Singapore Management University

Institutional Knowledge at Singapore Management University

Research Collection School Of Computing and Information Systems

School of Computing and Information Systems

9-2020

Group Instance: Flexible co-location resistant virtual machine placement in IaaS clouds

Vu Duc LONG

Nguyen Binh Duong TA

Group Instance: Flexible Co-Location Resistant Virtual Machine Placement in IaaS Clouds

Vu Duc Long
School of Computer Science and Engineering
Nanyang Technological University
long018@e.ntu.edu.sg

Ta Nguyen Binh Duong
School of Information Systems
Singapore Management University
donta@smu.edu.sg

***Abstract*—This paper proposes and analyzes a new virtual machine (VM) placement technique called *Group Instance* to deal with co-location attacks in public Infrastructure-as-a-Service (IaaS) clouds. Specifically, Group Instance organizes cloud users into groups with pre-determined sizes set by the cloud provider. Our empirical results obtained via experiments with real-world data sets containing million of VM requests have demonstrated the effectiveness of the new technique. In particular, the advantages of Group Instance are three-fold: 1) it is simple and highly configurable to suit the financial and security needs of cloud providers, 2) it produces better or at least similar performance compared to more complicated, state-of-the-art algorithms in terms of resource utilization and co-location security, and 3) it does not require any modifications to the underlying infrastructures of existing public cloud services.**

***Index Terms*—cloud security, co-location attacks, virtual machine placement.**

I. Introduction

In Infrastructure-as-a-Service clouds [1], large amounts of computing power are accessible to ordinary users as a service, in which anyone can request for computing resource on-demand without significant upfront expenditure. This service is typically served in the form of Virtual Machines (VM) that are created upon requests. Each VM is initialized and placed into one of many Physical Machines (PM) owned by the IaaS cloud provider. The physical resource of each PM is divided into isolated partitions by a VM monitor (hypervisor). Then, each of the partitions will be used to host a VM accordingly. The assignment of these VMs to PMs is controlled by the cloud provider through appropriate VM placement algorithms [2].

Although there are strong isolation mechanisms to ensure that no data would be shared between different VMs; if the VMs are in the same PM, they still have to share several types of common resource in the PM such as the Last-Level Cache, etc. Such resource sharing enables certain side channels which could leak data across VMs located on the same PM. This in turn motivates various kinds of co-location based attacks, which first involve getting a rogue VM into a particular PM hosting multiple other ordinary VMs. From there, the rogue VM might be able to extract confidential information and data from the victim VMs via existing or new side channels [3].

One notable example was demonstrated by Ristenpart et al. [4] in an empirical study of such attacks in Amazon EC2 cloud service. The researchers were able to recreate a map of the underlying physical cloud resources using network probes and to determine the co-location status of any VMs. Such information was then used to launch rogue VMs that could co-locate themselves with a target VM, of which data could be subsequently extracted via some side-channels. Co-location attacks have gained more attention as recent vulnerabilities such as Spectre [5] and Meltdown [6] critically impact major public cloud providers.

To deal with the threat of co-location attacks, a straightforward approach is to separate each user physically. This method has been practically used by current cloud service providers, e.g., Amazon EC2's Dedicated Instance (DI) [7], [8]. The obvious downside for this approach is low physical resource utilization, as VMs belong to different users could not be consolidated on the same PM. As a result, Dedicated Instance could be significantly more expensive compared to other types of VMs.

Popular approaches in this area focus on improving both resource utilization and co-location resistance of the VM placement algorithms at the same time. Better resource utilization could bring significant financial benefits for both cloud users and providers. Notable approaches include Previously Selected Server First (PSSF) by Hans et. al. [9] and Previously Co-located User First (PCUF) [10], [11]. More specifically, PSSF limits the number of different PMs that a VM from a particular user can be placed on. On the other hand, PCUF, which is one of the most recent approaches, aims to reduce the chance for malicious co-location by only letting VMs of previously co-located users to share the same PM.

In this paper, we propose a new approach called Group Instance (GI), in which we also aim to reduce the chance of malicious co-locations. In essence, GI introduces a new configurable parameter, $group\_limit$, to maintain a balance between security and resource utilization. With this new parameter, GI could be tuned to achieve comparable or even better performance in terms of resource utilization or co-location resistance compared to Dedicated Instance, PSSF, or PCUF in most practical cases. Compared to state-of-the-art approaches like PCUF, GI has several enhancements: 1) the cloud provider can easily control of the balance between co-location security and resource utilization (which directly affects the resource cost), 2) it is simpler to analyze and implement, and 3) it

2641-8169/20/$31.00 ©2020 IEEE
DOI 10.1109/WETICE49692.2020.00021

provides better or similar performance compared to existing approaches most of the time, as demonstrated in the empirical experiments using real workload traces from Azure [12].

The remaining part of the paper is organized as follows. In Section II, we discuss the most recent related work. In Section III, we elaborate the problem statement. We then describe our VM placement algorithm and its theoretical analysis in Section IV. Section V describes our empirical study of the algorithm. Finally, Section VI discusses the limitations of our research and some directions for future work.

II. Related Work

We broadly classify existing work in defending against co-location based attacks into two categories, namely direct and indirect approaches.

A. Direct approaches

These approaches hinder malicious users directly by applying changes or patches into the existing cloud infrastructure to remove any known side channels in the platform. Such changes may make it much harder for any attackers when they try to achieve co-locations with other users. Even if they manage to obtain co-location, the potential impacts could be limited as most known side channels have been inhibited. Such modifications can be done at many levels [13], for example at the CPU cache [14], [15], network layer [16], or at the hypervisor layer [17], [18].

As a result, direct approaches have been quite effective at eliminating most well-known cloud attack strategies based on co-location. However, there are two clear downsides to this category of defenses:

- They require significant modifications to the production cloud infrastructures. Such changes are not easy to carry out due to potential downtime and cost, given the scale of current commercial cloud providers.
- They might not work well against unknown vulnerabilities. Most recent work such as [19] and [20] have demonstrated that new side-channels will keep appearing; and it is almost impossible to eliminate all of them.

B. Indirect approaches

On the other hand, indirect approaches aim to reduce the chance that a malicious user can be co-located with other ordinary users on the same PM. If it is not possible to achieve co-location, it would be much harder to carry out subsequent attacks. Normally, co-location prevention could be obtained using either: i) some co-location resistant VM placement algorithms such as [2], [9], and [21]; or ii) live, random VM migration strategies so that it becomes hard for anyone to locate a particular target VM, as in [22] and [23]. We note that live migration approaches may negatively affect applications running on the cloud; while co-location resistant VM placement could reduce resource utilization.

Compared to direct approaches, the main issue here is that indirect approaches are not able to prevent potential side-channel attacks once malicious users have been successfully co-located with their victims. Another problem could be that a resourceful attacker with a large budget could launch many VM requests, or employ an increasing number of accounts until he can achieve co-location. However, this type of approaches might still be of interest for two main reasons: i) it does not require significant changes to existing cloud platforms, and ii) it is more likely to be resilient against arbitrary and currently unknown attacks.

Our proposed algorithm, Group Instance, belongs to the category of indirect approaches. Compared to existing methods in the same category, Group Instance stands out due to its ability to handle resourceful attackers more effectively: a single user could only co-locate with a predefined, fixed number of other users. It has been demonstrated empirically in our experiments that when the number of malicious users increases, the Group Instance algorithm with an appropriate group size remains quite resilient compared to most recent approaches like PCUF [10], [11].

III. Problem Statement

A. Assumptions

We assume that a multi-tenant public IaaS cloud service has a known number of users. The users of the service can be categorized into one of two types: benign or malicious. Benign users are the ordinary cloud users whose purpose is not to steal data from other users, while the malicious users would try to co-locate with other users to access or to compromise the victim's data. It is assumed that the cloud provider has no knowledge about the classification of any particular user; therefore it is not possible for the provider to impose certain restrictions on potentially malicious users.

The cloud service provider is the only one who can assign VMs to PMs. All VM requests arrive in a sequential manner, characterized by the ID of requesting user, the number of CPU cores and the amount memory for each request. The duration (lifetime) of each VM which includes its start time and end time is not known at VM placement time. We also assume that each VM will stay on the same PM until it is terminated, i.e., no live migration of the VM.

B. Co-location resistant VM placement

We use the same set of metrics originally proposed in [10] for quantifying the performance of VM placement algorithms. The objective of the **Co-Location Resistant VM Placement** (CVP) problem is to assign each newly provisioned VM to the available PMs in a way which maximizes both the resource utilization CU and the co-location resistance CLR.

1) Core Utilization (CU): The resource utilization of a PM could be measured using several factors, including memory utilization. In this paper, we measure the utilization by comparing actual running time of all CPU cores to their total active time, i.e., core utilization. We note that in [24], the authors demonstrated that the energy consumption of CPUs exceeds all the other computer hardware. Note that the total active time of a CPU core is basically the time during which its PM is on.

The actual running time of a CPU core is the duration during which the core is occupied by a VM.

2) Co-Location Resistance (CLR): Following [10], we consider two states for a user, which are safe and unsafe, respectively. A user is in the safe state if none of his/her VMs is co-located with at least one malicious user during the lifetime of the VMs. The Co-Location Resistance CLR is computed as the ratio of all the benign users who are safe to all benign users in the cloud. When CLR = 1, all the benign users are safe.

IV. The Group Instance (GI) Algorithm

A. Algorithm

The GI algorithm is invoked whenever a new VM request arrives. Its input parameters include the newly arrived VM V_i, a list of PMs that are currently active $open\_pms$ and a list of empty PMs $empty\_pms$. U_{cur} is the requester of V_i. The basic idea is that we aim to limit the number of users that a user can be co-located with. This is done by dividing all cloud users into a number of small groups with a fixed size $group\_limit$. This simple strategy could limit the extent of damage should a user is malicious. In addition, by controlling the variable $group\_limit$, we can easily fine tune the trade-off between CU and CLR. Below are the key steps in the GI algorithm:

- First, verify whether the U_{cur} is a new cloud user or not by checking if he/she already has a group index G_{U_i}. If he/she is not a new cloud user, we construct a list of $eligible\_pms$ by including all the PMs from $open\_pms$ satisfying two conditions: i) have enough resource to host V_i, and ii) currently host VMs belonging to U_{cur}'s group. If the $eligible\_pms$ list is not empty, we assign V_i to a PM belonging to $eligible\_pms$ which has the least number of free cores. Otherwise, we assign V_i to a random and empty PM, and mark that PM as occupied by U_{cur}'s group.
- If U_{cur} is a new cloud user, we construct a list of $eligible\_pms$ which includes PMs satisfying the following conditions: i) the PM is currently occupied by a group that is not full, i.e. the size of this group is smaller than the $group\_limit$, and ii) the PM still has enough resource to host V_i. If this $eligible\_pms$ list is not empty, we assign V_i to a PM having the smallest group size selected from $eligible\_pms$. On the other hand, if $eligible\_pms$ is empty, we create a new group G_{new} for U_{cur}, and assign V_i to an empty PM P_k.
- When a VM V_i is terminated from PM P_j, if there are no other VMs on P_j, we put P_j back into the list $empty\_pms$. We also clear the group index G_{P_j} currently assigned to P_j.

B. Theoretical Analysis

In this section, we derive the theoretical CLR values obtained by the GI algorithm.

1) Worst-case CLR: In this case, each individual malicious user would be assigned to a full group, i.e., a group with the same size as $group\_limit$. Thus, the malicious user has successfully achieved co-location with $group\_limit - 1$ other users. Hence, the number of benign users that are safe N_b^{safe} is calculated as N_b - N_m * ($group\_limit$ − 1), where N_b and N_m are the numbers of benign users and malicious users, respectively. As the total number of users who are safe cannot be negative, we have:

$$CLR = \frac{max(N_b - N_m * (group\_limit - 1), 0)}{N_b} \tag{1}$$

2) Expected CLR: Following the methodology suggested in [10], we derived a simple formula[1] to estimate the average CLR for the GI algorithm:

$$CLR_{GI}^{average} = \frac{N_b \times \left(\frac{N_b - 1}{N-1}\right)^{group\_limit - 1}}{N_b} = \left(\frac{N_b - 1}{N-1}\right)^{group\_limit - 1} \tag{2}$$

We provide an example on how to make use of the derived formula to estimate the CLR. Assume that there are 100 users, and 2% of them are malicious, we have N = 100 and N_b = 98. If we choose $group\_limit$ = 3 and use Equation (2), we will have CLR = 96.00%. However, a potential shortcoming of the derived formula is that with a large $group\_limit$, the accuracy of the estimated CLR would deteriorate due to the fact that it is less likely on average for group sizes to reach the limit.

The formula (2) can be easily generalized for the case where cloud users can have different $group\_limit$ assigned by the cloud provider. In this case, we divide N users into sub-groups based on their value of $group\_limit$. The next step is to calculate the number of users who are safe in each sub-group. Finally, the expected total number of safe users in the system is the sum of the results from all sub-groups.

V. Evaluation and Results

A. Methodology

Similar to [10], we use the dataset which has been released recently by Microsoft Azure [12]. The dataset contains a total of 2,013,767 VMs request over a period of 30 days. In this paper, we present the results obtained from using all VM requests from the 11th day to 20th day of the dataset, as the findings are quite similar for other periods of the original dataset. We make use of the following information in evaluating VM placement algorithms: VM id, subscription id (considered a distinct cloud user), VM start time, VM stop time, VM core count and VM memory (in GB). In total, there are 619846 VM requests and 1884 different cloud users during the 10-day duration used in this evaluation. We also assume that the core count and memory of all PMs in the system are the same: 32 cores and 224GB main memory for each PM.

[1] The calculation details have been omitted due to a restriction on the number of pages for this paper.

The largest VMs in the Azure dataset have 16 CPU cores and 112GB of memory.

Since the Azure dataset has no information regarding malicious users, for each run we randomly select $P_m\%$ of all users to be malicious. We repeat each algorithm over the 10-day dataset 20 times. We then aggregate CU and CLR values over all experiments, and use the averages for further analysis.

B. GI algorithm - impacts of different values for *group_limit*

In the following, we use the notation GI_x^t where x indicates the value for $group\_limit$ to denote the results derived from the theoretical formula (2). For example, GI_3^t denotes the theoretical results for GI with $group\_limit$ = 3. On the other hand, GI_x denotes the empirical results obtained with $group\_limit$ = x.

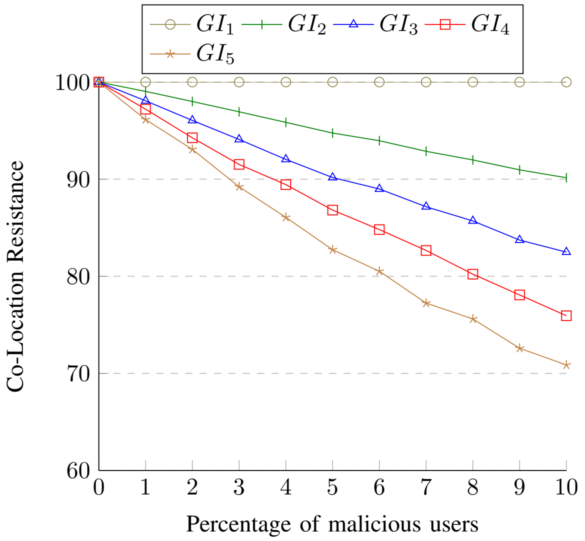

Fig. 1: Comparing CLR for different values of $group\_limit$. Smaller limits provide better co-location resistance, at the cost of lower resource utilization. $group\_limit$ = 1 has the same CLR as the Dedicated Instance approach.

TABLE I: CU values for the GI algorithm, obtained by varying $group\_limit$

group_limit	1	2	3	4	5
CU (%)	61.9	66.98	69.40	71.52	72.87

From Fig. 1 and Table I, we can observe that when $group\_limit$ increases, the value of CU will increase, while the value of CLR decreases. This is due to the fact that smaller group sizes result in less chance for malicious co-location. Thus, by setting appropriate values for $group\_limit$, we would be able to fine-tune the trade-off between CU and CLR if needed. We also observe that by increasing $group\_limit$, we can improve resource utilization, but only up to a certain extent. Table I shows that $group\_limit$ values beyond 3 do not significantly increase CU.

C. GI algorithm - theoretical vs. empirical performance

Fig. 2 summarizes the difference between results obtained empirically and those produced by the derived equation (2). The data indicate that for small values of $group\_limit$, the empirical and theoretical results are very similar. However, while there is negligible difference when $group\_limit$ = 2, the gap appears to become wider as this limit increases. When $group\_limit$ = 5, the difference is quite noticeable.

One possible reason for this difference is that we assume the average user group size to be the same or close to $group\_limit$ in our theoretical model. However, the actual average group size will be less likely to reach the the larger $group\_limit$ most of the time. Due to smaller average group sizes, the CLR values obtained in the empirical experiments are actually better compared to those obtained theoretically. Therefore, the theoretical CLR values may serve as indications of a lower bound for the co-location resistance performance of our algorithm.

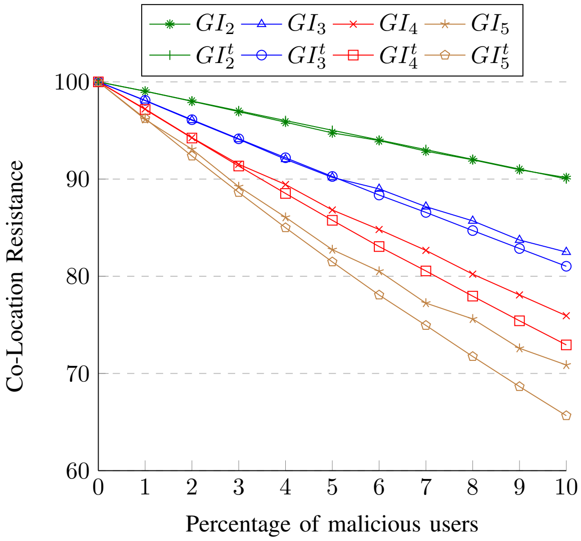

Fig. 2: Comparing CLR values obtained empirically and theoretically. We note that the gaps between theoretical and empirical values are not significant especially when the $group\_limit$ is small.

D. GI algorithm - compared to the state-of-the-art approaches

In this section, we compare our proposed placement strategy GI with the latest co-location resistant VM placement algorithms, namely: i) Amazon EC2's Dedicated Instance (DI) placement [7], and ii) Previously Co-located User First (PCUF) by Amit. et. al. [10], [11]. Note that we do not compare GI with Previously Selected Servers First (PSSF) by Han et al. [9], as [10] and [11] have thoroughly demonstrated that PCUF has a clear advantage.

1) DI approach: Two VMs that belong to different AWS accounts are not placed on the same PM. Therefore, for allocating a VM V_i requested by user U_j, we only consider those PMs which currently host U_j's VMs. Among all eligible PMs, we will fit U_j's new VM request using the Best-Fit bin

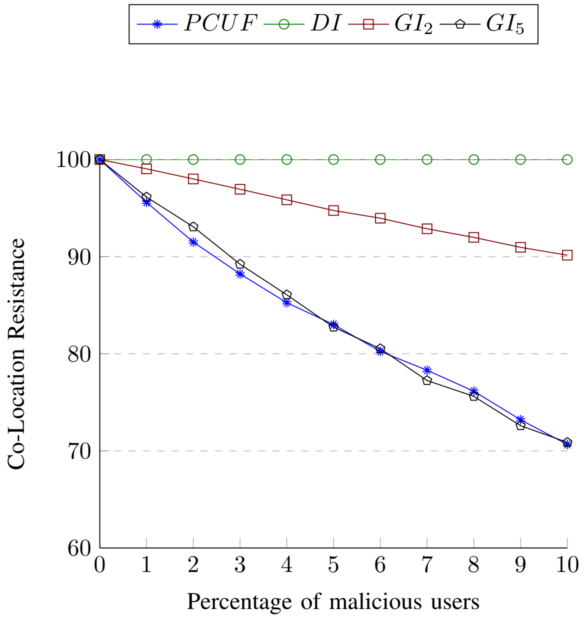

Fig. 3: CLR performance for different co-location resistant placement algorithms - GI, DI and $PCUF$.

packing approach. A new PM is started if the current PMs do not have enough resource to take in the new VM request.

2) PCUF: Most recently, Amit et. al. [10], [11] proposed a new placement strategy based on the co-location history of cloud users. In this approach, each user has a log file that records a peer list containing users whose VMs have been placed in the same PM previously. For each VM request, the PCUF algorithm will favor the PM which is hosting some VMs owned by the same user himself or his peers. If no such PM is found, a new empty PM would be started. If the user is new to the system, his first VM would be assigned to a randomly selected and running PM, or a brand-new PM.

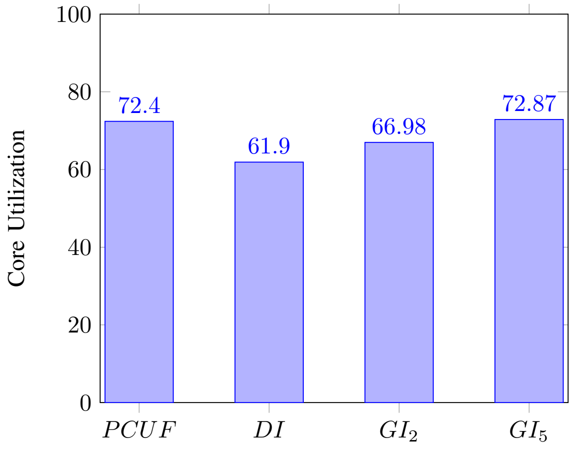

Fig. 4: CU performance for different co-location resistant placement algorithms. Note that CU does not depend on the percentage of malicious users in the system.

Fig. 3 and Fig. 4 show the CLR and CU respectively, for different co-location resistant placement algorithms on the Azure workload with varying percentage of malicious users. Below are some important observations from the obtained results:

- DI has the same performance in terms of CLR and CU as GI_1. This is because when *group_limit* is set to 1, no cloud user would be co-located with another user. In other word, DI can be considered a special case of GI.
- The overall performance of GI when *group_limit* = 5 is surprisingly close to that of PCUF. One possible reason is that in the Azure workload, the average number of co-located peers for each user is similar to our average group size when *group_limit* is set to 5.
- GI_2 has much better CLR performance compared to PCUF, especially when the percentage of malicious users in the system is high. For instance, when %10 of the users are potential attackers, GI_2 shows a CLR improvement of 30% compared to PCUF. This is at the cost of slightly lower CU performance for GI_2. We also note that GI_2 has better CU compared to DI.

From these observations, we conclude that our proposed GI algorithm can achieve better or at least similar overall performance compared to state-of-the-art VM placement policies. In addition, our new algorithm also provides the opportunity to fine-tune the trade-off between resource utilization and co-location resistance with ease. This can be done via setting the appropriate values for the *group_limit* parameter. This feature could be of great practical implications for the following reason. As the amount of malicious users increases with the popularity of a cloud service, static algorithms such as PCUF would produce less co-location resistance, as evidenced in Fig. 3. In contrast, GI is more adaptive to such a change: we can simply reduce the *group_limit* value and achieve a better CLR.

For instance, if the cloud provider observes that potentially malicious activities are increasing via tools such as CloudRadar [25], the *group_limit* should be set to smaller values such as 3 or 2. On the other hand, a larger limit such as 5 can be used to achieve better resource utilization. Our GI algorithm can be configured so that a mix of different *group_limit* values could also be used at the same time. Fig. 5 compares the CLR in such a scenario, where 20% of the users[2] in our experiments are given a $group\_limit = 2$, while the rest of them use $group\_limit = 5$. We observe that GI - due to its flexibility to switch between different group limits - clearly outperforms PCUF in this case.

VI. Conclusion

In this paper, we propose and evaluate a new co-location resistant VM placement algorithm named Group Instance. We then conduct performance analysis of the new approach using resource utilization and co-location resistance as the

[2] These users could be identified by the cloud provider as the current targets of some on-going co-location attacks.

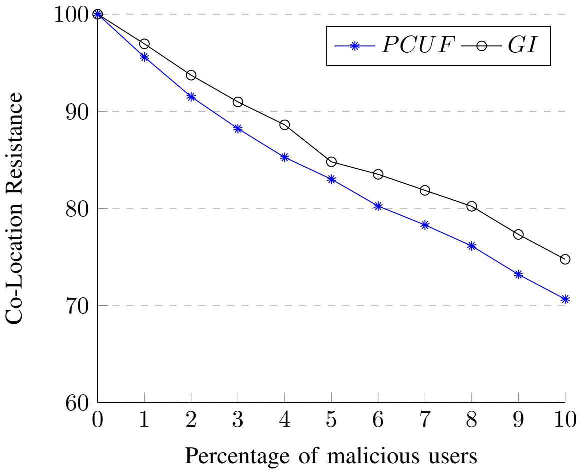

Fig. 5: CLR performance for different co-location resistant placement algorithms - GI using two $group\_limit$ values of 2 and 5, and $PCUF$.

metrics. Both the theoretical and empirical evaluation of Group Instance demonstrate that it can achieve better overall performance compared to the latest published approaches. On top of that, our new algorithm has the distinct advantage of being simple to implement and easily tuned to adapt to the changing landscape of cloud threats. We believe that this advantage makes Group Instance more desirable in practical settings.

We plan to extend this work in several directions. First, if cloud providers are able to detect on-going co-location attacks with certain accuracy, a complement VM migration policy in combination with our algorithm might provide better co-location resistance and resource utilization. Another direction could be developing a classifier to determine if a user is potentially malicious or not with certain probabilities. If such information is available, we believe that Group Instance would be able to obtain much better overall performance by grouping users that are more likely to be malicious.

References

[1] Y. Duan, G. Fu, N. Zhou, X. Sun, N. C. Narendra, and B. Hu, "Everything as a service (xaas) on the cloud: origins, current and future trends," in *2015 IEEE 8th International Conference on Cloud Computing*, pp. 621–628, IEEE, 2015.

[2] Y. Azar, S. Kamara, I. Menache, M. Raykova, and B. Shepard, "Co-location-resistant clouds," in *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security*, pp. 9–20, ACM, 2014.

[3] Y. Zhang, Y. Mao, M. Xu, F. Xu, and S. Zhong, "Towards thwarting template side-channel attacks in secure cloud deduplications," *IEEE Transactions on Dependable and Secure Computing*, 2019.

[4] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*, pp. 199–212, ACM, 2009.

[5] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre Attacks: Exploiting Speculative Execution," *ArXiv e-prints*, Jan. 2018.

[6] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown," *ArXiv e-prints*, Jan. 2018.

[7] "AWS EC2 Dedicated Instance." https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/dedicated-instance.html.

[8] N. B. D. Ta and N. Pimpalkar, "Handling co-resident attacks: A case for cost-efficient dedicated resource provisioning," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pp. 849–852, IEEE, 2018.

[9] Y. Han, J. Chan, T. Alpcan, and C. Leckie, "Using virtual machine allocation policies to defend against co-resident attacks in cloud computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 1, pp. 95–108, 2017.

[10] A. Agarwal and T. N. B. Duong, "Co-location resistant virtual machine placement in cloud data centers," in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 61–68, IEEE, 2018.

[11] A. Agarwal and T. N. B. Duong, "Secure virtual machine placement in cloud data centers," *Future Generation Computer Systems*, vol. 100, pp. 210–222, 2019.

[12] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," in *Proceedings of the 26th Symposium on Operating Systems Principles*, pp. 153–167, ACM, 2017.

[13] O. Weisse, I. Neal, K. Loughlin, T. F. Wenisch, and B. Kasikci, "Nda: Preventing speculative execution attacks at their source," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 572–586, ACM, 2019.

[14] F. Liu and R. B. Lee, "Random fill cache architecture," in *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, pp. 203–215, IEEE, 2014.

[15] M. Werner, T. Unterluggauer, L. Giner, M. Schwarz, D. Gruss, and S. Mangard, "Scattercache: thwarting cache attacks via cache set randomization," in *28th USENIX Security Symposium*, pp. 675–692, 2019.

[16] E. Pattuk, M. Kantarcioglu, Z. Lin, and H. Ulusoy, "Preventing cryptographic key leakage in cloud virtual machines.," in *USENIX Security Symposium*, pp. 703–718, 2014.

[17] P. Li, D. Gao, and M. K. Reiter, "Stopwatch: a cloud architecture for timing channel mitigation," *ACM Transactions on Information and System Security (TISSEC)*, vol. 17, no. 2, p. 8, 2014.

[18] V. Varadarajan, T. Ristenpart, and M. M. Swift, "Scheduler-based defenses against cross-vm side-channels.," in *USENIX Security Symposium*, pp. 687–702, 2014.

[19] W. He, W. Zhang, S. Das, and Y. Liu, "Sgxlinger: A new side-channel attack vector based on interrupt latency against enclave execution," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, pp. 108–114, IEEE, 2018.

[20] H. Naghibijouybari, A. Neupane, Z. Qian, and N. Abu-Ghazaleh, "Rendered insecure: Gpu side channel attacks are practical," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2139–2153, ACM, 2018.

[21] Y. Han, T. Alpcan, J. Chan, and C. Leckie, "Security games for virtual machine allocation in cloud computing," in *International Conference on Decision and Game Theory for Security*, pp. 99–118, Springer, 2013.

[22] Y. Zhang, M. Li, K. Bai, M. Yu, and W. Zang, "Incentive compatible moving target defense against vm-colocation attacks in clouds," in *IFIP International Information Security Conference*, pp. 388–399, Springer, 2012.

[23] S.-J. Moon, V. Sekar, and M. K. Reiter, "Nomad: Mitigating arbitrary cloud side channels via provider-assisted migration," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1595–1606, ACM, 2015.

[24] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," *Cluster computing*, vol. 12, no. 1, pp. 1–15, 2009.

[25] T. Zhang, Y. Zhang, and R. B. Lee, "Cloudradar: A real-time side-channel attack detection system in clouds," in *International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 118–140, Springer, 2016.