Security Threats in Cloud Rooted from Machine Learning-Based Resource Provisioning Systems

Hosein Mohammadi Makrani[1](✉), Hossein Sayadi[2], Najmeh Nazari[1], and Houman Homayoun[1]

[1] University of California, Davis, USA
{hmakrani,nnazaribavarsad,hhomayoun}@ucdavis.edu
[2] California State University, Long Beach, USA
hossein.sayadi@csulb.edu

Abstract. Resources provisioning on the cloud is problematic due to heterogeneous resources and diverse applications. The complexity of such tasks can be reduced with the aid of Machine Learning. Researchers have found, however, that machine learning poses new threats such as adversarial attacks. Based on our investigation, we found that adversarial ML can target resource provisioning systems (RPS) to perform distributed attacks. Our work proposes a fake trace generator (FTG), which can be wrapped around an adversary kernel to avoid detection by the RPS and to enable the adversary to get co-located with the victim's virtual machine.

Keywords: Machine-learning · Cloud · Resource-provisioning

1 Introduction

Due to the rise of social media, Internet-of-Things (IoT), and multimedia, the volume of data has increased continuously, resulting in an overwhelming amount of data known as big data. In order to efficiently process such massive data, scale-out architecture has gained interest as a promising solution that is designed to provide a massively scalable computer architecture. Recent improvements in the networking, storage, energy-efficiency and infrastructure management have made cloud (the best example of scale-out architecture) a preferable approach to respond to the new computing challenges.

A resource provisioning system provides various services including resource efficiency [11], security, fault tolerance, and monitoring to achieve the performance goals while maximizing the utilization of available resources [10] in the cloud. The latest recourse provisioning systems, which they were successful to significantly improve the utilization, used machine learning techniques to overcome the challenge of diversity of applications and heterogeneity of resources in the cloud.

RPSs routinely schedule multiple applications from multiple users on the same physical hosts to increase efficiency, in a way that applications have minimum impact on each other's performance. Moreover, a recent work proposed to exploit information used by resource provisioning systems for scheduling purposes, for detecting an adversary VM by its micro-architectural trace and behavior. In this way, they are actually adding another line of defense, this time in the scheduling phase, against attackers.

On the other hand, the interference on shared resources from multi-tenancy can lead to security and privacy vulnerabilities. Interference may leak important information ranging from a service's placement to confidential data, like private keys [3]. This has prompted significant work on distributed side-channel [9] and distributed denial of service attacks (DDoS) [6], data leakage exploitations [22], and attacks that pinpoint target VMs in a cloud system [19]. However, none of the above attacks targeted the resource provisioning system by itself to use it as a new point of vulnerability and a platform for their attacks. Most of those attacks leverage the lack of strictly enforced resource isolation between co-scheduled instances and the naming conventions cloud frameworks use for machines to extract confidential information from victim applications, such as encryption keys.

In this work, we show how utilizing machine learning in resource provisioning systems can become a blind spot and weakness to be exploited by adversaries for planning an attack. Despite the machine learning systems being deployed in numerous applications and shown robustness against random noises [18], the exposed vulnerabilities have shown that the outcome of ML models can be modified or controlled by adding specially crafted perturbations to the input data, often referred to as Adversarial samples. A plethora of works on adversarial attacks exists, focusing specifically on computer vision applications, where the number of features is often large. Recently, a few works on crafting adversarial traces are as well proposed [12].

We argue that the adversarial samples in ML can be leveraged to impose security risk and manipulate today's ML-based RPSs by reverse engineering the ML models from the performance and utilization data these systems generate. We show an example (DDoS attack) to how an adversary can bypass the instance initialization phase of RPS and get co-located by victims with high probability. We also will show how it is possible to disguise the malicious behavior of the adversary's VM and still remain on the same host with the victim and avoid the migration. To create such a fake trace generator, we use the concept presented in [12] for the adversarial sample generation in machine learning. By reverse engineer the resource provisioning system, we can create an adversarial sample for the adversary's application trace. We run FTG as a separate thread inside the adversarial VM and by expecting the transferability of such an attack, we improve the effectiveness of distributed attacks.

2 Security Threats

We show that ML solutions hides security vulnerabilities, since it enables an adversary to extract information about an application's type and characteristics. An adversarial VM has the goal of disguising as Friendly VM to determine the nature and characteristics of any applications co-scheduled on the same physical host, and negatively impact their behavior.

2.1 Threat Model

Our work focuses on IaaS providers that offer public clouds to mutually untrusting customers where multiple VMs can be co-located on the same server. VMs do not have any control over where they are placed, nor do they have any information about other VMs on the same physical host. As a result, at this point, we assume that the resource provisioning system will be neutral with respect to detection of adversarial virtual machines, which means that it won't assist such attacks or employ additional resource isolation techniques to prevent them.

Adversarial VM: Adversary virtual machines are designed to steal information or negatively impact the performance of the victims by getting co-located with them and evading detection mechanisms embedded in resource provisioning systems.

Friendly VM: One or more applications are run on this virtual machine, which is scheduled on a physical host. No techniques for preventing detection, such as obfuscation of memory patterns, are used.

2.2 Distributed Attack

A distributed attack [1] goal is to retrieve secret information, or decrease the performance of computing nodes on a distributed system, where each computing node processes a part of the overall data. The examples of retrieved information may be a set of encryption keys that can be used to compromise the functionality of the whole distributed system. A distributed attack may also be used to retrieve information about the cloud infrastructure such as FPGA cartography and fingerprinting. In the following, we present some characteristics and provide more details of such attacks.

Definition 1. We can define the distributed attack over a set M_{vic} of virtualized instances running in a distributed system S, as a tuple $DSCA = (S$, M_{vic}, D, M_{mal}, A, CP, $EP)$ where: S is a distributed system; M_{vic} are the VMs that are targeted by the attack; D is the distributed dataset to be compromised (partially or totally); M_{mal} are malicious VMs, co-located with the victim VMs; A is a set of local attack techniques (such as side channel [16], denial of service, or resource freeing attack); CP is a protocol to coordinate the attacker VMs in M_{mal}; EP is a protocol to exfiltrate data.

We consider $D = d_1, ..., d_n$ a dataset to be processed by the distributed system $S = s_1, ..., s_n$ implemented on a set of VMs $M_{vic} = m_{vic1}, ..., m_{vicn}$ on a virtualized platform. Each component s_i of S processes data d_i locally and runs in its own VM m_{vici}. To perform the distributed attack, the adversary sets up a number of malicious VMs(at least equal to the number of M_{vic}) $M_{mal} = m_{mal1}, ..., m_{maln}$, co-located with the victim instance M_{vic}. The adversary also masters a set $A = a_1, ..., a_m$ of local attack techniques, i.e., Flush+Reload.

The objective of a distributed attack is to first attack each component of the system s_i running on m_{vici} through m_{mali} running local attack technique a_j to retrieve d_i. The synchronization between attack instances and a central server may be performed using a coordination protocol CP. A protocol EP may be used to control attacking instances remotely, and to send collected information to a remote server to exfiltrate sensitive data. In the following, we briefly explain three well-known local attack on a distributed system:

Side Channel Attack. By sharing physical resources like processor caches, or by using virtualization mechanisms, side-channels may occur due to lack of enforced isolation. The side channel is a hidden information channel that is different from the main channel (e.g., network), in that the protection mechanisms around the data might not be adequate to prevent security violations. The purpose of a side channel attack is to exploit a side channel for obtaining critical information. Side channel attacks can be classified according to the type of exploited channel. The two most popular types of SCA are timing attacks and cache-based attacks, where the cache memory of the processor is often exploited by adversaries.

Denial of Service Attack. The overloading of server resources caused by a denial of service attack degrades the performance of the victim service. These attacks can be classified as external or internal (or host-based) in cloud settings specifically. IaaS cloud multi-tenancy allows internal DoS attacks to launch adversarial programs on the same host as the victim and impact its performance.

Resource Freeing Attack. In addition, resource-freeing attacks (RFAs) hurt the victim's performance as well as forcing them to surrender resources to the adversary [17]. Despite their effectiveness, RFAs require significant compute and network resources, and are subject to defenses, like live VM migration.

2.3 Attack's Setting: VM Co-location

An adversarial VM is rarely interested in a random service running on a public cloud. They need to pinpoint where the target resides in a practical manner to be able perform DoS, RFA, or SC attacks. This requires a launch strategy and a mechanism for co-residency detection. The attack is practical if the target is located with high accuracy, in reasonable time and with modest resource costs. We show that by black box attack to the RPS's model and eventually generating adversarial sample, we can force the RPS to put the Adversarial VM on the desired host. Once a target VM is located, the adversary can launch RFA, or DoS attack.

2.4 Locating Physical Hosts Running Victim Instances

In order to accomplish co-residency with the victim instance, an attacker needs to launch several VMs. This is impractical and not feasible. As side-channel and

RFA attacks are local attacks, it is essential that the malicious VMs reside on the same physical host as the victim VMs. Finding the physical hosts running virtual machines on which the victims are running is therefore the first and most important step. It is important to consider factors such as datacenter region, instance type, and time interval when aiming for co-residency. Among IaaS clouds, these variables may vary. The application type is, however, considered an important factor in placement [21]. Let $P(m_{mali})$ be the probability of instance m_{mali} to be co-resident with instance victim m_{vici}. The value of P will be raised by increasing the number of launched attack instances. To make sure that both attacker and victim VMs achieve coresident placement, the adversary can perform co-residency detection techniques such as network probing [7]. The attacker can also use data mining techniques to detect the type and characteristics of a running application in the victim VM by analyzing interferences introduced in the different resources to increase the accuracy of co-residency detection.

2.5 Avoidance of Detection and Migration

In virtualized environments, there are several techniques for detecting attacks. A side-channel attack, for example, would require very fine-grained information in order to be detected [15]; this information can primarily be provided by Hardware Performance Counters (HPCs) [13]. Modern microprocessors contain a set of special-purpose registers called HPCs that capture hardware events such as last-level cache (LLC) load misses, branch instructions, branch misses, and executed instructions while executing an application. Events of this type are primarily used for analyzing program behavior and are accessible to everyone in the user space. Detection of abnormalities in computer systems is also based on these events. We distinguish two different methods of detection: (1) signature based [14] and (2) threshold-based [2]. The signature-based approach generates a signature of the attack based on information received from HPCs and compares the behavior of the system with the generated signature to identify if any malicious activity has been detected. On the other hand, threshold-based approaches utilize the HPCs trace to flag anomaly resource utilization that goes beyond a pre-specified threshold.

3 ML Based Resource Provisioning System

Figure 1 shows how a normal ML based RPS works. First, they monitor the application and extracts its micro architectural information. Then based on the current behavior and server configuration, they generate a performance model for the application. By leveraging an optimization techniques and available cost model, they determine the suitable configuration and host for the application. In this study we use PARIS [20], a ML based performance model proposed at Berkeley as a cost aware resource provisioning system.

PARIS uses Random Forest for predicting performance from the application fingerprint to find the best VM type configuration. To generate the fingerprint of application, PARIS extracts 20 resource utilization counters spanning the following broad categories and calls it fingerprint: CPU utilization, Network utilization, Disk utilization, Memory utilization, and System-level features. On the other hand, CPU count, core count, core frequency, cache size, RAM per core, memory bandwidth, storage space, disk speed, and network bandwidth of the server are the representation of the configuration provisioned by PARIS.

We denote the micro architectural fingerprint and system level information of an application as *Finger_print* vector. In Eq. (1), f_i denotes each architectural feature.

$$Finger\_print = \{f_1, f_2, ..., f_{20}\} \tag{1}$$

configuration parameters of the server platform referred to configuration inputs is as follow:

$$Configuration = \{c_1, c_2, ..., c_9\} \tag{2}$$

where *Configuration* is the configuration vector and c_i is the value of the ith configuration parameter (number of sockets, number of cores, core frequency, cache size, memory capacity, memory frequency, number of memory channel, storage capacity, storage speed, network bandwidth).

The RPS is responsible to provision *Configuration* based on *Finger_print*:

$$Configuration = f(Finger\_print) \tag{3}$$

Note that $f(Finger\_print)$ is just a data model, which means there is no direct analytical equation to formulate it.

3.1 Reverse Engineering the Model

As mentioned, RPS can be considered as a blackbox (worst-case scenario). In such cases, we perform a reverse engineering to mimic the functionality of the RPS. Thus, as a first step to craft adversarial malware, we perform reverse engineering similar to that proposed in [8].

In order to reverse engineer, we first create a training dataset that comprises of all types of applications. Nearly 11,000 applications are used in the reverse engineering process. The Original RPS is fed with all the applications and the responses are recorded. These responses are utilized to train different ML classifiers in order to mimic the functionality of the original RPS. Further, it is tested by comparing the outputs from original RPS response and the reverse engineered RPS's response. Reverse engineering is non-trivial as the adversaries generated on a closely functional model will be highly effective compared to a weakly generated adversary. To ensure the reverse engineering is performed in an efficient way, we train multiple ML classifiers and choose the classifier that yields high accuracy.

Table 1. Detailed information of local cluster

Server (Xeon)	Freq. (GHz)	Socket	Core	Cache (MB)	Mem. (GB)	Storage	Server type	Count
E5-4669 V4	2.2	4	22	55	96	SSD PCIe	HPC	2
E5-4667 V4	2.2	4	18	45	64	SSD SATA	HPC	2
E5-4650 V4	2.2	4	14	35	32	SSD SATA	HPC	2
E5-2690 V4	2.6	2	14	35	512	SSD/HDD	Memory opt.	4
E5-2650 V4	2.2	2	12	30	256	SSD/HDD	Memory opt.	4
E5-2667 V4	3.2	2	8	25	32	SSD PCIe	I/O opt.	4
E5-2643 V4	3.4	1	6	20	32	SSD PCIe	I/O opt.	4
E5-2660 V2	2.2	2	10	25	16	HDD	General purp.	6
E5-2650 V2	2.6	2	8	20	16	HDD	General purp.	6
E5-1630 V4	3.7	1	4	10	8	HDD	Power opt.	2
E5-1680 V4	3.4	1	8	20	12	HDD	Power opt.	2
E3-1270 V6	3.8	1	4	8	8	HDD	Power opt.	2

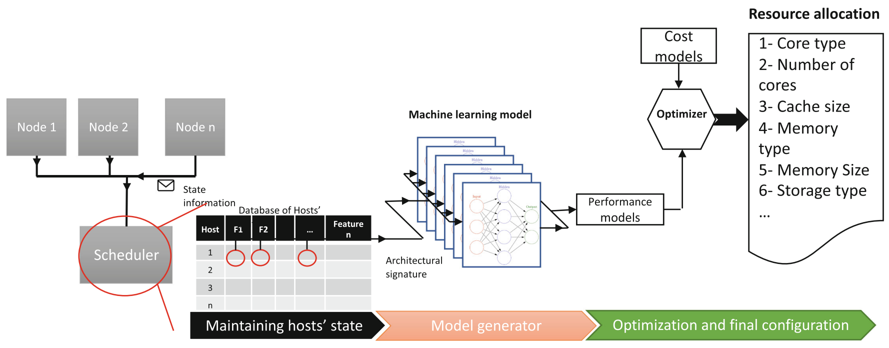

Fig. 1. ML based resource provisioning system

We perform the data collection in a controlled environment, where all applications are known. We use a 40-machine cluster (presented in Table 1), and schedule a total of 120 workloads, including batch analytics in Hadoop and Spark and latency-critical services, such as webservers, Memcached and Cassandra. For each application type, there are several different workloads with respect to algorithms, framework versions, datasets, and input load patterns. The training set is selected to provide sufficient coverage of the space of resource characteristics. The selected workloads cover the majority of the resource usage space.

We submit all of these applications to RPS. In the beginning, the RPS profiles the application and extracts the fingerprint. Then, the RPS uses the Random Forest model to determine an appropriate server configuration. We collect all the fingerprints and their correspondent configurations generated by the RPS to shape our dataset.

3.2 Adversarial Sample Generator

Once the reverse engineered RPS is built, it is non-trivial to determine the level of perturbations that need to be injected into application's micro architectural patterns in order to get the desired host configuration. The micro architectural patterns are perturbed by applying a gradient loss based approach, similar to the Fast-Gradient Sign Method (FGSM), which is widely used in image processing. The low complexity and low computation overheads are the benefits of such an approach. To train our neural network, we use reverse engineered ML RPS i.e., neural network with θ as the hyper parameters, x being the input to the model, and y is the output for a given input x, and $L(\theta, x, y)$ be the cost function used to craft adversarial perturbations. Using the gradient of the cost function of the neural network, the perturbation required to change the output to the target configuration is calculated. The adversarial perturbation generated based on the gradient loss, similar to the FGSM [5] is given by:

$$x^{adv} = x + \epsilon\, sign(\nabla_x\, L(\theta, x, y)$$

where ϵ is a scaling constant ranging between 0.0 to 1.0 is set to be very small such that the variation in $x(\delta x)$ is undetectable. In case of FGSM the input x is perturbed along each dimension in the direction of gradient by a perturbation magnitude of ϵ. Considering a small ϵ leads to well-disguised adversarial samples that successfully fool the ML model. In contrast to the images where the number of features are large, the number of features i.e., micro architectural metrics are limited, thus the perturbations need to be crafted carefully and also be made sure it can be generated during runtime by the applications. For instance, a negative value cannot be generated by an application. Hence, we provided lower bound on the adversary values. [4] presented how to craft the adversarial application so as to generate the perturbations during runtime.

3.3 Case Study

To evaluate our proposed approach, we implemented 8 distributed attacks as follow: SC1: Prime+Probe, SC2: Flush+Reload, SC3: Flush+Flush, SC4: Evict + Time, DoS1: increasing latency by saturating the network, DoS2: decreasing throughput by saturating storage, RFA1: freeing memory resource, and RFA2: freeing CPU resource. We perform these attacks on 20 unseen victim applications from different domains (SPEC, Hadoop, Spark, Memcache, and Cassandra). Based on our evaluation, the success rate of being co-located with victims, evading the detection and migration, and getting the desired outcome from attack depends on many factors such as victim's type, the period of monitoring phase, and amount of perturbation.

We now perform the DoS attack on utilization. If it causes the resource saturation, DoS will be detected and the victim will be migrated to a new machine. Our cluster supports live migration. Figure 2 compares the tail latency and CPU utilization with adversarial VM to that of a naive DoS that saturates the CPU through a CPU-intensive task. It shows the adversarial attack does not saturate the resource and does not cause the migration while still can put pressure on the victim.

Table 2. Effectiveness of distributed attacks based on application type

	SPEC	Hadoop	Spark	Memchahced	Cassandra
SC1	***	*	**	*	**
SC2	***	*	*	*	**
SC3	***	*	**	**	*
SC4	***	**	**	*	**
DoS1	*	*	***	***	**
DoS2	*	***	*	*	***
RFA1	*	**	***	***	**
RFA2	***	**	***	***	*

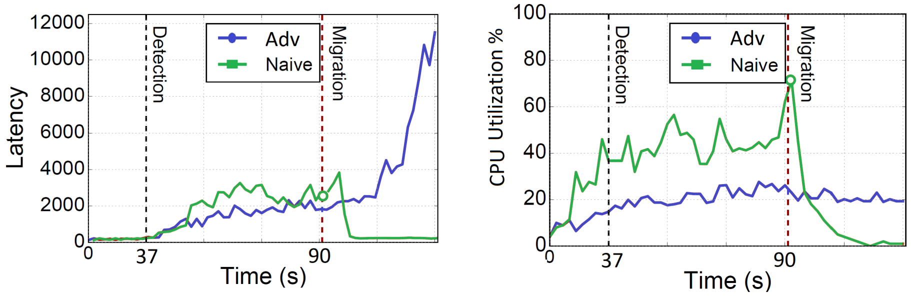

Fig. 2. Latency and utilization with adversarial sample and a naive DoS attack that saturates memory resources

Table 2 shows the impact of victims' type on the success rate of each type of attack. The interesting observation is that there is a meaningful relationship between the application's type and the nature of the attack by itself. For instance, we observe that side-channel attacks are more successful when the cache hit rate of the victim is low. Similarly, we observed that RFA is more successful when the resource utilization of the victim is high. One reason is that in such case FTG can generate a better fake trace to convince the RPS to stay at the current host. In a case that the difference between the behavior of the adversary kernel and the victim is high, the FTG has to generate more perturbation and this may lead to a migration decision by RPS.

4 Conclusions

The proposed adversarial attack on RPS comprises of three phases. Firstly, we perform reverse engineering to build a ML RPS that mimics the functionality of the original RPS. Further, with the aid of adversarial sample generator, the micro architectural pattern required to obtain the target server configuration is determined. Lastly, this crafted adversarial micro architectural pattern generator is spawned as separate thread, leading to overall pattern close to the victim VM's pattern, and eventually causes to be co-located with it. This means without saturating the resources or act as an abnormal application, by generating only small noise in applications behavior, we can force RPS to co-locate the adversary VM with victim and also fool the RPS to change the resources required for the targeted VM and impacts on its behavior. The goal of this study is to encourage public cloud providers to implement more stringent isolation solutions for their platforms and system engineers develop robust RPSs to deliver predictability and security at high utilization levels.

References

1. Bazm, M.-M., Lacoste, M., Südholt, M., Menaud, J.-M.: Isolation in cloud computing infrastructures: new security challenges. Ann. Telecommun., 197–209 (2019). https://doi.org/10.1007/s12243-019-00703-z
2. Chiappetta, M., Savas, E., Yilmaz, C.: Real time detection of cache-based side-channel attacks using hardware performance counters. Appl. Soft Comput. **49**, 1162–1174 (2016)
3. Delimitrou, C., Kozyrakis, C.: Quasar: resource-efficient and QoS-aware cluster management. In: ACM SIGARCH Computer Architecture News, vol. 42, pp. 127–144. ACM (2014)
4. Dinakarrao, S.M.P., et al.: Adversarial attack on microarchitectural events based malware detectors. In: DAC (2019)
5. Goodfellow, I.J., et al.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
6. Gupta, S., Kumar, P.: VM profile based optimized network attack pattern detection scheme for DDOS attacks in cloud. In: Thampi, S.M., Atrey, P.K., Fan, C.-I., Perez, G.M. (eds.) SSCC 2013. CCIS, vol. 377, pp. 255–261. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40576-1_25
7. İnci, M.S., Gulmezoglu, B., Eisenbarth, T., Sunar, B.: Co-location detection on the cloud. In: Standaert, F.-X., Oswald, E. (eds.) COSADE 2016. LNCS, vol. 9689, pp. 19–34. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-43283-0_2
8. Khasawneh, K.N., et al.: RHMD: evasion-resilient hardware malware detectors. In: MICRO (2017)
9. Liu, F., Ren, L., Bai, H.: Mitigating cross-VM side channel attack on multiple tenants cloud platform. JCP **9**(4), 1005–1013 (2014)
10. Makrani, H.M., et al.: Adaptive performance modeling of data-intensive workloads for resource provisioning in virtualized environment. ACM Trans. Model. Perform. Eval. Comput. Syst. (TOMPECS) **5**(4), 1–24 (2021)
11. Makrani, H.M., Sayadi, H., Motwani, D., Wang, H., Rafatirad, S., Homayoun, H.: Energy-aware and machine learning-based resource provisioning of in-memory analytics on cloud. In: Proceedings of the ACM Symposium on Cloud Computing, pp. 517–517 (2018)
12. Makrani, H.M., et al.: Cloak & co-locate: adversarial railroading of resource sharing-based attacks on the cloud. In: 2021 IEEE International Symposium on Secure and Private Execution Environment Design (SEED). IEEE (2021)

13. Mucci, P.J., Browne, S., Deane, C., Ho, G.: PAPI: a portable interface to hardware performance counters. In: Proceedings of the Department of Defense HPCMP Users Group Conference, vol. 710 (1999)
14. Payer, M.: HexPADS: a platform to detect "Stealth" attacks. In: Caballero, J., Bodden, E., Athanasopoulos, E. (eds.) ESSoS 2016. LNCS, vol. 9639, pp. 138–154. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30806-7_9
15. Sayadi, H., et al.: Towards accurate run-time hardware-assisted stealthy malware detection: a lightweight, yet effective time series CNN-based approach. Cryptography **5**(4), 28 (2021)
16. Sayadi, H., et al.: Recent advancements in microarchitectural security: review of machine learning countermeasures. In: 2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS), pp. 949–952. IEEE (2020)
17. Varadarajan, V., Kooburat, T., Farley, B., Ristenpart, T., Swift, M.M.: Resource-freeing attacks: improve your cloud performance (at your neighbor's expense). In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 281–292. ACM (2012)
18. Wang, H., Sayadi, H., Sasan, A., Rafatirad, S., Mohsenin, T., Homayoun, H.: Comprehensive evaluation of machine learning countermeasures for detecting microarchitectural side-channel attacks. In: Proceedings of the 2020 on Great Lakes Symposium on VLSI, pp. 181–186 (2020)
19. Xu, Z., Wang, H., Wu, Z.: A measurement study on co-residence threat inside the cloud. In: 24th USENIX Security Symposium (USENIX Security 15), pp. 929–944 (2015)
20. Yadwadkar, N., et al.: Selecting the best VM across multiple public clouds: a data-driven performance modeling approach. In: ACM SoCC (2017)
21. Zhang, W., et al.: A comprehensive study of co-residence threat in multi-tenant public PaaS clouds. In: Lam, K.-Y., Chi, C.-H., Qing, S. (eds.) ICICS 2016. LNCS, vol. 9977, pp. 361–375. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-50011-9_28
22. Zhang, Y., Juels, A., Reiter, M.K., Ristenpart, T.: Cross-VM side channels and their use to extract private keys. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 305–316. ACM (2012)