MIGRATE: Towards a Lightweight Moving-target Defense against Cloud Side-Channels

Mohamed Azab[1]
The City of Scientific Research and Technological Applications, Alexandria, Egypt
ACIS , Electrical and Computer Engineering Department, University of Florida, Gainesville, USA
mazab@vt.edu

Mohamed Eltoweissy
Department of Computer and Information Sciences
Virginia Military Institute, USA
The Bradley Department of Electrical and Computer Engineering, Virginia Tech ,USA
eltoweissymy@vmi.edu

***Abstract*—Recent research has demonstrated the severity of co-residency side-channel attacks on computing clouds. These attacks have been successfully employed by malicious tenants to extract sensitive private information from selected neighboring tenants. Solutions towards addressing such attacks have presented customized solutions for specific variants of these attacks that often require significant modifications to the hardware, client virtual machines (VM), or hypervisors. These solutions are not generic and will not succeed with mutating versions of these attacks. Except for the impractical, resource inefficient, and costly single tenant solutions, co-residency will always be an issue to cloud service providers. In this paper, inspired from the camouflaging process of the sea chameleons evading predators, we present MIGRATE. MIGRATE is a container management framework that employs resource-efficient, scalable, real-time moving target defense to obfuscate the container execution behavior complicating the attacker's task to locate their targets. MIGRATE, offers generic defense against side-channel attacks and employs efficient real-time probabilistic random migrations of cloud tenants' applications contained in Linux containers between different hosts. To minimize the probability of attacker-victim co-residency on the same host. Eliminating the stable co-residency issue eliminates most of the side-channel attacks that face such a platform. Given the current implementation of MIGRATE tested on VMware V-Sphere Cloud, results showed that it can induce high frequency migrations with almost no effect on the enclosed applications making it suitable for mission-critical applications and as a mitigation against fast side-channel attacks.**

Index Terms - Cloud computing; Cross-VM side-channel attacks; VM migration, Linux containers.

I. INTRODUCTION

Public clouds are offering a much more feasible, cost-efficient, reliable on-demand scalable replacement to the conventional isolated data centers. To effect that, cloud providers share physical resources to support multi-tenancy of cloud platforms. However, the possibility of sharing the same hardware, software, libraries, or filesystem space poses a serious threat unless isolation boundaries are professionally set. Current service providers employ various isolation techniques to set such boundaries. Runtime-based isolation, User-based isolation, Container-based isolation, and VM-based isolation are examples of such techniques [5]. Among this list, VM and container isolation perform the best, offering enough isolation to block many attacks that other techniques cannot mitigate.

Unfortunately, researchers in [5, 7, 8] proved that even VM- and container-based isolation cannot offer enough protection against side channel attacks. Containers and VMs executing on the same physical machine share a range of hardware and software resources. Even when solid logical isolation is deployed by the hypervisor and hosting OS against abuse of explicit logical channels, shared resources open the door to *side-channel attacks.* When tested on public and private clouds [5] employing various isolation techniques, side-channel attacks were largely successful.

In this paper, we focus on side-channels: cross-VM/container information leakage due to the sharing of physical resources with malicious "same host" neighbors (e.g., the CPU's data caches). These are among the most devastating attacks facing cloud computing service providers. To avoid such type of attacks, enterprises with sensitive date, often demand physical isolation for their cloud deployments. However, cloud providers cannot guarantee all-time physical isolation [15] even if at a much higher cost than the logically isolated resources. In the multi-process environment, such attacks have been shown to enable extraction of RSA [16, 17] and AES [16 , 18] secret keys. That attack depends on the attacker ability to place its malicious container on the same host with the victim. That attack requires two main steps: placement and extraction. Placement refers to the adversary arranging to place their malicious VM on the same physical machine as that of a target customer. Recent studies [16] showed that in some attack scenarios, just a few dollars invested in launching VMs can produce a 40% chance of placing a malicious VM on the same physical server as a target customer.

In this paper, we target the placement vector. Inspired by previous work [19,20], our main objective is to excessively complicate the attacker process in successfully place his VM/Container within the same host with his victim for the entire time of the attack. We introduce MIGRATE, a VM/container management framework that offers periodic non-deterministic runtime live-migration for operating VM/containers between various physical hosts inducing enough obfuscation to disable the attacker mission in placing his VMs/containers in the same host with his victim and maintain its placement for the entire time of the attack.

[1] The author is also affiliated with the Smart CI Research Center of Excellence, Alexandria University, Alex, Egypt

Due to the small footprint and fast instantiation, Linux containers are more preferred than VMs by most of the modern clouds. Linux-containers have been presented in many implantations such Linux-VServer (linux-vserver.org), Docker [21], and LXC [22], and OpenVZ [23]. Generally, Docker seems to be the most promising implementation adopted by many commercial clouds as It offers a much faster instantiation and stable operation than the other techniques. MIGRATE was tested on our local test-bed built on a local VMware V-Sphere cloud. Multi-tenancy was offered by employing Docker-based container isolation. We believe that the same approach can be employed on other types of container implementations and VMs. However, in this paper we will present our framework managing that Docker-based infrastructure trying to obfuscate the placement and the operational aspects of the working containers to evade side-channel attacks.

The paper's contribution can be summarized as follows:

1- Container management framework enabling live container migration; and
2- Efficient migration management mechanism to evade attackers with minimal overhead.

The remaining part of the paper is organized as follows: Section II presents a literature review, Section III shows the threat model, Section IV describes the system architecture while Section V discusses the security evaluation, and finally Section VI concludes the paper and outlines future work.

II. Related Work

Cloud computing changed the conventional service delivery model with single tenancy approach to a more resource- and cost-efficient multi-tenancy model with extensive resource sharing. Cloud computing offers shared hardware resources hosting multiple user applications contained in a vitalization capsule. The capsule can extend to a full virtual machine or shrink to include only the needed libraries as in containers [1, 2]. Regardless of the virtualization technology and the employed level of sharing, such sharing exposes hosted tenants to many risks and attacks launched by their neighboring tenants [3, 4].

Cross-side/Covert channel attack is one of the major threats that can be classified under such category. The cryptographic implementations inside virtual machines can be exploited due to its weakness.

Zhang et al [5] presented a cross virtual machine side-channel attack with sufficient granularity to extract some secret keys from the victim. Their technique is a more evolved version of the work presented in [6]. The presented attack enables leveraging the processor caches to observe the victim application execution behavior to allocate the critical values used by the application such as the PRG used to device encryption keys.

Flush+Relod attack is a form of cache-based side-channel attack, which employs a monitoring process. That works in three stages "flushing stage, target accessing stage, and reloading stage".

Irazoqu et al [7] developed a technique to recover cryptographic keys by employing the Flush+Reload technique across the virtual machines that is used to discover if specific cache lines have been accessed or not by observing the code under attack. Using the clflush command, the attacker can flush the desired (shared) memory lines from all the caches of all the (shared) cores in the machine. Yarom et.al [8] stated that the severity of the Flush+Reload attack is based on two properties. First if the attack was successful to exploit memory lines then it will leverage secret data. Second it can access the furthest level of cache from the processors core to reach the LLC.

Zhang et.al [9] proposed a framework that uses Flush-Reload attack in PaaS public clouds. They extended the work presented in [10] employing an automaton-driven strategy for tracing a victim's execution. The framework aims to confirm the tenant co-location and then extract secrets across tenant boundaries.

All these attacks rely mainly on the ability of the attacker to deploy his machine on the same hardware that his victim uses. That can be achieved by employing one of the Co-residency attacks.

Co-residency attack is a placement attack where the attacker tries to identify the victim's machine host and deploy his own machine side by side to the victim on the same host.

Varadarajan et.al [12] proposed a framework that evaluates public clouds vulnerabilities. They showed that three popular public cloud providers (amazon ec2, google cloud, Microsoft azure) are vulnerable to co-location attack. Moreover they revisited the placement issues, and ran some experiments that showed how easy it is for an attacker to control the his malicious VM deployment on the same host that hosts his victim's VM.

Adam et.al [11] have proposed an attack technique based on injecting a watermark signature into the network flow of a target instance. It can be used to ex-filtrate and broadcast co-residency data from the physical machine, compromising isolation without reliance on internal side channels.

Researchers tried to present a set of mechanism and techniques to **mitigate** the side channel attacks or the co-residency problem. Most of these solutions aimed to complicate the targeted placement procedure to obfuscate the sensitive data in the shared memory.

Taesoo et al [13] developed a system-level protection mechanism against cache-based side channel attacks in the cloud called STEALTHMEM. The system can modify popular encryption schemes such as AES, DES and Blowfish. The system aims to lock the pages of a virtual machine in the shared cache to block attacker access.

Stephen [14] proposed a technique to protect running VMs against cache side-channel attacks by diversifying the execution characteristics of the victim application mainly by reloading the cache on random context switches and rewriting encryption routines to avoid optimized lookup tables. The main disadvantage of their work was the sever user involvement and application customization needed to enable their technique.

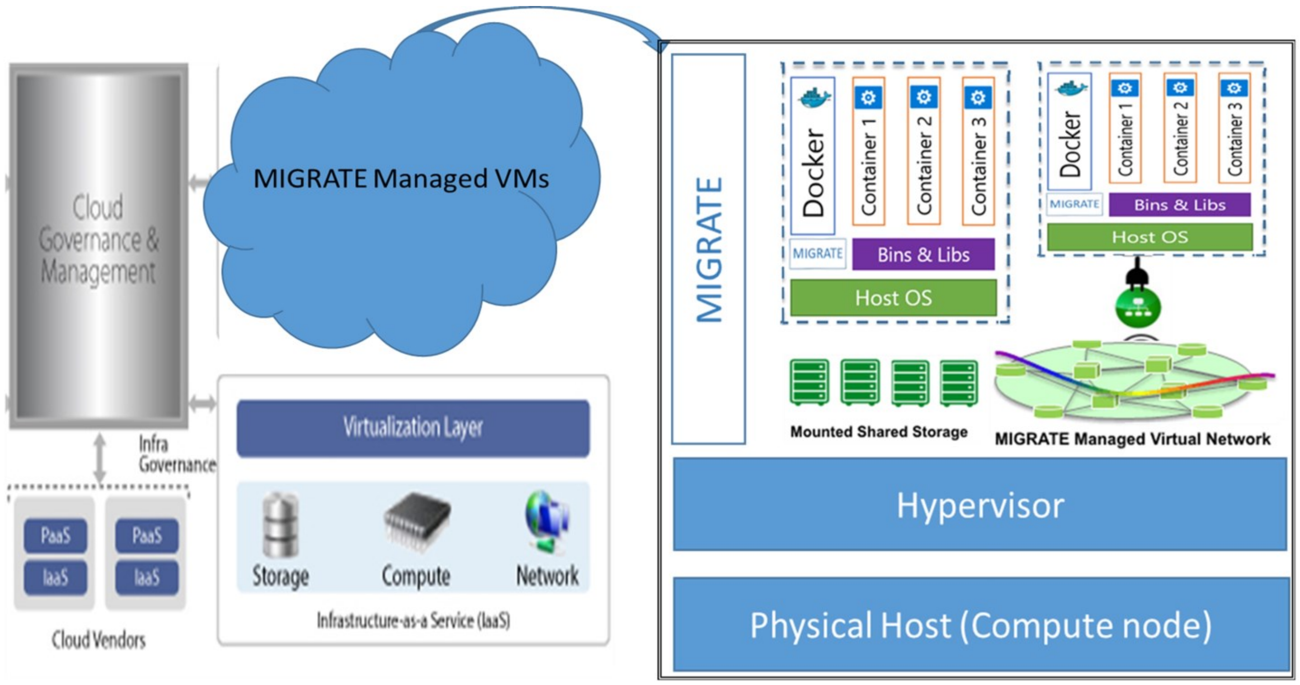


Figure 1. MIGRATE-enabled Architecture

Soo et al [15] proposed a defence system against side-channel attacks, named NOMAD. NOMAD aimed to coordinate the placement and migrating of the virtual machines, to avoid attacker co-residency. NOMAD used a detection tool to allocate information leakage or attacker attempts to cross the boundaries of his VM. In response to that they used provider assisted migration mechanism to move the victim VM away from the attacker. NOMAD is the closest approach to MIGRATE. However, NOMAD is a pure reactive approach that comes with a major cost in terms of application downtime. The migration process for an entire VM is slow and consumes too much resources. That is why the authors acknowledged that their mechanism will not work for mission critical applications, or fast side channel attacks. On the other hand, our MIGRATE relying on containers as a virtualization capsule with the migration mechanism used, it can offer fast and high frequency migration to mitigate fast side-channel attacks with almost zero downtime. Such features enable MIGRATE to operate on containers handling mission-critical applications that cannot tolerate long downtimes.

III. The Threat Model

We use the following threat model and use it to evaluate MIGRATE performance in evading cross-side channel attacks. Assuming that each cloud client (victim) has some private information valid for a certain amount of time. The goal of the adversary is to extract as much information as possible. The attacker operates based on a powerful adversary model with the following characteristics. The adversary is capable of launching a wide spectrum (of possibly unknown) side-channel attacks against other co-resident containers. We also assume that the adversary can determine if/when the target client of interest is co-resident with a VM it owns. The adversary has free control on their containers and has the needed tools to collect data from the shared resources, assemble into meaningful information for their own benefit. We also assume that the information has an expiry date that defines its value. If the adversary managed to collect all the data needed within this lifetime, we call it a success; if not then they will have to start the collection process from the beginning. We did not consider the case were the information has no expiry date or it is valid for long time, "Ex. PGP key, it may be valid for years". In this case MIGRATE will not block the attack. However, it will substantially increase the time needed for a successful attack. Such increase will raise the attacker cost and the chance of detecting his attempts.

Finally, we assume that the adversary does not have explicit control on the cloud management platform to enable them to control the placement of the running virtual machine or containers in the cloud.

IV. The MIGRATE System

MIGRATE was built to be as generic as possible with minimal application customization. The main advantage of using generic tools is to give the user/system administrator the chance to select the most appropriate tools and applications that suits their needs with no constraints or limitations. MIGRATE manages general purpose Linux containers used as a lightweight operating system virtualization technology. We used Docker [2] , an LXC-based container management tool hosted on Linux operating systems to sandbox the users' applications. Docker employs the resource isolation features of the Linux kernel to allow independent containers to run in total isolation from each other and the underlying host. The host kernel isolates the container and the contained applications

views of the operating environment within a single Linux instance including process trees, network, user IDs and mounted file systems, while the (Control Groups) *cgroups* provide resource isolation, including CPU, memory, block I/O and network.
MIGRATE moves running containers between multiple physical hosts. The migration process starts by check-pointing the running container to save whatever work being done inside the container. This feature is not enabled by default by the current hypervisors. To enable check-pointing of running application, we used a check-pointing tool named CRIU [24] to momentarily freeze the running container and its enclosed applications taking a live snapshot of the memory content and any used files. The dumped images are stored in a persistent migrate-able state. The details of this process will be discussed later.
Figure 1 shows MIGRATE-enabled system architecture. Migrate operates with two cooperating agents, one running inside the VM (container host) and another one running either as an external agent managing MIGRATE-operated containers, or as a part of the cloud management platform. The current implementation of MIGRATE presented in this paper realizes the first approach. Further work is needed to build a set of APIs that enable direct communication between MIGRATE control modules and the underlying cloud management platform.
The entire control structure of MIGRATE where the networking, container instantiation, check-point/migrate modules, and the data repository works are hosted in a VM on the cloud managing the other VMs and containers . The task of each module will be described in the next subsections. Figure 2 highlights the abstract architecture of MIGRATE.

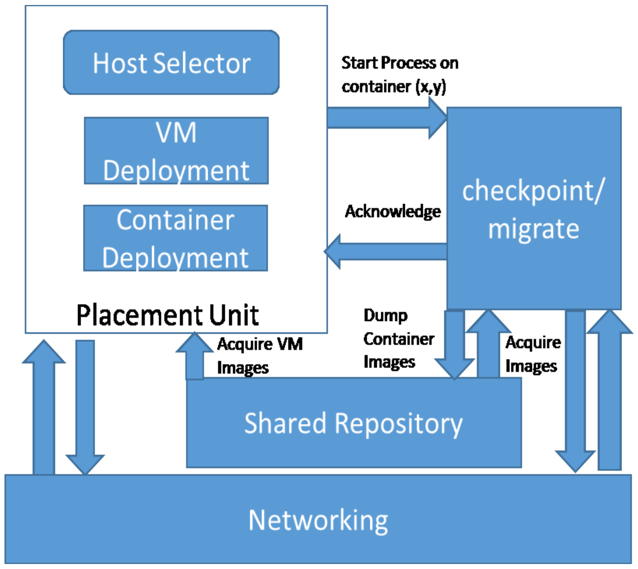


Figure 2. MIGRATE Architecture

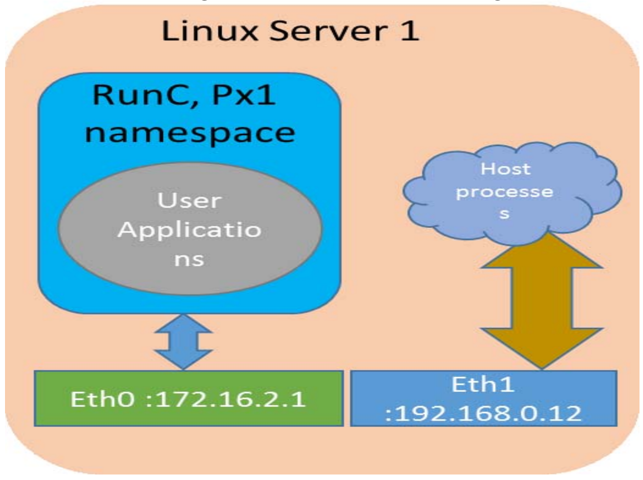
A. Application encapsulation

As mentioned before, Users will prepare their containers either manually, or using an automated script. These containers will hold the user application, and all the needed files for that application to run.
Once the container is ready with the tenant application inside, MIGRATE uses an integrated export tool to dump the customized container into a set of files that can be executed independently from the Docker management demon service. Doing so enabled us to execute the container in an unprivileged mode in the user space for easier management and better protection against privilege-escalation attempts.

B. Container networking

Linux containers are a software construct that can host an application and its dependencies as an isolated process on a Linux kernel. It allows containerized applications to share that kernel with other containers. The basic network primitive in Docker is a virtual bridge called docker0. When Docker boots up on a Linux server, it creates a default docker0 bridge inside the Linux kernel, and docker0 creates a virtual subnet on the Docker host so it can pass packets back and forth between containers on the same host. Docker also creates a pair of virtual interfaces on each container, randomly assigning them an IP address and a subnet from a private address range not already used by the host machine.

Figure 3. Container Networking



The Container intercommunication network architecture is shown in Figure 3. We prefer using a virtual network interface with a dedicated IP address for each container to facilitate runtime migration and to enhance the achieved isolation.
MIGRATE will handle the runtime mapping by local or network wide mapping of interfaces and IPs. In order to MIGRATE from Docker engine network management and enable such direct association we had to launch that container as an independent process without losing the isolation feature that Docker provides. MIGRATE uses an integrated export tool to dump the configured container into a set of image files. MIGRATE uses runC [25], a tiny tool for spawning and running containers according to the Open Container Protocol specification, to execute the container as a sub-process of runC.

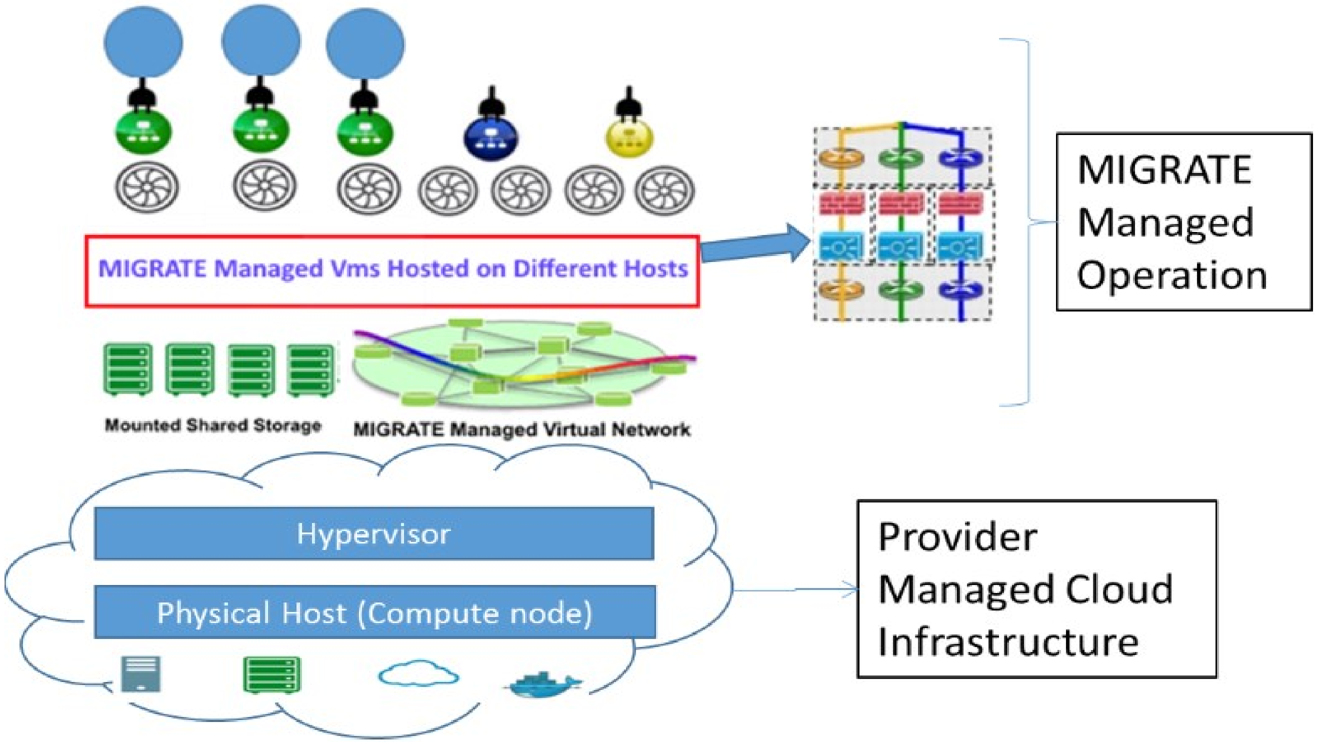

MIGRATE Managed Vms Hosted on Different Hosts
Mounted Shared Storage
MIGRATE Managed Virtual Network
MIGRATE Managed Operation
Hypervisor
Physical Host (Compute node)
Provider Managed Cloud Infrastructure

Figure 4. Container migration Process.

C. *Container checkpoint/restore and live migration*

Our primary goal is to avoid any customization or administrator or the programmer involvement, we leveraged the encapsulation state of the application and used CRIU to dump the container memory into persistent set of files easy to share and recover. CRIU is used only on state-full applications based containers, we prefer not to use it on stateless type as the memory content and the executed states are not important for container restoration. However, to offer a seamless migration process for both stateless and state-full applications, we can follow the same approach.

CRIU is a tool to checkpoint/restore running tasks in user space. CRIU momentarily freezes the running (container) runC process and all its sub-processes (user apps) and checkpoint it to a collection of image files that can be used to restore the container to the exact state later. Between these dump events, containers uses the host memory to operate to maximize the application response rate

The container image files is usually large in terms of space. In our experiments, containers with full database server can be as large as 500 MB. However, the memory dump is usually less than 10 MB. The migration process for stateless type is much easier, we replicate the container on the destination server, then make a quick network switching between the source and destination. The replication process and container instantiation time is totally negligible as the original container will still be running. The migration process is entirely logical as the network connections are the only thing that is going to migrate. MIGRATE adjust the ARP table for the NAT to point to the new server instead of the old one. For faster instantiation, quick recovery, and easy container migration, MIGRATE uses a remote shared storage as a container repository to store runC containers. Running the container from a remote storage gives instant access to multiple remote servers to instantiate such containers. Using remote repositories to host the base image of the container, massively reduced the time needed to move all the files between hosts in case of failure or live migration. The only files that have to be synchronized between the source and destination servers are the memory dump which are so small and synchronize momentarily.

MIGRATE can operate either on the cloud management layer or over it forming cloud on the cloud. In this paper, we present the later as shown in Figure 4. A set of applications hosted by MIGRATE-managed containers operating on Linux virtual servers running on a conventional cloud.

D. *Live migrations process*

For MIGRATE to work, the participating servers must run a Linux operating system customized kernel to enable some features [24] that are necessary for the checkpointing process to work. Once the server is up and running, MIGRATE mount the remote shared storage and start launching the selected container with the attached applications enclosed. As mentioned before, MIGRATE will dedicate a virtual network interface from the host server for each application. Checkpointing can occur either on a timely manner, or upon certain event triggers. On each checkpoint event, the checkpoint process starts by stalling the container dumping the files on the shared storage with a unique timestamp tag. The migration process starts by checkpointing the container, killing the process on the original host, make an ARP update to change the MAC/IP assignment of the old server network interface to match the new one while mainlining the IP value , and restore the container and all enclosed applications on the destination server. The entire process occurs in matter of milliseconds. Figure 5 lists the main algorithmic steps for the migration process.

Deployment procedure

- Instantiate Container using Docker file
- Export Container into RunC Image
- Create Virtual Network Interface for the Image
- Copy Container Image To MIGRATE managed Shared Storage
- Instantiate MIGRATE ready VM with type X on a Random Host
- Mount the Shared Storage as a Local Drive
- Connect Container Network to the Host Network
- Start Container

At each time To Start Migration procedure

- Select random destination host different than the source
- Instantiate MIGRATE ready VM with type X on a Random Host
- Mount the Shared Storage (local shared or remote shared in case of multiple data centers) as a Local Drive
- On the source VM, Call MIGRATE (Checkpoint) dump images to shared storage folder
- Connect Container Network to the Host Network
- Migration within the same network (data center) protocol
    - o make ARP change request point to the new network interface replacing the old one (same IP address)
- Migration within different networks (data center) protocol (addressing using Domain Names)
    - o make DNS change to point to the new real IP of the destination network
    - o assign a virtual IP to match the source virtual IP to bidirectional forward requests to that IP
- On the Destination VM call MIGRATE (Restore) from the dump images

Figure 5 the main algorithmic steps for the migration process.

V. SECURITY EVALUATION

The main purpose of this preliminary security evaluation is to show the effect of MIGRATE in increasing the level of attack complexity and the effort invested by attackers to allocate their targeted containers. The induced number of dynamic changes in the operational pattern of the operating containers can represent how hard it is to allocate a certain container in the network [19]. We devised a simple model that uses a set of random distributions to create the different system events.

In order to devise the mathematical representation of the migration process, we assume that the network behavior is a matrix (n*m) where each point in the matrix represents a Container (Y) as an entry in the (n, m) plan. Each (Y) entry in the (n, m) plan has a value (H) representing the id of the current host hosting this container.

MIGRATE's spatial migration is concerned with manipulating the location for each container in the matrix. We use Poisson distribution to calculate the time between two consecutive spatial shuffling events. At each event t, each container follows a uniform distribution to determine the new location that such container will migrate to.

$$H \in \{0,1,\ldots a\},\ I \in \{0,1,\ldots n\},\ J \in \{0,1,\ldots m\}$$

$$\Delta t = f_{\_p}(q),\ \Delta t \neq 0,\ q >= 0$$

$$t_{x+1} = \Delta t + t_x$$

Where f_p is the function that we use to generate the distribution controlling t. Δt is the time interval between shuffling events, MIGRATE determines the value of q controlling the shuffling frequency randomly at this stage of development. A more controlled/supervised estimation that takes into consideration the host, container, and network interest will be presented in our sequel paper.

$$i_{x+1} = f_{\_in}(z)\ ,$$

$$j_{x+1} = f_{\_jn}(z)$$

Where f_n is the function used to generate a new location (i,j) for the Container to migrate-to in the (n,m) plan, and z is a random seed set to insure that the output range of i, and j ranges from 0 to (n,m) respectively

Assuming that f_n is a normal distribution, it will be calculated as follows

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Simulator design

We used the aforementioned model to build a simulator and a set of experiments to evaluate the level of behavior change due to spatial migration. Evaluating that change reflects the level of complexity that the attacker shall face in attempting to attach the controller hosts. The level of complexity should be even much higher when we introduce the ability to switch between heterogeneously configured hosts. We did not evaluate that dimension as it was not tested yet on our testbed.

Table 1 shows the main parameters used in the simulation. The network parameters are mainly static parameters used to setup the experiments, except for the deployment of fresh Cells in the network. The dynamic part depends on a set of distributions mentioned in the column named “Generator “.

Through the experiment we are simulating the case that all the hosts have average capabilities and we assumed that a host would not refuse relocation requests. With that assumption, it is closer to a population description; which makes the normal distribution a good distribution to describe the location of the next event. While the rate of change, or inter-arrival time " the time frame between consecutive events" is best represented as a Poisson distribution.

The shuffling event parameters represent the spatial distribution of shuffling commands to induce obfuscation while the attack/failure parameters show the spatial distribution of attack events

All experiments had the same time period of 6 hours with a sample rate of 6 minutes giving us 60 samples of events of changes within the network of containers.

Table 1 Simulation parameters

Classification	**Parameter**		**Generator**	**Slow Change**	**Fast Change**
Network	Network size		Static	10*10	10*10
	Exp_Time		Static	6	6
	Speed of change factor		Static	10	15
	App_exe_time		normal	50	50
	Deploy new container	**Period**	**Poisson**	20	**18**
		Location	normal	**8,3**	**8,3**
	Spatial Shuffling event	**Period**	**Poisson**	18	18
		Location	Normal	**8,3**	**8,3**
	Attack or failure event	Timing	**Poisson**	21	20
		Location	Normal	11,3	9,4
		Type	Uniform	10	10

The presented study simulated the simple action of a moving target defense we did not separate between attacker and victim containers. We modeled it as a game where attacker succeeds only if he manages to keep the target sharing the same host with his containers for the entire time of the attack. This scenario will be tested in a large scale network in our sequel papers with a comparison between this simulated readings and actual measurements collected from our testbed.

Figure 6 shows the system automated response to the increase of number of migrations in the chance of successful attack. The system was adjusted to autonomously increase the shuffling speed, and widening the shuffling scope to mitigate increase of the chance of attack (deploying more attacker containers). Results reflected that increasing the frequency of shuffling makes it too hard for the attacker to maintain residency with his victim on the same host for the entire time of the attack. We assumed in this study that we cannot identify the attacker containers and the main goal was to minimize the chance of sharing the same host with any untrusted container.

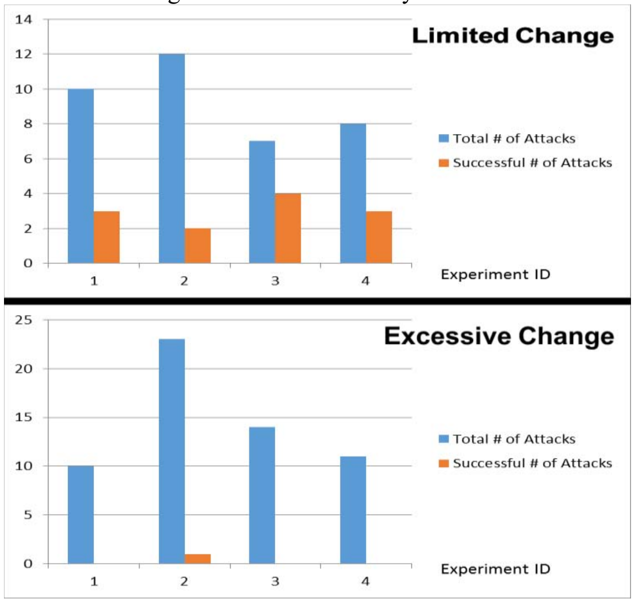


Figure 6: The effect of increasing the shuffling frequency in mitigating coresidencey based attacks.

VI. CONCLUSION

VM migration was proposed as a solution to side-channel attacks in clouds with co-residency. The induced downtimes due to frequent migrations were the main challenge facing such solution. In this paper we introduced a lightweight migration mechanism that employs Linux containers as virtualization capsules. Our approach was tested on a testbed and showed near zero downtime and extremely low resource-consumption overhead when compared to a full VM migration. A preliminary security evaluation was presented to illustrate the effect of increasing the number of migrations in mitigating side-channel attacks even with no prior knowledge of the attacker-container locations. Our future work includes smarter manipulations of the operational characteristics of the working containers, comprehensive evaluation of the system on our testbed, and enabling live migration of containers between heterogeneously-configured hosts to mitigate host-based attacks.

Acknowledgments

The authors wish to thank Eng. Ahmed Neil from Mansoura University Information and Network Department, for his help through the course of this wok. Ahmed conducted extensive literature survey reflecting the latest cross-side channel attack avoidance solutions, their benefits and limitations.

REFERENCES

[1] Twinkle Garg, Rajender Kumar, and Jagtar Singh "A way to cloud computing basic to multitenant environment" International Journal of Advanced Research in Computer and Communication Engineering, vol. 2, issue 6, June 2013, pp: 2394-2399

[2] M.Saraswathi and Dr.T.Bhuvaneswari " Multitenancy in Cloud Software as a Service Application" International Journal of Advanced Research in Computer Science and Software Engineering, vol.3, issue 11, November 2013, pp: 159-162

[3] 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, 2009.

[4] Z. Wu, Z. Xu, and H. Wang." Whispers in the hyper-space: Highspeed covert channel attacks in the cloud". In USENIX Security symposium, pp: 159–173, 2012.

[5] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart.," Cross-VM Side Channels and Their Use to Extract Private Keys", In Proceedings of the ACM conference on Computer and communications security (CCS '12),pp: 305-316, 2012.

[6] Dag Arne Osvik, Adi Shamir, and Eran Tromer. "Cache attacks and countermeasures: The case of AES". In Proceedings of The Cryptographers' Track at the RSA Conference on Topics in Cryptology pp: 1–20, 2006.

[7] Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar, "Wait a Minute! A fast, Cross-VM Attack on AES", In RAID, pp: 299–319, 2014

[8] Yuval Yarom, and Katrina Falkner," FLUSH+RELOAD: a High Resolution, Low Noise,L3 Cache Side-Channel Attack". In USENIX Security symposium, pp: 719–732, 2014.

[9] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart," Cross-Tenant Side-Channel Attacks in PaaS Clouds", In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS '14), pp: 990-1003, 2014

[10] Endre Bangerter, David Gullasch, and Stephan Krenn. "Cache games– bringing access-based cache attacks on AES to practice", In IEEE Symposium on Security &Privacy, pp: 490–505, 2011

[11] Adam Bates, Benjamin Mood, Joe Pletcher, Hannah Pruse, Masoud Valafar, and Kevin Butler, "Detecting co-residency with active traffic analysis techniques", In Proceedings of the 2012 ACM Workshop on Cloud computing security workshop (CCSW '12), pp:1-12, 2012.

[12] Venkatanathan Varadarajan, Yinqian Zhang, Thomas Ristenpart, and Michael Swift "A Placement Vulnerability Study in Multi-Tenant Public Clouds", In Proceedings of the 24th USENIX Security Symposium, August 2015, pp: 913-928

[13] Taesoo Kim, Marcus Peinado, and Gloria Mainar-Ruiz: "STEALTHMEM: system-level protection against cache-based side channel attacks in the cloud", In Proceedings of the 21st USENIX conference on Security symposium (Security'12) pp: 1-16, 2012.

[14] Stephen Crane, Andrei Homescu, Stefan Brunthaler, Per Larsen, and Michael Franz, "Thwarting Cache Side-Channel Attacks Through Dynamic Software Diversity", 22nd Annual Network and Distributed System Security Symposium, pp: 1-14, 2014

[15] Yinqian Zhang , Ari Juels , Alina Oprea , Michael K. Reiter, "HomeAlone: Co-residency Detection in the Cloud via Side-Channel Analysis", Proceedings of the 2011 IEEE Symposium on Security and Privacy, p.313-328,2011

[16] Thomas Ristenpart , Eran Tromer , Hovav Shacham , Stefan Savage," Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds", Proceedings of the 16th ACM conference on Computer and communications security, November 09-13, 2009

[17] Paul C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems", Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology, p.104-113, August 18-22, 1996

[18] Tal Garfinkel , Ben Pfaff , Jim Chow , Mendel Rosenblum , Dan Boneh, Terra, "a virtual machine-based platform for trusted computing", Proceedings of the nineteenth ACM symposium on Operating systems principles, October 19-22, 2003,

[19] M.Azab, R.Hassan and M.Eltoweissy, "ChameleonSoft: A Moving Target Defense System," 7th International Conference on Collaborative Computing, 2011.

[20] M Azab, M Eltoweissy , "ChameleonSoft: Software behavior encryption for moving target defense", Mobile Networks and Applications ,2013 18 (2), 271-292

[21] "Docker - Build, Ship, and Run Any App, Anywhere." [Online]. Available: https://www.docker.com/. [Accessed: 16-Nov-2015].

[22] "Linux Containers." [Online]. Available: https://linuxcontainers.org/. [Accessed: 16-Nov-2015].

[23] "OpenVZ Virtuozzo Containers Wiki." [Online]. Available: https://openvz.org/Main_Page. [Accessed: 16-Nov-2015].

[24] "CRIU." [Online]. Available: https://criu.org/Main_Page. [Accessed: 16-Nov-2015].

[25] "Open Container Project." [Online]. Available: https://runc.io/. [Accessed: 16-Nov-2015].

[26] Mohamed Azab and Mohamed Eltoweissy,"CyberX: A Biologically-inspired Platform for Cyber Trust Management," 8th International Conference on Collaborative Computing, USA, 2012