLC events. So at the granularity of 100 µs, sampling the victim can give finer information, while sampling the attacker will introduce large Signal-to-noise ratio (SNR), making the results less accurate.

7.2 Performance

Detection Latency. Table 2 reports the detection latency of *CloudRadar* under different window sizes w and sampling granularities. This detection latency is defined as the period from the time the victim VM starts to execute sensitive



(a) PRIME-PROBE attack (b) FLUSH-RELOAD attack

Fig. 8. ROC curve of attack detection under different sampling intervals.

operations (*i.e.*, start of the second stage in Fig. 1) to the time an alarm for side-channel attacks is flagged. We see that *CloudRadar* can identify the attack on the order of milliseconds. Considering side-channel attackers usually need at least several cryptographic operations to steal the keys, this small latency can achieve our *real-time* design goal. We also observe that smaller window sizes and finer granularity can effectively reduce the detection latency, at the cost of slightly lower accuracy.

Table 2. Detection latency (µs) under different window sizes and sampling intervals

(µs)	granularity = 1 ms			granularity = 100 µs		
	$w = 1$	$w = 3$	$w = 5$	$w = 1$	$w = 3$	$w = 5$
PRIME-PROBE	1021.41	3065.86	5110.04	120.49	361.97	603.03
FLUSH-RELOAD	1021.50	3064.38	5107.57	122.48	363.27	605.30

Performance Overhead. We select a mix of benchmarks and real-world applications to evaluate the performance of *CloudRadar*. Our benchmarks can be categorized into three types: (1) crypto programs (AES, SHA, HMAC, BF and MD5 from OpenSSL; ElGamal, RSA and DSA from GnuPG); (2) CPU benchmarks (mcf, gobmk, omnetpp, astar, soplex and lbm from SPEC2006; canneal and streamcluster from PARSEC); (3) Cloud applications from CloudSuite [8] (data analytics, data caching, data serving, graph analytics, media streaming, software testing, web searching and web serving).

We test the performance penalty due to *CloudRadar* and show the normalized run time of each of the benchmark applications in Fig. 9 (results are average of 5 runs, error bars show one standard deviation). The results suggest *CloudRadar* has little impact on the performance of the monitored VM: even in the worst case, performance overhead is within 5 %.

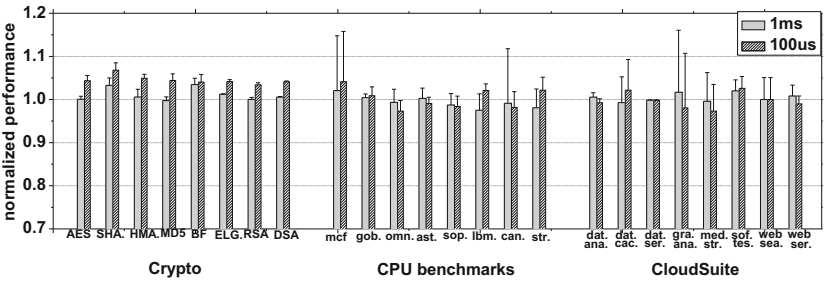


Fig. 9. Performance of different benchmarks under *CloudRadar*

8 Discussions

8.1 Detecting Other Side Channels

One can extend *CloudRadar* to detect cache-based side-channel attacks in other cloud models (*e.g.*, PaaS [46]), or in non-virtualization environments. The only change we need to make is to use performance counters to monitor the processes or threads instead of VMs. Besides, this method can be applied to other micro-architectural side-channel attacks that exploit resource contention. We can use performance counters to count the corresponding events that the attacker uses to retrieve information. For instance, we can monitor the DRAM bandwidth event to detect the DRAM side-channel attacks in [38]. Generalization of this method beyond cache-based side-channel attacks will be future work.

8.2 Potential Evasive Attacks

There can be potential evasive attacks against *CloudRadar*. To evade the detection of *CloudRadar*, a side-channel attacker can try to reduce the cache probing speed, so the abnormal increase in cache misses or hits may not be observed by *CloudRadar*. However, the attacker needs a much longer time to recover the keys, making side-channel attacks more difficult and less practical. An attacker can also try to evade the detection by adding noise to *CloudRadar*'s observations. However, such noise can also blur the attacker's observations, making it more difficult to extract side-channel information. How to design efficient evasive attacks and how to detect such attacks will be future work.

8.3 Limitations

CloudRadar may be limited in several aspects. First, each of its three modules (`Victim Monitor`, `Attacker Monitor` and `Signature Detector`) requires an exclusive use of one physical CPU core as they keep conducting data collection and analysis at full CPU speed. This can potentially reduce the server's capacity for hosting VMs. However, as many cloud servers today are equipped with

dozens of CPU cores, the impact is not as big as one might imagine. Besides, public clouds usually have low server utilization (<20 %) for preserving VMs' QoS [3,22]. So using three cores will not affect VMs' performance. Second, due to the limited number of performance counters available in modern processors, *CloudRadar* has to multiplex the monitoring for each VM using the same counter. When the number of *monitored* vCPUs scales up, *CloudRadar* may miss attacks. We expect future generations of processors will incorporate more performance counters and *CloudRadar* can make use of different counters to monitor different VMs at the same time.

9 Conclusions

This paper designs *CloudRadar*, a real-time detection system to detect cache-based side-channel attacks in clouds. *CloudRadar* leverages the existing hardware performance counter feature to both monitor a victim VM's cryptographic operations and capture a potential attacker VM's abnormal behavior during this time. *CloudRadar* is designed as a lightweight extension to the cloud system and does not require new hardware, hypervisor/OS or application modifications. The feasibility of *CloudRadar* is validated by our implementation on the open source OpenStack cloud system. Our evaluation shows *CloudRadar* can detect cache-based side-channel attacks with high fidelity, while introducing little overhead to the cloud applications.

Acknowledgements. We thank Fangfei Liu and Dr. Yuval Yarom for providing side-channel attack codes, and the anonymous reviewers for their feedback on this work. This work was supported in part by the National Science Foundation under grants NSF CNS-1218817 and NSF CNS-1566444. Any opinions, findings, and conclusions or recommendations expressed in this work are those of the authors and do not necessarily reflect the views of the NSF.

References

1. Azar, Y., Kamara, S., Menache, I., Raykova, M., Shepard, B.: Co-location-resistant clouds. In: ACM Workshop on Cloud Computing Security (2014)
2. Bahador, M., Abadi, M., Tajoddin, A.: HPCMalHunter: behavioral malware detection using hardware performance counters and singular value decomposition. In: IEEE International Conference on Computer and Knowledge Engineering (2014)
3. Barr, J.: Cloud computing, server utilization & the environment (2015). https://aws.amazon.com/blogs/aws/cloud-computing-server-utilization-the-environment/
4. Chiappetta, M., Savas, E., Yilmaz, C.: Real time detection of cache-based side-channel attacks using hardware performance counters. Cryptology ePrint Archive, Report 2015/1034 (2015). http://eprint.iacr.org/
5. Demme, J., Maycock, M., Schmitz, J., Tang, A., Waksman, A., Sethumadhavan, S., Stolfo, S.: On the feasibility of online malware detection with performance counters. In: ACM International Symposium on Computer Architecture (2013)

6. Domnitser, L., Jaleel, A., Loew, J., Abu-Ghazaleh, N., Ponomarev, D.: Non-monopolizable caches: low-complexity mitigation of cache side channel attacks. ACM Trans. Archit. Code Optim. **8**, 35:1–35:21 (2012)
7. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification, 2nd edn. Wiley-Interscience, Hoboken (2000)
8. EPFL: Cloudsuite. http://parsa.epfl.ch/cloudsuite/cloudsuite.html
9. Gruss, D., Maurice, C., Wagner, K., Mangard, S.: Flush+flush: a fast and stealthy cache attack. In: Detection of Intrusions and Malware and Vulnerability Assessment (2016)
10. Gruss, D., Spreitzer, R., Mangard, S.: Cache template attacks: automating attacks on inclusive last-level caches. In: USENIX Conference on Security Symposium (2015)
11. Gullasch, D., Bangerter, E., Krenn, S.: Cache games - bringing access-based cache attacks on aes to practice. In: IEEE Symposium on Security and Privacy (2011)
12. Han, Y., Alpcan, T., Chan, J., Leckie, C.: Security games for virtual machine allocation in cloud computing. In: Das, S.K., Nita-Rotaru, C., Kantarcioglu, M. (eds.) GameSec 2013. LNCS, vol. 8252, pp. 99–118. Springer, Heidelberg (2013)
13. Herath, N., Fogh, A.: These are not your grand daddys CPU performance counters: CPU hardware performance counters for security. In: Black Hat USA (2015)
14. Irazoqui, G., Eisenbarth, T., Sunar, B.: S$A: a shared cache attack that works across cores and defies VM sandboxing - and its application to AES. In: IEEE Symposium on Security and Privacy (2015)
15. Irazoqui, G., Inci, M.S., Eisenbarth, T., Sunar, B.: Wait a minute! A fast, cross-VM attack on AES. In: Stavrou, A., Bos, H., Portokalidis, G. (eds.) RAID 2014. LNCS, vol. 8688, pp. 299–319. Springer, Heidelberg (2014)
16. Jamkhedkar, P., Szefer, J., Perez-Botero, D., Zhang, T., Triolo, G., Lee, R.B.: A framework for realizing security on demand in cloud computing. In: IEEE Conference on Cloud Computing Technology and Science (2013)
17. Kim, T., Peinado, M., Mainar-Ruiz, G.: STEALTHMEM: system-level protection against cache-based side channel attacks in the cloud. In: USENIX Conference on Security Symposium (2012)
18. Li, P., Gao, D., Reiter, M.K.: Stopwatch: a cloud architecture for timing channel mitigation. ACM Trans. Inf. Syst. Secur. **17**, 8:1–8:28 (2014)
19. Liu, F., Ge, Q., Yarom, Y., Mckeen, F., Rozas, C., Heiser, G., Lee, R.B.: Catalyst: defeating last-level cache side channel attacks in cloud computing. In: IEEE International Symposium on High Performance Computer Architecture (2016)
20. Liu, F., Lee, R.B.: Random fill cache architecture. In: IEEE/ACM International Symposium on Microarchitecture (2014)
21. Liu, F., Yarom, Y., Ge, Q., Heiser, G., Lee, R.B.: Last-level cache side-channel attacks are practical. In: IEEE Symposium on Security and Privacy (2015)
22. Liu, H.: A measurement study of server utilization in public clouds. In: IEEE International Conference on Dependable, Autonomic and Secure Computing (2011)
23. Malone, C., Zahran, M., Karri, R.: Are hardware performance counters a cost effective way for integrity checking of programs. In: ACM Workshop on Scalable Trusted Computing (2011)
24. McCalpin, J.D.: Stream: sustainable memory bandwidth in high performance computers. http://www.cs.virginia.edu/stream/
25. Moon, S.-J., Sekar, V., Reiter, M.K.: Nomad: mitigating arbitrary cloud side channels via provider-assisted migration. In: ACM Conference on Computer and Communications Security (2015)

26. Natarajan, R.: 50 most frequently used unix/linux commands (with examples). http://www.thegeekstuff.com/2010/11/50-linux-commands/?utm_source=feedburner
27. Percival, C.: Cache missing for fun and profit. In: Proceedings of BSDCan (2005)
28. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In: ACM Conference on Computer and Communications Security (2009)
29. Sakoe, H., Chiba, S.: Dynamic programming algorithm optimization for spoken word recognition. IEEE Trans. Acoust. Speech Signal Process. **26**, 43–49 (1978)
30. Sherwood, T., Perelman, E., Hamerly, G., Sair, S., Calder, B.: Discovering and exploiting program phases. IEEE Micro **23**, 84–93 (2003)
31. Shi, J., Song, X., Chen, H., Zang, B.: Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring. In: IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (2011)
32. Tang, A., Sethumadhavan, S., Stolfo, S.J.: Unsupervised anomaly-based malware detection using hardware features. In: Stavrou, A., Bos, H., Portokalidis, G. (eds.) RAID 2014. LNCS, vol. 8688, pp. 109–129. Springer, Heidelberg (2014)
33. Varadarajan, V., Ristenpart, T., Swift, M.: Scheduler-based defenses against cross-VM side-channels. In: USENIX Conference on Security Symposium (2014)
34. Varadarajan, V., Zhang, Y., Ristenpart, T., Swift, M.: A placement vulnerability study in multi-tenant public clouds. In: USENIX Security Symposium (2015)
35. Vattikonda, B.C., Das, S., Shacham, H.: Eliminating fine grained timers in Xen. In: ACM Workshop on Cloud Computing Security (2011)
36. Wang, X., Karri, R.: Numchecker: detecting kernel control-flow modifying rootkits by using hardware performance counters. In: ACM/EDAC/IEEE Design Automation Conference (2013)
37. Wang, X., Konstantinou, C., Maniatakos, M., Karri, R.: Confirm: detecting firmware modifications in embedded systems using hardware performance counters. In: IEEE/ACM International Conference on Computer-Aided Design (2015)
38. Wang, Y., Ferraiuolo, A., Suh, G.E.: Timing channel protection for a shared memory controller. In: IEEE International Symposium on High Performance Computer Architecture (2014)
39. Wang, Z., Lee, R.: A novel cache architecture with enhanced performance and security. In: IEEE/ACM International Symposium on Microarchitecture (2008)
40. Wang, Z., Lee, R.B.: New cache designs for thwarting software cache-based side channelattacks. In: ACM International Symposium on Computer Architecture (2007)
41. Xia, Y., Liu, Y., Chen, H., Zang, B.: CFIMon: detecting violation of control flow integrity using performance counters. In: IEEE/IFIP International Conference on Dependable Systems and Networks (2012)
42. Yarom, Y., Falkner, K.: Flush+reload: a high resolution, low noise, l3 cache side-channel attack. In: USENIX Conference on Security Symposium (2014)
43. Yuan, L., Xing, W., Chen, H., Zang, B.: Security breaches as PMU deviation: detecting and identifying security attacks using performance counters. In: Asia-Pacific Workshop on Systems (2011)
44. Zhang, T., Lee, R.B.: Cloudmonatt: an architecture for security health monitoring andattestation of virtual machines in cloud computing. In: ACM International Symposium on Computer Architecture (2015)
45. Zhang, Y., Juels, A., Reiter, M.K., Ristenpart, T.: Cross-VM side channels and their use to extract private keys. In: ACM Conference on Computer and Communications Security (2012)

46. Zhang, Y., Juels, A., Reiter, M.K., Ristenpart, T.: Cross-tenant side-channel attacks in PaaS clouds. In: ACM Conference on Computer and Communications Security (2014)
47. Zhang, Y., Li, M., Bai, K., Yu, M., Zang, W.: Incentive compatible moving target defense against VM-colocation attacks in clouds. In: Gritzalis, D., Furnell, S., Theoharidou, M. (eds.) SEC 2012. IFIP AICT, vol. 376, pp. 388–399. Springer, Heidelberg (2012)
48. Zhang, Y., Reiter, M.K.: Düppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud. In: ACM Conference on Computer and Communications Security (2013)