THEME ARTICLE: SECURITY AND PRIVACY-PRESERVING EXECUTION ENVIRONMENTS

Adversarial Attacks Against Machine Learning-Based Resource Provisioning Systems

Najmeh Nazari, *University of California Davis, Davis, CA, 95616, USA*
Hosein Mohammadi Makrani, *Apple, Inc. Cupertino, CA, 95014, USA*
Chongzhou Fang, *University of California Davis, Davis, CA, 95616, USA*
Behnam Omidi, *George Mason University, Fairfax, VA, 22030, USA*
Setareh Rafatirad, *University of California Davis, Davis, CA, 95616, USA*
Hossein Sayadi, *California State University Long Beach, Long Beach, CA, 90840, USA*
Khaled N. Khasawneh, *George Mason University, Fairfax, VA, 22030, USA*
Houman Homayoun, *University of California Davis, Davis, CA, 95616, USA*

Microarchitectural attacks, such as side-channel, exploit shared resources to leak sensitive information. Performing microarchitectural attacks on the cloud is possible once the attacker's virtual machine (VM) is co-located with the victim's VM. Hence, the co-location requirement with the victim limits the practicality of microarchitectural attacks on the cloud. In this work, we demonstrate that resource provisioning systems (RPSs) can be exploited to solve the co-location challenge of microarchitectural attacks in the cloud by deploying adversarial evasion attacks on RPSs to co-locate attackers' VMs with victims' VMs. Moreover, we discuss the adaptability of defense techniques proposed against adversarial attacks in the image classification domain on the RPSs.

Multitenancy capability enables security and privacy vulnerabilities to be exploited. Since the cloud resources are shared among different users in a multitenant environment, it facilitates the setup for performing a wide range of resource sharing-based attacks, i.e., microarchitectural attacks.[1] The prerequisite for such attacks on the cloud is the co-location of the adversary and victim on the same host. Therefore, the co-location step is important and challenging especially for attacks that exploit microarchitectural vulnerabilities in the cloud.

It has been shown how resource provisioning systems (RPSs) could be an Achilles heel and vulnerability that can be exploited to solve the co-location challenge of resource sharing-based attacks.[1] In this adversarial machine learning analysis, we explore an attack called *Cloak & Co-locate,*[1] which allows an attacker to put a Cloak on any malicious VM to force the RPS to Co-locate the malicious VM with the victim's VM, without modifying the malicious program kernel.

The attack methodology is composed of two main steps: 1) reverse engineering and 2) generating adversarial samples. The assumption is that the attacker has black-box access to the machine language (ML)-based RPS; therefore, they require a proxy model for their adversarial ML attack setup. Thus, reverse engineering of RPS is necessary to generate the substitute (proxy) model.

After the reverse engineering, the attacker generates adversarial examples (an input added with crafted noises to fool the ML model) for the substitute model, instead of the black-box original RPS. Adversarial examples are utilized to create a Cloak that allows an adversary to co-locate with the victim. It is noteworthy that

0272-1732 © 2023 IEEE
Digital Object Identifier 10.1109/MM.2023.3267481
Date of publication 19 June 2023; date of current version 28 August 2023.

the Cloak keeps unchanged not only the functionality of the malicious kernel but also its syntax. The Cloak is implemented as a fake trace generator (FTG) that wraps around an adversary kernel to generate adversarial examples based on the victim's performance profiling trace to enforce the co-location in the first phase of the attack. In the next phase of the attack, the adversarial examples are used to fool the RPS to prevent malicious behavior detection and migration to another host.

Although several defense techniques were proposed against adversarial attacks on computer vision applications, these defenses are not studied for enhancing the security of ML models used in RPS. The assumption in computer vision-based defenses is that the attacker has a limitation in the amount of perturbation that can be added to the input (an image) without changing its content. It is important because in the computer vision domain, adversaries aim to perturb an image to fool a machine learning classifier, and at the same time, the image should be classified correctly by human eyes. In the RPS domain, there is no such limitation for perturbing a program, and this allows the attacker to have more freedom in adding perturbations.

IN THE RPS DOMAIN, THERE IS NO SUCH LIMITATION FOR PERTURBING A PROGRAM, AND THIS ALLOWS THE ATTACKER TO HAVE MORE FREEDOM IN ADDING PERTURBATIONS.

To the best of our knowledge, this is the first work that discusses the challenges of adopting defense mechanisms against adversarial attacks that target RPSs. We argue that only a few of the current defenses are applicable to RPSs in the cloud, and we highlight a serious need for new techniques to be invented that guarantee security in the RPS domain.

BACKGROUND

The two major tasks of RPSs are 1) instance initialization and 2) periodic monitoring.[2] In the first task, when an instance is submitted to the RPS, RPS should profile it and estimate the resources required for meeting the SLA, based on its profiling trace. Then, RPS allocates a suitable host in the cloud to the instance.

The second task starts after the resource allocation, where the RPS monitors the application's behavior to guarantee the SLA all the time. If the application's behavior changes, the RPS should reschedule and migrate the instance to a new host to allocate the appropriate resources for meeting the SLA agreement.

A periodic monitoring phase can be utilized to improve both security and performance. For security, the profiling trace can be used to detect abnormal behavior. RPS predicts the application's performance based on the current behavior and server configuration, to prevent performance degradation in the subsequent monitoring window of execution. If in the next window, the behavior of the application (the architectural signature) is not the same as what RPS predicted previously, it can mark it as unknown or adversary, since this behavior has not been seen during the training phase of RPS's ML model. Afterward, the VM will simply be isolated/migrated, and a signal will be sent to the operator for further decisions.

THREAT MODEL

Our threat model targets infrastructure as a service (IaaS) which is a platform for mutually untrusting users. In this model, VMs from different users can be co-located on the same host. VMs cannot control where they are placed, and the information of other VMs running on the host has not been disclosed to them. The assumption is that the RPS is neutral in a way that it does not assist attackers or deploy resource isolation techniques to prevent attacks run by adversarial users. In our threat model, RPS is assumed ideal and fair to all VMs and allocates resources based on the workload characteristics of the application rather than their place of origin or intention. Moreover, a black-box access to the RPSs is considered and the underlying model of RPS for placing VM instances is unknown. In this way, we can only submit VM instances and monitor the placement outcome.

In this work, we have two types of VMs: adversarial VM that wants to get co-located with the victims and evade from detection mechanism of RPS (avoiding abnormal behavior or behavior which is not known for RPS) to degrade victims' performance or steal sensitive data. A friendly VM is a benign VM that runs one or more applications. It does not deploy any memory pattern obfuscation, nor does it employ a preventive detection mechanism from an adversary.

We considered the worst case for an attacker. It means that if the adversarial application does not change its behavior and architectural signature, the RPS will detect its abnormal/unknown behavior. Another assumption is that RPS uses any arbitrary technique for detection. In this case, the simplest way is to observe whether the adversary application's profiling trace is predictable, as discussed in the background section.

Hence, adversarial applications should change their micro-architectural and system-level profiling trace to evade detection. Changing the micro-architectural and system-level profiling trace does not include hacking the performance counters or accessing the RPS's database. The adversary application should change its behavior, e.g., using the CPU less, or performing dummy memory access to change the cache misses or memory bandwidth usage. On the other hand, the adversary must remain co-located with the victim. For example, in the case of a change in the application's behavior, there is a high chance that RPS migrates the adversary VM to another host.

Moreover, black-box access to the RPSs is considered; the attacker has limited information and cannot access the internal workings of the RPS. Specifically, the attacker cannot directly observe the metrics/features used by the RPS to make placement decisions nor have knowledge about the underlying model of the targeted RPS. Thus, the attacker can only submit VM instances and monitor the placement outcome.

ADVERSARIAL MACHINE LEARNING IN CLOUD

Most attackers are not interested in a random service running on the cloud. They should pinpoint where the target resides to perform micro-architectural/resource-sharing-based attacks. Hence, co-location and co-residency detection is required first. To locate the victim with high accuracy, in a reasonable amount of time, and with modest resource costs, the following steps should be taken.

Pinpointing Victim

As we mentioned before, the prerequisite is that the adversary VM resides on a host that the victim VM is running on. To achieve co-residency, several parameters should be considered, such as data center region, time interval, and instance type. These parameters can change among IaaS clouds. The application type is one of the most important parameters in the placement strategy. The attacker can deploy co-residency detection techniques such as network probing or use data mining techniques to detect the type of running application in the victim VM to make sure it is co-located with the victim.[1]

Continuous Co-Location With Victim

Most cloud providers, such as Microsoft Azure and Amazon AWS, provide consumer performance monitoring tools. For example, Amazon AWS provides full-stack observability that includes AWS-native, and application performance monitoring (APM), allowing users to understand what is happening across their technology stack at any time. AWS observability lets users collect and analyze telemetry in their network, infrastructure, and applications to gain insights into their systems' behavior, performance, and health.

One of the sources of information that can be used for monitoring the performance of a system is hardware performance counters (HPCs), a set of special-purpose registers, to monitor hardware-related events such as last-level cache (LLC) load misses. These events and other architectural-level traces can be used to detect abnormal behaviors in computer systems.

THEREFORE, THE ATTACKER MUST BEHAVE ADAPTIVELY TO PREVENT DETECTION AND MIGRATION AND TO BE ABLE TO REMAIN BESIDE THE VICTIM TO CONTINUE THE ATTACK.

Although a few low-level microarchitectural events may not be available with fine granularity in some multitenant environments, there are hundreds of other architectural and system-level information that profiling tools can still extract. While the Cloak & Co-locate attack does not fundamentally rely on only HPC traces to show a proof of concept on our infrastructure, we did not restrict access to any profiling information available on our systems. Another recent work shows that even with high-level system information, it is still possible to perform the co-location attacks.[3,4]

There are two types of malicious behavior detection using performance traces of applications: 1) signature-based and 2) threshold-based. Signature-based approaches create the signature of the attack (profiling trace) and compare the trace of the system with the attack signature to identify malicious activity. Resource usage of an application correlates with the type of application.[1] For example, cache activity has been shown to be a very strong indicator of microarchitectural side-channel attacks. If the application has a very high level-1 instruction cache (L1-i) and high LLC pressure, with a high probability it is a cache-based side-channel attack. On the other hand, threshold-based approaches utilize the profiling traces to flag anomaly resource utilization that goes beyond a prespecified threshold. Therefore, the attacker must behave adaptively to prevent detection and migration and to be able to remain beside the victim to continue the attack.

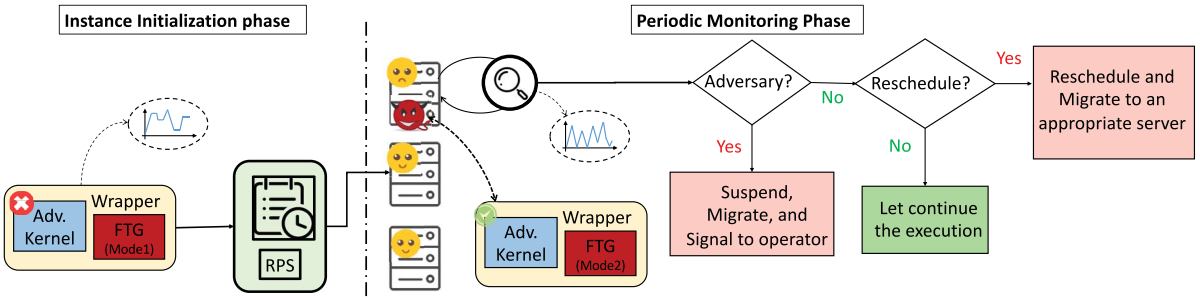

FIGURE 1. Overview of Cloak & Co-locate.

A PRACTICAL ADVERSARIAL ATTACK AGAINST RPS

In any resource-sharing-based attack, instance initialization plays a preventive role in such attacks by preventing a malicious instance from co-locating with the victim instance. This phase of the VM placement algorithm is not disclosed for security reasons by cloud service providers. Despite this, we show how adversaries can bypass the instance initialization phase and Co-locate with victims in a high percentage of cases. When co-residency attacks are detected by periodic monitoring, it depends on the rescheduling policy and how to mitigate them. Our goal is to demonstrate how to conceal the adversary's malicious behavior so that the adversary's instance does not migrate and remains on the same host as the victim.

We propose to use an FTG, called Cloak, to wrap it around an adversary application, first for getting *co-located* with the victim and then evading detection and migration mechanism. During the instance initialization phase, to maximize the chance of co-location, Cloak will perform a trace mimicry task that mimics the behavior of the targeted victim application. If the adversary VM does not get co-located with the victim, it can terminate its VM, and submit a new VM to be instantiated. The cost of such an approach is lower than forcing RPS to migrate the adversary VM to another platform that hosts the victim VM. After co-residency, the Cloak will change its mode to constantly generate a new specialized trace that changes the adversary application's behavior to evade detection and migration.

We exploit the concept of adversarial samples in machine learning to create the Cloak, i.e., FTG, to add specially crafted perturbations to the input signal for fooling the RPS. Our goal is to change the model's output to become similar to the victim's output in the first phase. In the second phase, the FTG performs a targeted adversarial attack on RPS's ML model to find the minimum amount of perturbation to add to the input trace for forcing the RPS to keep the VM on the same host. This is different from the mimicry attack[5] for the first phase which tried to mimic the behavior of the victim's application. Moreover, when we are performing a targeted attack, the perturbation is more inclined to be similar to another legitimate input but in nontargeted attacks, the perturbation is more randomly distributed.

To achieve this, our attack has two parts. In the first part, we have to reverse-engineer the RPS using a machine learning model proxy to know how the RPS makes decisions. In the second part, we utilize the reversed-engineered model (proxy model) to craft our Cloak; an FTG that will add perturbations to the adversary's application trace to force the RPS to Co-locate it with the victim. Figure 1 illustrates the overview of our adversarial attack.

IMPLEMENTATION

In this section, we explain how we reverse-engineered the RPS and then crafted perturbation to fool it. However, the detailed scheme is discussed in Makrani et al.[1]

Reverse Engineering RPS

In the first step, we should train an arbitrary ML model, i.e., a proxy model, that can provision a server with the same configuration that the original RPS provisions for a submitted application to the RPS. As a case study, we consider PARIS, an RPS proposed at UC Berkeley, to act as our original RPS.[1]

PARIS is a machine learning-based RPS that predicts the performance of the application from its micro-architectural signature to find the most efficient configuration. PARIS extracts 20 resource utilization information to create the fingerprint of the application. The information used for the application's fingerprint is collected from the following resources: CPU utilization, network utilization, disk utilization, memory utilization,

and other system-level features. Moreover, CPU count, core count, core frequency, cache size, RAM per core, memory bandwidth, storage space, disk speed, and network bandwidth of the server is used to construct the set of configurations.

We denote the micro-architectural fingerprint and system-level information of an application as a Fing vector

$$\mathrm{Fing} = \{a_1, a_2, ..., a_{20}\} \tag{1}$$

where a_i denotes each architectural feature.

Configuration parameters of the server's platform referred to as configuration inputs are as follows:

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

where Conf is the configuration vector, and c_i is the value of the ith configuration parameter (such as the number of cores, core frequency, etc).

The RPS is responsible for the provision Conf based on Fing:

$$\mathrm{Conf} = f(\mathrm{Fing})\,. \tag{3}$$

We should note that $f(\mathrm{Fing})$ is a data model which means there is no direct analytical equation to formulate it.

We require a training dataset to train the proxy model to mimic the functionality of the original RPS. The dataset has two parts: the first part is the application's fingerprint and the second part is the corresponding configuration provisioned by RPS. By having such a dataset, we can use it for training our proxy model to map those fingerprints to final configurations. More details can be found in Makrani et al.[1]

We use an artificial neural network (ANN) to map the Fing to Conf. We exploit one ANN per each c_i in the Conf. In our detailed evaluation reported in Makrani et al,[1] ANNs perform well with an overall accuracy of 92.7% to mimic the original RPS. Therefore, after reverse-engineering the RPS using an ANN, we have a proxy model ready for simulating the actual RPS to be utilized for generating adversarial samples.

Fake Trace Generator

The Cloak's (i.e., FTG) design has two modes. The first mode activates in the instant initialization phase (before co-location) and the second mode activates in the periodic monitoring phase. The FTG starts working during the instant initialization phase to generate a trace similar to the victim application to get co-located with the victim on the same host.

The adversary kernel starts after co-location. While the kernel starts its attack, the Cloak, i.e., the FTG, switches to mode 2), to carefully generate a fake trace (adversarial example) and disguise the behavior of the adversary kernel. This trace must fool the RPS to maintain the adversary kernel on the same host as the victim. Hence, the FTG constantly monitors the system's state and extract profiling trace such as the HPCs information with the use of profiling tools such as the Perf tool available in Linux.

To craft a trace for fooling the RPS, we employ a targeted adversarial attack on the RPS. Hence, we deploy a low-complex gradient loss-based approach, similar to the fast-gradient sign method (FGSM), which is widely employed in image processing. The advantage of this approach is its low complexity and low computational overheads. We discussed the implementation of targeted adversarial attack details in Makrani et al.[1] When the trace has been calculated [either in mode 1) or mode 2)], FTG uses iBench to put pressure on a specific shared resource by calling a few tunable micro benchmarks. iBench consists of a set of carefully crafted benchmarks that generate contention on the core, the cache, the main memory, the storage, and networking subsystems.

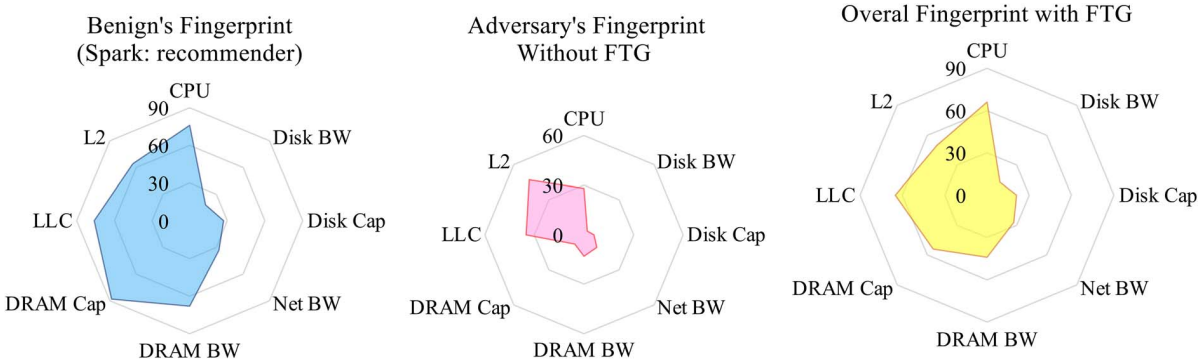

FIGURE 2. Activation of FTG and increase in the similarity of fake trace and victim's fingerprint during instant initialization phase.

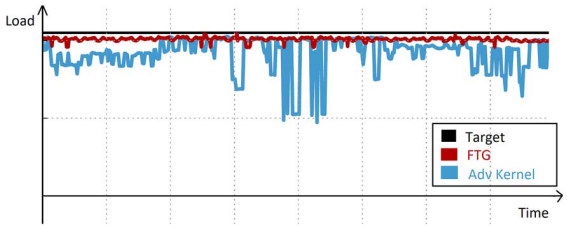

FIGURE 3. Overall trace using FTG, adversary kernel trace, and the target.

In contrast to the images where the number of features is huge, the number of features, i.e., micro-architectural metrics are limited in RPS. Another issue is that a negative perturbation value cannot be generated by an application. The challenges of crafting the adversarial application to generate the perturbations in the micro-architectural features during runtime are discussed in Makrani et al.[1]

Figure 2 demonstrates a side-channel attack and its fingerprint's similarity with a victim application (Spark: recommender system) during the instant initialization phase. The similarity of fingerprint increases to 78%, after the activation of the FTG in mode 1). Hence, the adversary VM can be identified as a friendly VM. The similarity is calculated based on the following Formula: 100 $-$ Average($|X_i - Y_i|$), where X and Y each are an item of profiling trace.

Figure 3 shows how the FTG can generate the desired trace. This example illustrates an adversary CPU load with/without the FTG. The FTG, first, calculates how much CPU load is required using FGSM to be able to fool the RPS. Then it uses iBench to generate the required contention. The target (black line) is a target CPU load calculated by the FTG in the previous monitoring window (this is not the same as the victim's CPU load). If the FTG is not enabled, the adversary CPU load would be the blue line. If the FTG is active, the CPU load would be the red line. As is shown, FTG can match the CPU load of the adversary kernel to the target load by successfully using iBench.

We have demonstrated that adversarial attacks against RPS are practical and require urgent attention. Therefore, we survey the popular solutions proposed for the image classification domain to explore their effectiveness for the RPS domain in the next section.

DISCUSSION: DEFENSES AGAINST ADVERSARIAL ATTACKS TARGETING RPS

In general, there are three approaches for defenses against adversarial attacks: modifying training data, modifying models, and exploiting external models as network add-ons. The first approach does not impact the learning models. However, modifying networks and network add-ons directly deals with the models. A network add-on appends another model to the original one without changing the original model. Each approach can be distinguished as detection only or complete defense. Table 1 summarizes the defense techniques that we briefly discuss here.

Approach 1: Modifying Training

1) *Using adversarial examples in training (adversarial training)*[6]: In this approach, the adversarial examples will be used as a training set during the learning process and retrain the model. This approach regularizes the network and reduces over-fitting to improve the robustness of the network. However, generating adversarial examples against the already adversarially trained models is still possible. Unfortunately, this solution cannot be applied against a Cloak & Co-locate attack. Suppose that we have an ML model. We can generate an adversarial example for it by using one of the well-known methods, such as FGSM, and fool the model. Now, if the model gets trained with the adversarial example again, then the adversarial example generated before cannot fool the model anymore. However, for the new adversarial-trained model, we can still generate another adversarial example by redeploying the FGSM. Back to our scenario, the FTG calculates and generates a new adversarial example each time it is called. Hence, as long as the RPS's ML model is not retrained and updated after we finish the reverse engineering process, the FTG will be able to generate an example to fool the model. This assumption is mostly valid since ML models deployed in cloud infrastructures would not be updated frequently (at least in a period shorter than the execution of an application). If the RPS's ML model gets updated, then redoing the reverse engineering would mitigate the issue of the attack.
2) *Data randomization*[8]: Researchers showed that Gaussian data augmentation and random resizing of inputs reduce the effectiveness of attacks. Since the number of features in RPS is not as large as images, resizing the input may have a large impact on the accuracy of the model. Moreover, the impact of each feature in image classification is almost equal to other features, as pixels are the same type. However, this is not the case in the RPS domain. Therefore, an evaluation is

TABLE 1. Summary of defense techniques.

Method	Category	Purpose	Validation Data Sets	Architecture	Comments on RPS	Effectiveness
Adversarial training[6]	Modified training	Robustness	CIFAR-10, CIFAR-100	VGG, ResNet	Possible to regenerate the perturbation	*
Data compression[7]	Modified input	Robustness	ImageNet	ResNet	Low number of features in RPS	*
Gaussian data augmentation[8]	Modified input	Robustness	MNIST, CIFAR-10	ResNet	Features are not the same in RPS	*
DCN[9]	Modified networks	Complete defense	MNIST	Custom	Not an up-to-date solution	*
Gradient regularization[10]	Modified networks	Complete defense	MNIST, SVHN	Custom	Penalizing model for reacting to small changes	***
Distillation[11]	Modified networks	Complete defense	MNIST, CIFAR-10	Custom	Low number of features in RPS	*
DeepCloak[12]	Modified networks	Complete defense	CIFAR-10	Residual network	Low number of features in RPS	*
Rectify[13]	Add on	Complete defense	ILSVRC 2012	CaffeNet, VGG-F, GoogLeNet	Perturbation detector	****
GAN[14]	Add on	Complete defense	CIFAR-10 and 100	Custom	Robust against FGSM	**
Feature squeezing[15]	Add on	Detection only	MNIST, CIFAR-10, ImageNet	DenseNet, MobileNet	Applicable to images only	*
PCA[16]	Modified training	Robustness	MNIST, CIFAR-10	ResNet	Low number of features in RPS	*
Ensemble[16]	Add on	Robustness	MNIST, CIFAR-10	Custom	Exploiting the combination of solutions	***

required to analyze the tradeoff of security and accuracy when using this technique as a defense.

3) *Data compression*[7]: In the image classification domain, inputs are usually JPEG images. Since JPEG compression itself can remove the high-frequency components, it is observed that JPG compression can be used against FGSM perturbations. The main issue of compression is that compression with a high rate decreases the classification accuracy, while compression with a lower rate is not effective enough. The major difficulty of compression in the RPS domain is that several features utilized by RPS are not suitable for compression.

Approach 2: Model Modifications

1) *Distillation*[11]: Distillation is a training technique that transfers the knowledge of a complex model to a simple model. It is also possible to use the knowledge of the model to enhance its robustness. The evaluation results showed that distillation increases the model's resilience for image classification. However, the applicability of such techniques is under question for the RPS domain because of a much lower number of features available; hence, extracting knowledge and transferring it to itself may not be as effective as the one in the image classification domain.
2) *AutoEncoders*[9]: It has been demonstrated that denoising contractive auto encoders are able to reduce adversarial noise. On the other side, when they stack them with the original models, this results in a more vulnerable model against perturbations. After the success of deep contractive networks against the Limited-memory Broyden–

Fletcher–Goldfarb–Shanno-based[13] attacks, several stronger attacks on deep neural networks are developed. Hence, it may not be an up-to-date solution for even the RPS domain. However, an imperial study is required to understand the impact of autoencoders in RPS models.

3) *Masking*[12]: DeepCloak has been introduced to add a masking layer just before the classification layer. The inserted layer should have the following features: 1) It should be trained by forward-passing both original and adversarial examples. 2) For both original and adversarial examples, it should be able to encode the differences between the output features of the previous layer. This helps to force the dominant weights of the inserted layer to zero just by masking them and, therefore, reduces the sensitivity of the model. For the same reason discussed above, this technique could not be suitable for increasing the robustness of RPS models since the number of features is very limited, and therefore every piece of information is important for RPS to keep the overall accuracy high.
4) *Regularization*[10]: It has been shown that by penalizing the model during training for some degree of variation in the output with respect to change in the input, the model can be robust against small adversarial perturbation. If this method combines with adversarial training, it significantly increases the robustness against attacks. Unfortunately, this will double the complexity of training a model. However, it might seem an interesting technique to robust the RPS since, by default RPS is penalized if it frequently changes its classification in response to a small change in the application's behavior. The cost for RPS to change its classification is to migrate the application to a new system that may lead to SLA agreement violations.

Approach 3: Add-Ons to the Model

1) *Feature squeezing*[15]: Feature squeezing is a technique to detect whether an image is an adversarial example. By using two extra models besides the main model, extra models can reduce the color bit depth pixels, and apply spatial smoothing on the input image. If the classification of the original image and the squeezed is different, the adversarial example would be detected. Based on our investigation, the current technique is not directly applicable to RPS since the input of the RPS model is not an image. Moreover, it is just a detection technique, and for a complete defense, it should be used along with another technique.
2) *Generative adversarial network-based defense*[14]: It has been shown that using generative adversarial networks to train a model can be robust against FGSM attacks. In this approach, The model is trained by a generator network that is responsible for creating perturbation for that model during the training. The model learns to classify both the clean input and adversarial examples. It is a common trend now to train ML models using this approach. Therefore, it could be the first layer of defense even for the RPS model to increase its robustness against Cloak & Co-locate attack.
3) *Rectify*[13]: After introducing universal perturbations, Akhtar et al.[13] proposed a defense framework that inserts an extra layer before the input called "pre-input." The purpose of the pre-input layer is to train the target model to rectify an adversarial example in a way that the model's classification becomes similar to its classification of the original input. These pre-input layers will be trained without modifying the parameters of the original model. Moreover, rectify trains a perturbations detector for the training inputs by extracting features from the differences between the input and output of the pre-input layer. In this way, when a perturbation is detected, rectify uses the output of the pre-input layer for classifying the input. Since rectify is equipped with the perturbation detector, it can be a good candidate for defense in the RPS domain, especially against the Cloak & Co-locate attack. As soon as the perturbation is detected, a new feature will be used for the classification that will alleviate the impact of the FTG.

Other techniques proposed for malware detection, like ensemble or dimensionality reduction (PCA), can be applied to the RPS domain.[16] Based on our quick survey, most of the current solutions cannot be directly used in RPS and need modifications. Few of them are not effective in the RPS domain at all, and therefore, we conclude that this domain requires more attention from academic research groups.

CONCLUSION

In this work, we explore the effectiveness of defense techniques against Cloak & Co-locate—a novel approach to improve the effectiveness of distributed attacks on cloud infrastructure. For this purpose, we first demonstrated that by reverse-engineering the resource provisioning system and employing the adversarial machine learning attack, the adversary VM can

Co-locate itself with the victim and evade detection, as well as migration caused by the scheduler. Then we used an FTG (Cloak) and wrapped it around the adversary kernel. The FTG is spawned as a separate thread, generating a pattern close to the victim VM's pattern, fooling the scheduler to Co-locate it with the victim VM. After co-location, FTG continuously crafts new behavior to disguise itself and fool the RPS into remaining co-located on the same host as the victim. We then selected the most popular defense techniques against adversarial attacks and analyzed them to check if they are suitable to mitigate the RPS vulnerabilities against the Cloak & Co-locate attack.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under Awards 2155002 and 2155029.

REFERENCES

1. H. M. Makrani et al., "Cloak and co-locate: Adversarial railroading of resource sharing-based attacks on the cloud," in *Proc. IEEE Int. Symp. Secure Private Execution Environ. Des. (SEED)*, 2021, pp. 1–13, doi: 10.1109/SEED51797.2021.00011.
2. H. M. Makrani et al., "Adaptive performance modeling of data-intensive workloads for resource provisioning in virtualized environment," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 5, no. 4, pp. 1–24, Mar. 2021, doi: 10.1145/3442696.
3. C. Fang et al., "REPTTACK: Exploiting cloud schedulers to guide co-location attacks," in *Proc. Netw. Distrib. Syst. Secur. (NDSS) Symp.*, 2022, pp. 1–15.
4. C. Fang et al., "HeteroScore: Evaluating and mitigating cloud security threats brought by heterogeneity," in *Proc. Netw. Distrib. Syst. Secur. (NDSS) Symp.*, 2023, pp. 1–15.
5. K. N. Khasawneh, N. Abu-Ghazaleh, D. Ponomarev, and L. Yu, "RHMD: Evasion-resilient hardware malware detectors," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2017, pp. 315–327, doi: 10.1145/3123939.3123972.
6. S. Sankaranarayanan, A. Jain, R. Chellappa, and S. N. Lim, "Regularizing deep networks using efficient layerwise adversarial training," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, doi: 10.1609/aaai.v32i1.11688.
7. G. K. Dziugaite, Z. Ghahramani, and D. M. Roy, "A study of the effect of jpg compression on adversarial images," 2016, *arXiv:1608.00853*.
8. V. Zantedeschi, M.-I. Nicolae, and A. Rawat, "Efficient defenses against adversarial attacks," in *Proc. 10th ACM Workshop Artif. Intell. Secur.*, 2017, pp. 39–49, doi: 10.1145/3128572.3140449.
9. S. Gu and L. Rigazio, "Towards deep neural network architectures robust to adversarial examples," 2014, *arXiv:1412.5068*.
10. L. Nguyen, S. Wang, and A. Sinha, "A learning and masking approach to secure learning," in *Proc. Int. Conf. Decis. Game Theory Secur.*, Cham, Switzerland: Springer-Verlag, 2018, pp. 453–464, doi: 10.1007/978-3-030-01554-1_26.
11. N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Proc. IEEE Symp. Secur. Privacy (SP)*, 2016, pp. 582–597, doi: 10.1109/SP.2016.41.
12. J. Gao, B. Wang, Z. Lin, W. Xu, and Y. Qi, "DeepCloak: Masking deep neural network models for robustness against adversarial samples," 2017, *arXiv:1702.06763*.
13. N. Akhtar, J. Liu, and A. Mian, "Defense against universal adversarial perturbations," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2018, pp. 3389–3398.
14. H. Lee, S. Han, and J. Lee, "Generative adversarial trainer: Defense to adversarial perturbations with gan," 2017, *arXiv:1705.03387*.
15. W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," 2017, *arXiv:1704.01155*.
16. K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial perturbations against deep neural networks for malware classification," 2016, *arXiv:1606.04435*.

NAJMEH NAZARI is a Ph.D. candidate in the Electrical and Computer Engineering Department, University of California Davis, Davis, CA, 95616, USA. Her research interests include deep learning, embedded systems, and hardware security. Nazari received her M.S. degree in computer engineering from Isfahan University of Technology, Iran. Contact her at nnazari@ucdavis.edu.

HOSEIN MOHAMAMDI MAKRANI is a system-on-chip engineer at Apple, Inc., Cupertino, CA, 95014, USA. His research interests include applied machine learning, cloud computing, and hardware acceleration. Makrani received his Ph.D. degree from the University of California Davis. Contact him at hmakrani@ucdavis.edu.

CHONGZHOU FANG is a Ph.D. candidate in the Electrical and Computer Engineering Department, University of California Davis, Davis, CA, 95616, USA. His research interests include cloud security and edge device security. Fang received his bachelor's degree in computer engineering from

Southeast University, Nanjing, China. Contact him at czfang@ucdavis.edu.

BEHNAM OMIDI is a Ph.D. candidate in the Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA, 22030, USA. His research interests include detecting malicious activity on hardware, micro-architectural side-channel attacks, and designing circuits for supporting hardware security. Omidi received his M.S. degree in computer engineering from Amirkabir University of Technology, Iran. Contact him at bomidi@gmu.edu.

SETAREH RAFATIRAD is an associate professor in the Department of Computer Science, University of California Davis, Davis, CA, 95616, USA. Her research interests include applied machine learning, and Internet of Things security. Rafatirad received her Ph.D. degree in computer science from the University of California Irvine. Contact her at srafatirad@ucdavis.edu.

HOSSEIN SAYADI is an assistant professor in the Department of Computer Engineering and Computer Science, California State University Long Beach, Long Beach, CA, 90840, USA. His research interests include hardware security and machine learning. Sayadi received his Ph.D. degree in computer engineering from George Mason University. Contact him at hossein.sayadi@csulb.edu.

KHALED N. KHASAWNEH is an assistant professor in the Electrical and Computer Engineering Department, George Mason University, Fairfax, VA, 22030, USA. His research interests include architecture support for security. Khasawneh received his Ph.D. degree in computer science from the University of California Riverside. Contact him at kkhasawn@gmu.edu.

HOUMAN HOMAYOUN is a professor in the Department of Electrical and Computer Engineering, University of California Davis, Davis, CA, 95616, USA. His research interests include machine learning, security, and computer architecture. Homayoun received his Ph.D. degree in electrical and computer engineering from the University of California Irvine. Contact him at hhomayoun@ucdvais.edu.