DSEOM: A Framework for Dynamic Security Evaluation and Optimization of MTD in Container-based Cloud

Hai Jin, *Fellow, IEEE,* Zhi Li, Deqing Zou and Bin Yuan, *Member, IEEE*

Abstract—Due to the lightweight features, the combination of container technology and microservice architecture makes container-based cloud environment more efficient and agile than VM-based cloud environment. However, it also greatly amplifies the dynamism and complexity of the cloud environment and increases the uncertainty of security issues in the system concurrently. In this case, the effectiveness of defense mechanisms with fixed strategies would fluctuate as the updates occur in cloud environment. We refer this problem as *effectiveness drift problem* of defense mechanisms, which is particularly acute in the proactive defense mechanisms, such as moving target defense (MTD). To tackle this problem, we present DSEOM, a framework that can automatically perceive updates of container-based cloud environment, rapidly evaluate the effectiveness change of MTD and dynamically optimize MTD strategies. Specifically, we establish a multi-dimensional attack graphs model to formalize various complex attack scenarios. Combining with this model, we introduce the concept of *betweenness centrality* to effectively evaluate and optimize the implementation strategies of MTD. In addition, we present a series of security and performance metrics to quantify the effectiveness of MTD strategies in DSEOM. And we conduct extensive experiments to illustrate the existence of the effectiveness drift problem and demonstrate the usability and scalability of DSEOM.

Index Terms—Container, Microservice, Moving Target Defense, Cloud Computing.

1 Introduction

In recent years, container technology has been widely adopted in cloud environments. As a lightweight alternative to the virtual machine (VM) in conventional cloud infrastructures, container has shorter startup time and lower virtualization overhead [1]. Moreover, container also provides a consistent and portable software environment, which can make cloud services ignore the difference between platforms to easily run and scale anywhere. Because of the proven benefits, container further boosts the rapid development of microservice architecture which has been widely applied in many companies, such as Uber and Netflix etc. [2], [3]. In the microservice architecture, a monolithic application is decoupled into a set of independent microservices performing a single function. It can help service providers simplify the process of updating, scheduling and scaling service [4]. At the same time, the microservice architecture has better modularity and enables application developers to take advantage of services from other providers.

Given the ever-expanding usage of container technology and microservice architecture, one of the biggest concern is whether the changes brought by them would affect the security of cloud environment. Different from the rigid VM-based cloud environment [5], microservice architecture makes service scaling and updating more flexible. And the update frequency of each service can be as high as hundreds of times a day [6]. However, the flexibility of microservices also causes the dynamism and complexity of cloud environment to increase tremendously, and meanwhile, may introduce some new security issues unintentionally. Specifically, due to the high-frequency update, defense mechanisms with fixed strategies may hardly cover the attack surface persistently in this dynamic environment. Thus, the effectiveness of defense mechanism would fluctuate (either increase or decrease) over time, which we call the *effectiveness drift problem*. Particularly, the drift problem is acute in proactive defense mechanism because its effectiveness highly depends on current situation of defense targets. If the defense strategies don't match the supposed condition, the proactive defense mechanism may not work well.

Consequently, we focus on Moving Target Defense (MTD), a representative proactive defense mechanism, as the study subject in our work. MTD can continuously transform the attack surface of target system, such as randomizing execution environment [7], software implementation [8], and network topology [9] etc., to reduce the opportunity of the attack surface being exploited by attackers [10], [11]. Currently, to maximize the effectiveness of MTD, existing studies work on evaluating and optimizing implementation strategies of MTD [12], [13], [14]. However, these methods focus on the static deployment scenarios and can't meet the requirements of dynamically optimizing MTD strategies in the container-based cloud environment. In this paper, we propose a framework called DSEOM to study and solve these problems mentioned above. As far as we know, DSEOM is the first framework for dynamically evaluating and optimizing the effectiveness of MTD techniques in the container-based cloud environment at real time.

DSEOM can automatically perceive update events in the container-based cloud environment, rapidly evaluate the effectiveness changes of MTD techniques and optimize its implementation strategies. To rapidly evaluate and optimize MTD strategies, we establish a holistic attack graph (HAG)

• *H. Jin, Z. Li, D. Zou and B. Yuan are with the National Engineering Research Center for Big Data Technology and System, Cluster and Grid Computing Lab, Services Computing Technology and System Lab, Big Data Security Engineering Research Center, Huazhong University of Science and Technology, Wuhan, 430074, China. Professor Zou is the corresponding author. E-mail: deqingzou@hust.edu.cn*

to model all feasible attack events in the container-based cloud environment. And based on the attackers' behavior pattern, we introduce the concept of *betweenness centrality* used to locate weak points in HAG to help assess and optimize the effectiveness of MTD strategies. However, due to the complexity of the cloud environment and diversity of attack behaviors, the betweenness centrality in HAG can poorly depict the weak points in some special attack scenarios. To accommodate this problem, we propose a multi-dimensional attack graphs model to cover a set of specific attack scenarios. Furthermore, we combine the betweenness centrality in these two kinds of attack graph models to locate the weak points in cloud environment. From this, DSEOM can rapidly ascertain the effectiveness drift problem of MTD and accurately determine the weak points to generate optimal MTD strategies.

In addition, DSEOM is a flexible and extensible framework that provides a series of MTD techniques as the study examples which can be dynamically added or removed. In order to effectively demonstrate the availability of DSEOM, we define a set of security and performance metrics to quantify and compare the effectiveness difference between the MTD strategies provided by DSEOM and the initial strategies. Based on the experiment results, we show that DSEOM can rapidly optimize and maintain the effectiveness of MTD strategies with acceptable costs in the container-based cloud environment.

In summary, we propose a framework that can automatically and effectively develop MTD strategies adapting to the high dynamic container-based cloud environment. More specifically, our contributions are:

- We present a framework for automatically generating, deploying and adjusting MTD strategies to enhance the security of the high dynamic cloud environment.
- We propose a multi-dimensional attack graphs model to comprehensively cover and analyze complexity attack scenarios in container-based cloud environment.
- We combine betweenness centrality with multi-dimensional attack graphs to accurately evaluate and optimize the effectiveness of MTD strategies.

The remainder of this paper is structured as follows. In Section 2, we discuss container-based cloud environment and MTD techniques in more details. Section 3 describes the threat model. We demonstrate our framework design and detailed elaboration on its theoretical basis in Section 4. Section 5 shows the details of DSEOM. Section 6 presents evaluation results of our framework. We discuss the limitations and extensions of the framework in Section 7 and related work in Section 8. We conclude in Section 9.

2 Background

2.1 Container-based Cloud Environment

Based on lightweight (OS-level) virtualization, container needs less system resources to run a service and has faster startup times, superior performance of I/O throughput and lower latency compared with VM [15]. At the same time, by packaging application code and dependencies together, container management platforms (such as Docker [16] and Kubernetes [17]) can provide a consistent environment for development, testing, and production of applications, which can shorten the release and update cycles greatly. Due to these characteristics, container can run in any cloud environments regardless of the differences between environments.

The aforementioned characteristics promote development of a new service architecture, microservice architecture [18], where the application is decoupled into a set of simple functional services called microservice. Each microservice serves as a single purpose, and is deployed in the container. In the cloud environment, a set of microservices interact with each other using HTTP or RPC to form a service chain and provide an integrated function to users. Incorporating features of the container mentioned above, the applying of microservice enables the cloud services to flexibly scale, deploy and schedule. However, it also extremely increases the dynamism of cloud environments since the microservice may be independently updated 10-100 times a day [6].

Different from traditional cloud environments, container technology and microservice architecture make up the container-based cloud environment. However, excepting for the advantages brought to container-based cloud environment, some security issues introduced by these new technologies are also being widely concerned. The main security issues include the weak isolation problem among containers [19] which also exists in VM-based cloud environment [20], greater attack surface caused by microservice [21], [22], and the security drift problem caused by its high dynamic [23].

2.2 MTD Techniques

Moving target defense (MTD) is a proactive defense mechanism which continuously alters the attack surface of IT system to increase uncertainty and complexity of the attack [24]. The higher attack costs and lower successful opportunity introduced by MTD effectively increase the difficulty for penetrating the system and maintaining the persistent presence to attackers. Cloud environment is one of the practice scenarios for MTD techniques. Based on the deployment location in cloud environment, MTD techniques can be categorized into three different domains: application level, platform level and network level. (1) Application-level, MTD technique provides applications with equivalent functions but different implementations [25], [26] (e.g. language, data structure, algorithmic logic etc.); (2) Platform-level, the running environment of the application is randomized [27], [28], such as operation system (OS) diversity and container/VM live migration; (3) Network-level, the network properties are continuously rearranged [29], [30], [31] (e.g., ip shuffling, topology reconstruct etc.).

Though MTD techniques can improve the security of the cloud environment, the direct or indirect influence caused by them to the service performance cannot be ignored. For example, the network-level MTD techniques will directly cause the quality of web services declining and influence the communication between services [32]. Similarly, other MTD techniques, such as container/VM live migration, will consume certain system resources (CPU, memory or disk), which may cause the performance degradation of cloud services indirectly. Consequently, determining appropriate MTD implementation strategies to balance the performance overhead and its effectiveness is critical [33].

3 Threat Model

The cloud environment in which exist multiple assets is one of the most common target for attackers. We assume that the cloud platform and the service provider are trusted, and meanwhile attackers come from external network. The goal of attackers is penetrating into cloud environment to obtain important data. We first display an example of attack scenario on container-based cloud environment in Fig. 1. Then, we formally depict the attack targets as follows:

Attack targets: In the container-based cloud environment, service instances ST run on worker nodes and contain the *service application* and *container* to be the attack targets which are formalized as $ST = \{S, C\}$. The service application S consists of service code and its dependency libraries, and the container C is the virtual environment to run these codes. As shown in Fig. 1, S and C are locate on *service layer* and *virtualization layer* in the cloud environment respectively. Moreover, we assume that these targets have some vulnerabilities that can be exploited by attackers.

Attack Process: To compromise the target, we assume that attackers wage attacks according to the *cyber kill chain* [34]. Specifically, attackers first perform various reconnaissance actions to identify vulnerabilities in the target. Further, they select an appropriate vulnerability and prepare exploitation tools against it. Next, attackers can use these tools to execute malicious code for compromising the target.

Attackers' Capability: In general, we assume that powerful attackers have the ability to attack multiple targets simultaneously and continue their attack process on the basis of compromised targets until achieving the final goal. Specifically, we scope attackers' capabilities based on the layers on which these targets locate in Fig. 1.

- *External Network:* We assume that attackers are able to perform the attack on service application S of public service instances. Based on the compromised target S, attackers can access the service instance located on *service layer* and continue their attack process.
- *Service Layer:* Through the internal network, attackers can conduct the attack process on service application of the service instances which exist dependency relationship with the compromised target S. This constraint comes from the network separation policies established by the cloud management tools running on manager nodes.

  Moreover, attackers can also implement attack to the container C on which the compromised targets S run. By exploiting the poorly configurations or vulnerabilities (e.g. CVE-2016-5195) associated with the container, attackers can escape from virtual environment to the worker node. After that, attackers have full control of the worker node below the containers and access *virtualization layer*.
- *Virtualization Layer:* With limitation of network separation, attackers can perform the attack process on reachable service applications. Also, they can directly get access to the service instances located on the same node.

Moreover, the attackers' capabilities also have constraints. We assume that attackers don't know the details of cloud services configurations, including the orchestration of services, placement of services in the cloud environment and the number of replicas a service has. As well, they don't know the details of security configurations for each container.

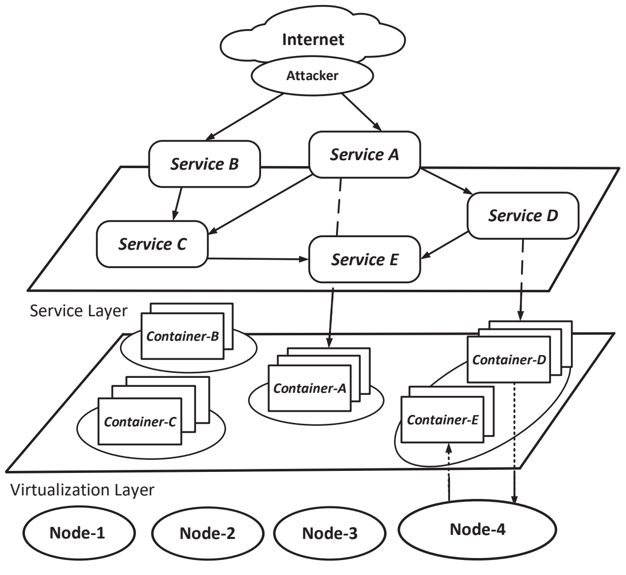


Fig. 1: An example of container-based cloud environment.

4 Framework Overview

In this section, we describe the theoretical model for designing the framework DSEOM to rapidly evaluate and optimize the effectiveness of MTD, and show the design details of the framework. At last, we analyze the feasibility of MTD strategies optimized by DSEOM on defending against attackers.

4.1 Theoretical Modeling

According to the threat model, we formally define the HAG covering overall attack events in container-based cloud environment, and the multi-dimensional attack graphs to formalize specific attack scenarios. Then, we introduce the concept of betweenness centrality to determine the weak points in these attack graphs and assess variation in effectiveness of MTD strategies.

Definition 1. Holistic Attack Graph (HAG) is a directed graph $G = (N, E)$, where N is a set of nodes mapping the attack targets $f(N) \rightarrow \{S, C\}$ in the cloud environment, and $E \subseteq N \times N$ is a set of edges where each edge is an attack path between the ordered pair of nodes (N_i, N_j) that $N_i, N_j \in N$ and $i \neq j$.

An attack path signifies an attack action that allows attackers to compromise the target node N_{post} from a compromised node N_{pre}. To each attack path, its attack difficulty is associated with the exploitability of vulnerabilities in the target node N_{post}. In the following, we formally define the attack difficulty as the weight of edge in the attack graphs.

Definition 2. Given an edge $E_i = N_{pre} \rightarrow N_{post}$, where $E_i \in E$ and $N_{pre}, N_{post} \in N, pre \neq post$, we define $DF(E_i)$ as the weight of edge E_i to denote the attack difficulty for compromising node N_{post}.

In order to evaluate the attack difficulty, we first use metrics defined in Common Vulnerability Scoring System (CVSS) to quantify the exploiting difficulty with vulnerabilities in the target node [35]. The metrics defined in CVSS v3.0 [36] consist of three parts (base, temporal, and environmental metrics). Among them, the base metrics are used to assess the exploitability of the vulnerability, including the attack vector (AV), attack complexity (AC), privilege required (PR) and user interaction (UI). Based on these base metrics, we estimate the exploiting difficulty ED of each vulnerability in the node by the formula shown in Eq. (1).

$$ED = (AV \times AC \times PR \times UI)^{-1} \tag{1}$$

For compromising a node, the attacker only has to successfully exploit a vulnerability contained in this node. Due to the unknown of attackers capabilities and preferences, each vulnerability in the node is likely to be attacked. We use the exploit code maturity (ECM), which is part of temporal metrics, to evaluate the likelihood of each vulnerability been attacked. Further, we take this likelihood as the weight to calculate the weighted average of exploiting difficulty ED on all vulnerabilities, which is the attack difficulty $DF(E_i)$ of the node. And the formula is shown in Eq. (2).

$$DF(E_i) = \frac{\sum(ED \times ECM)}{\sum ECM} \tag{2}$$

Following attackers' behavior patterns mentioned in Section 3, we introduce the *betweenness centrality* to identify the significance of nodes in the attack graph. In a traditional network, betweenness centrality is used to detect the amount of influence a node has over the flow of information. The node with higher betweenness centrality plays a more important role to bridge the network. Similarly, the higher betweenness centrality of a node in attack graphs means compromising it is more helpful to attackers penetrate the system. Specifically, the betweenness centrality can be calculated by Eq. (3), where σ_{st} is the total number of shortest paths between any two nodes s and t, and $\sigma_{st}(v)$ is the amount of paths passing through node v in these shortest paths. The shortest path between two nodes in the attack graph means that the sum of the attack difficulty of its constituent edges is minimized.

$$B_n(v) = \sum_{s \neq t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}} \quad s, t \in N \tag{3}$$

Nevertheless, considering the complexity and diversity of attackers' behaviors in real world, the betweenness centrality calculated on HAG may not cover the important nodes in some specific attack scenarios. For example, the attack scenarios happen on virtualization or hardware level (e.g. container escaping). In this case, the attack targets generally have a great attack difficulty, and its subsequent targets are rarely contained in shortest paths in HAG. Therefore, the betweenness centrality can't effectively reflect the important nodes related to these attack scenarios.

To solve the problem, we define the *multi-dimensional attack graphs* in Definition 3 to depict these specific attack scenarios. The multi-dimensional attack graphs consist of a set of independent attack graphs divided from HAG. Each independent attack graph describes the actions of attackers located on the different layers of cloud environment respectively. By integrating the betweenness centrality in the HAG and the multi-dimensional attack graphs (AGs), we can rank the importance of nodes more accurately.

Definition 3. Multi-dimensional attack graphs $G^* = (N^*, E^*, L)$ is a set of subgraphs of HAG $G = (N, E)$, where the L is the layer of cloud environment, $E^* \subseteq N^* \times N^*$ and $(N_i^*, N_i^*) \notin E^*$. When L=service, $N^* = \{N^{'} \in N \mid f(N^{'}) = S\}$; When L=virtualization, $N^* = \{N^{'} \in N \mid f(N^{'}) = C\} \bigcup \{N^{''} \in N \mid \text{from } N^{'} \text{ to } N^{''} \text{ is reachable }\}$.

The important nodes in attack graphs are not only exploitable weak points for attackers but also the location on which the MTD deployed. However, considering performance impact, we cannot deploy MTD on all nodes. Therefore, a *decision algorithm* is needed to determine on which nodes the MTD should be deployed for maximizing the system security with minimum impact on performance. Due to the difference between nodes, the properties of each node are key factors to influence the decision. Therefore, we concretely define a set of properties to depict the importance of a node and the costs of MTD implementation on it.

Definition 4. A node's properties is a three-tuple $P = (Cost, Dec, Imp)$, where $Cost$ quantifies the time costs of MTD implementation on the node, and the resulting decline in quality of service is quantified as Dec. And Imp denotes the betweenness centrality of the node.

In the real cloud environment, the time costs of MTD implementation and the service performance declining caused by it are related to many factors, such as types or configurations of services. Therefore, $Cost$ and Dec are measured in the real cloud environment.

After MTD deploying on selected nodes, the attack process on them has a great probability be interrupted due to the changing attack surface. It makes attackers need to pay a steeper price compromising these nodes. In this case, we design Eq. (4) to formalize the attack difficulty of the nodes where MTD is deployed. The $\Delta T / Cost$ describes the times of MTD implementation per unit time ΔT, where the $Cost$ is time costs of MTD implementation mentioned in the Definition 4. We suppose that attackers have to restart the attack process after each time of MTD implementation. So in unit time, the attack difficulty for attackers is $\Delta T / Cost$ times than the original.

$$DF(E_i^*) = \frac{\Delta T \times DF(E_i)}{Cost} \tag{4}$$

At the same time, the shortest path between some nodes also changes in AGs accordingly. It means the minimal attack difficulty across these pairs of nodes also has altered. We quantify the change ΔDF in Definition 5 to evaluate the effectiveness of MTD strategies.

Definition 5. Given a started node N_s and a target node N_t in AGs, the shortest path between these nodes is $sp : N_s \to N_t$ with attack difficulty $AF(sp)$. After implementing the MTD, the total attack difficulty on this path is $AF^*(sp)$. The $\Delta DF(s,t) = AF^*(sp) - AF(sp)$ is the security gain brought by MTD implementation.

Expanding to the whole cloud environment, we denote the effectiveness of MTD strategies as EM. It is computed by Eq. (5) that sums the attack difficulty increment on the shortest paths between any two nodes in AGs. In a global perspective, the trend of EM can also reflect the effectiveness drift of MTD strategies after update events occurred. This is because that the updates on services can be transformed into the changing of attack difficulty on related nodes, which directly lead changes of partial shortest paths in attack graphs.

$$EM = \sum_{s \neq t}^{N} \Delta DF(s,t), \quad s,t \in N \tag{5}$$

4.2 Framework Design

In this section, we provide a design overview of the DSEOM framework which can monitor update events in cloud environment, and provide an evaluation and optimization mechanism based on the theoretical model to keep the effectiveness of MTD strategies optimal in real time. Fig. 2 shows the overall framework architecture of DSEOM that consists of three modules, *Monitor Engine*, *Decision Engine* and *Defense Handler*.

The *Monitor Engine* module is in charge of collecting update events and monitoring service performance in real time. In the cloud environment, the update events cover various aspects containing service scaling, upgrading, scheduling, and migrating etc. For simplicity, we only focus the updates on service version, instance scale and the dependency relationship between services, which can map to the changes on attack graphs. While the updates occurred, the *Monitor Engine* collects them and notifies *StatsCollector*.

Moreover, *Monitor Engine* is also used to monitor the performance change of services caused by MTD implementation. Before framework running online, the performance declining caused by MTD implementation on given services will be measured as initial parameters to help generate AGs. However, with the services updating, the accuracy of these parameters can't be guaranteed anymore. Hence, *Monitor Engine* will monitor the performance change of services caused by MTD in real time, and update these parameters recorded in *StatsCollector*.

Further, based on the update events, *StatsCollector* will reconstruct the attack graphs (HAG and multi-dimensional attack graphs) generated by it when the framework initialize. On the basis of AGs, *MTD Optimizer* can evaluate the effectiveness drift of MTD strategies with Eq. (5). Once the effectiveness of MTD strategies is reduced, *MTD Optimizer* will re-select appropriate nodes to deploy MTD. Specifically, to maximize the security, MTD should deploy on the nodes with high betweenness centrality as many as possible, meanwhile, keep total performance costs acceptable. Thus, we formalize this decision process as a *Knapsack problem* to rapidly determine a set of nodes on which deploying MTD can maximize security and ensure performance costs under the preset limit.

More details on this decision process are shown in Algorithm 1. Firstly, the AGs and the update events are enumerated as the inputs of the algorithm. Then, we need to recompute the shortest path between all nodes in the AGs

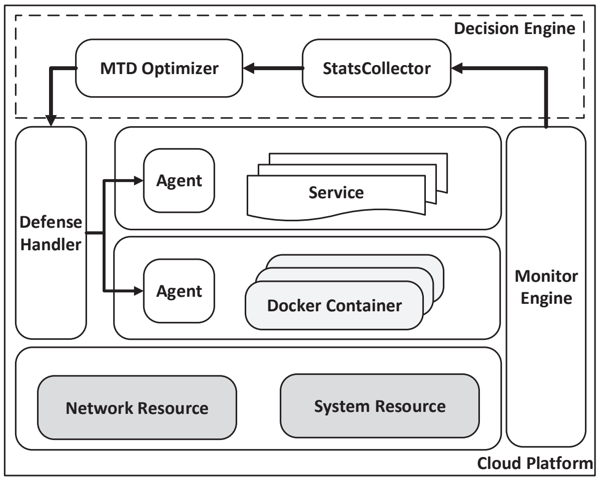


Fig. 2: DSEOM Framwork Overview.

based on the update events. However, the time-consuming of recomputing process greatly affect the scalability and real-time of the decision algorithm. Unfortunately, the *Dijkstra* algorithm and the locally shortest paths (LSP) algorithm [37] are not effective in our scenario. To accelerate this process, we introduce the fully dynamic algorithm (D-KIN) [38] and modify it for maintaining all-pairs shortest paths after updates. Based on the new all-pairs shortest paths, we use the Eq. (3) to calculate the betweenness centrality of each nodes in AGs. Next, the betweenness centrality of each node and the QoS declining for deploying MTD on each node will as parameters in 0-1 knapsack algorithm to decide the optimal nodes for deploying MTD.

The *MTD Decision Algorithm* is the core of *Decision Engine* which consists of *StatsCollector* and *MTD Optimizer*. Next, the selection results will be sent to *Defense Handler* which is MTD deploying module on the cloud platform. Based on the results, *Defense Handler* will deploy MTD on specified targets through *Agent* located on each worker node in cloud platform. At the same time, *Defense Handler* notifies *Agent* to repeal the MTD implementation on the old targets.

4.3 Security Analysis

In this section, we analyze the rationality of the MTD strategies provided by DSEOM. For this goal, we capture and analyze the behavior pattern of attackers to determine the targets with high probability to be attacked. According to these analysis, we show the validity of betweenness centrality which is used to formalize the importance nodes in AGs.

In order to penetrate a large system, attackers need to carry out multi-stage attacks to achieve their final targets. Powerful attackers or an attacker team can simultaneously invade multiple targets and continually penetrate into the system based on the compromised targets. Relatively, junior attackers limited by the capability need to explore and compromise the targets with low attack difficulty step by step till achieving final targets. On the basis of these behavior patterns, powerful attackers can take shorter periods of time to achieve their goals along the path with the lowest

Algorithm 1 MTD Decision Algorithm

```
Input: Original attack graphs $AGs$=$(G_h, G_s, G_v)$ and all-pairs shortest paths set $p$=$(p_h^*, p_s^*, p_v^*)$ in $AGs$; $u$=$(N^*, E^*)$ is the changes set of nodes and edges in $G_h$ after update; QoS declining caused by implementing MTD on each node $w$=$(w_1, ..., w_n)$; The acceptable QoS declining $W$
Output: The optimal objects of MTD deployment;
1:  function MTDOptimizing($AGs, p, u, w, W$)
2:      $AGs'$, $p'$ = D-KIN($AGs$,$p$,$u$)
3:      $n$ = NodesNum($AGs'.G_h$)
4:      for $i = 1$ to $n$ do
5:          $bc[i]$ = BetweennessCentrality($p'$,$AGs'$,$i$)
6:      end for
7:      $N^*$ = 0-1-knapsack($bc, w, W, n$)
8:      $AGs = AGs'$, $p = p'$
9:      return $N^*$
10: end function
11:
12: function 0-1-knapsack($bc, w, W, n$)
13:     for $wg = 0$ to $W$ do
14:         $c[0, wg]$ = 0
15:     end for
16:     for $i = 1$ to $n$ do
17:         $c[i, 0]$ = 0
18:         for $wg$ = 1 to $W$ do
19:             if $w_i \leq wg$ then
20:                 if $bc[i] + c[i-1, wg - w_i] > c[i-1, wg]$ then
21:                     $c[i, wg]$ = $bc[i]$ + $c[i-1, wg - w_i]$
22:                 else
23:                     $c[i, wg]$ =$bc[i]$ + $c[i-1, wg]$
24:                 end if
25:             else
26:                 $c[i, wg]$ = $bc[i]$ + $c[i-1, wg]$
27:             end if
28:         end for
29:     end for
30:     return $c[n, W]$
31: end function
```

attack difficulty. Although junior attackers may not follow the same shortest path, their attack paths between partial targets will have the least attack difficulty.

In addition, the attack difficulty of the vulnerabilities or services can be objectively estimated by CVSS and is formalized as the weight of edges in AGs. Therefore, the shortest paths between the entry and target of attackers in AGs have great probability be the attack paths for attackers. For maximizing attack difficulty for attackers, deploying MTD on the nodes located on these shortest paths will work significantly. Moreover, due to the uncertainty of attackers' goal and complexity of the cloud environment, each node in AGs can be entry or target of attackers. In this case, the more a node locates on different shortest paths in AGs, the higher probability it will be compromised in the attack. According to the definition of betweenness centrality and the analysis above, the betweenness centrality of each node in the weighted directed AGs can directly reflect the probability of nodes being attacked. Therefore, deploying MTD on the nodes with high betweenness centrality is validity for interrupting the attack process.

However, in some attack scenarios, attackers would break through the virtual environment for greater benefits. And we observe that this kind of attack actions have a high attack difficulty for attackers. Thus, in HAG, its subsequent nodes are rarely contained in the shortest attack paths. It means that the betweenness centrality based on HAG can't cover the important targets in these attack scenarios. To cope with unpredictable behaviors of attackers and cover all attack scenarios, DSEOM provides the multi-dimensional attack graphs that respectively models the attack scenarios of each layer in the container-based cloud environment to revise the accuracy of betweenness centrality. When attackers start attack in virtualization layer, the important targets can be independently ascertained by the multi-dimensional attack graphs. Based on above analysis, the betweenness centrality synthetically computed by multi-dimensional attack graphs and HAG can locate important targets in all attack scenarios to help us effectively develop MTD strategies.

5 DSEOM Implementation

In this section, we describe implementation details of DSEOM following the structure in Fig. 2. In addition, we introduce the implementation of two kinds of MTD techniques incorporated in our framework, and the optimization to decision algorithm.

The container-based cloud environment is the framework's backend. We use *Docker* in *swarm mode* [39] to build and manage the container-based cloud environment. *Docker swarm mode* is a popular open-source cluster management integrated with *Docker Engine* and provides the functions containing scaling, service discovery, load balancing and rolling update to help users easily deploy and maintain the services distributed in multiple nodes. And we implement the MTD techniques as an independent functional module in *Docker swarm mode*. When *Defense Handler* is notified to deploy the MTD strategies, it sends deploying strategies to the manager node of *Docker swarm mode*, which will activate the corresponding MTD function in the cloud environment.

MTD techniques: We implement two kinds of MTD techniques, *ip shuffling* and *live migration*, as the examples in our framework. IP shuffling technique, working on the network level, is used to shuffle the ip addresses of a set of containers. We build an ip address pool in *docker swarm* to store available ip addresses for containers. For every fixed period of time, the ip addresses of a set of containers running the selected service will be updated with new ip addresses. Live migration technique works on the platform level. We use *Checkpoint/Restore In Userspace (CRIU)* [40] integrated with *Docker* to achieve the container live migration. First, when a set of containers need to be migrated, the *Docker engine* create new checkpoints for these containers and notify the target nodes to receive the checkpoint files. And then, target nodes will restore running containers from these checkpoint files. After finishing this process, *Docker swarm* will destroy the old containers. These

MTD techniques will be implemented after every preset fixed interval.

Monitor Engine: In the *Docker swarm mode*, the manager node maintains and logs the state of the entire cloud environment, which contains events of service updating, scaling, and the changes of service configuration etc. Moreover, service providers generally use third-party services, such as Prometheus [41], to monitor performance or running status of target services in real time. These monitor data will be aggregated in the manager node and be used to maintain high availability and load-balance of service. Thus, we embed *Monitor Engine* in the manager node and grab the newest logs of the swarm and performance status through the corresponding API. In this way, the framework can collect the update events and performance status in real time with negligible overhead.

Decision Engine: In our study, we only focus on vulnerabilities that can bypass firewall and execute code via remote access. At framework initialization, *Decision Engine* creates a database with the information (service version, CVE-ID, vulnerability description, and CVSS score) about this kind of vulnerabilities in given services from NVD [42], and generate the attack graphs based on it. Moreover, the *Decision Engine* also periodically update the database with new homogeneous vulnerabilities reported in NVD. The *Decision Engine* is embedded as a standalone module in the manager node of swarm. Combined with update events, it can access this database to revise the attack graphs.

6 Evaluations

In our experiments, we use the *Docker Swarm mode* to build cloud environment on our testbed containing 7 nodes (1 controller node and 6 computing node) equipped with 2.40GHz 64-bit Intel Xeon CPU E5-2630 v3 processor with 32-cores, 64 GB RAM, 4T disks, and two network interfaces with 1Gbps network speed.

We deploy a set of services on our cloud environment to make up a complete web application. The construction of these services is same as the example shown in Fig. 1. And each service has at least one vulnerability to be exploited by attackers. More details of these services and its vulnerabilities are depicted in Table 1. Moreover, these services are distributed in different nodes and run with various scale. Through the internal network, only interdependent services can communicate with each other based on the expected rules configured by security group [12], [43] and this communication is stateless in container-based cloud environment [44]. In addition, the update events in our experiments include service scaling (instances increase or decrease) and service upgrading (version update). In more detail, we use the common *canary deployment* to upgrade services and the built-in functions of docker swarm mode to implement scaling.

6.1 Evaluation Metrics

According to the threat model mentioned in Section 3, the attack process to a service can be divided into two phases, including reconnaissance and exploiting phase. During the reconnaissance phase, attackers gather vulnerabilities information in the target service. And then, these vulnerabilities are utilized to compromise the service in exploiting phase. We define some metrics to quantitatively evaluate the capability of attackers in different phases, meanwhile, it can also reflect the effectiveness of MTD strategies indirectly.

TABLE 1: CVE vulnerabilities of services.

Label	Target	CVE ID	Exploitability Score	DF
A	Apache	CVE-2018-11776	2.2	2.74
		CVE-2017-5638	3.9	
		CVE-2017-12611	3.9	
		CVE-2016-4461	2.8	
		CVE-2016-3090	2.8	
B	Tomcat	CVE-2017-12617	2.2	2.93
		CVE-2017-12615	2.2	
		CVE-2016-8735	3.9	
		CVE-2016-0714	2.8	
C	Memcached	CVE-2016-8705	3.9	2.64
		CVE-2016-8704	3.9	
		CVE-2016-8706	2.2	
D	ImageMagick	CVE-2016-5841	3.9	2.11
		CVE-2016-6520	3.9	
E	MySql	CVE-2016-6662	2.8	2.90
-	Container	CVE-2016-5195	1.8	3.2

Definition 1. We define temporal metrics T_R as the reconnaissance time that attackers get vulnerabilities information and T_E as the time needed by successfully exploiting a vulnerability to penetrate a service.

The attacker roughly gets the basic information of target service at time t_0. And at time t_1, the attacker use precise sniffing to obtain useful vulnerabilities in the service. We use $T_R = t_1 - t_0$ to evaluate the capability of the attacker during reconnaissance phase. When the attacker has collected all the information they need at time t_2, the attacker begin to exploit the vulnerabilities in the service until successful invading the service at time t_3. We use $T_E = t_3 - t_2$ evaluate the capability of the attacker during exploiting phase.

Further, we define some other metrics from different aspects to directly assess the effectiveness of MTD strategies.

Definition 2. The metric N_D is the number of interruptions in the attack process caused by defense mechanism.

Definition 3. The temporal metric T_D is the time needed by implementing the defense mechanism.

Influenced by various factors, the implementation time costs T_D is an important parameter to impact the effectiveness of MTD techniques. To MTD techniques, T_D can be measured with the difference between the time t_4 when the attack surface begins to change and the time t_5 when the attack surface completes changes. On the other hand, the interruption times N_D can intuitively show the contribution of the MTD techniques. Synthesizing them, we can evaluate the effectiveness of MTD techniques directly.

Furthermore, the implementation of MTD techniques would consume a part of resources that used to maintain the quality of service. We use the following metrics to evaluate the performance damage introduced by MTD: (1) *Throughput* measured the number of processed requests per second; (2) *Latency* measured the job completion time for a user. The trends of these metrics can help us evaluate the performance costs brought by defense mechanism.

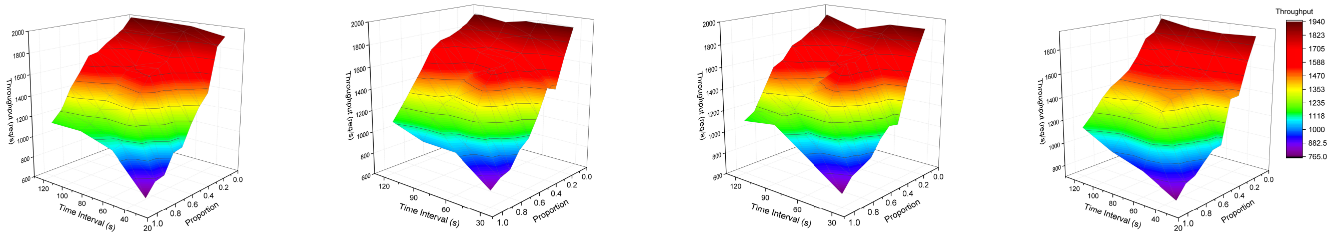

(a) 15 containers (b) 30 containers (c) 45 containers (d) 60 containers

Fig. 3: The QoS trend of web service when implementing IP Shuffling under different scale and parameters.

TABLE 2: Average performance overhead of implementing IP shuffling with acceptable proportion and frequency.

Frequency	10%		20%		30%		40%		50%	
	Avg. (req/s)	stdenv.	Avg. (req/s)	stdenv.	Avg. (req/s)	stdenv.	Avg. (req/s)	stdenv.	Avg. (req/s)	stdenv.
30s	1838.34	19.14	1712.88	25.49	1639.32	18.45	1549.38	20.33	1456.86	25.88
60s	1931.88	41.05	1864.18	50.50	1780.63	41.00	1643.43	73.10	1599.42	76.06
90s	1901.89	59.61	1861.92	12.75	1720.50	32.07	1618.84	33.47	1532.23	36.97
120s	1951.08	27.24	1852.42	25.55	1805.41	19.78	1729.98	54.62	1695.92	40.84

6.2 Preliminary Evaluation

In this section, we discuss the time costs for two kinds of MTD techniques implementing on different service scale. Accordingly, we assess the performance degradation of different kind of services caused by MTD techniques. Due to the requirement of QoS, the number of containers running a service is flexible and dynamic. In actual conditions, we conducted our experiments with the number of containers shown in Table 3.

TABLE 3: Experiment Scales.

Type	Container			
Numbers	15	30	45	60

6.2.1 IP Shuffling

We implemented ip shuffling with frequency ranging from 30 seconds to 120 seconds on the web service. To minimize the impact to normal packets, the setting of frequency is based on the Maximum Segment Lifetime (MSL) in TCP packet, which defines the time a TCP segment existing in network and commonly be set as 30s, 60s or 120s. Further, we simulated 10,000 requests originating from 500 connections with Apache Benchmark tool [45] to measure the throughput of this web service.

We observe that the time costs for implementing ip shuffling is the millisecond-level and can be negligible compared to the attack time. But the performance degradation of web service caused by ip shuffling is evident. Specifically, the throughput of web service decline by an average of 89% when the ip of all service instances are shuffled simultaneously. Fig. 3 shows the throughput trend of the web service with different scale under ip shuffling. In these figures, *x-axis* denotes the implementation frequency of ip shuffling and *y-axis* denotes the proportion of instances to be shuffled in each time. In more detail, Table 2 displays the average and standard deviation of throughput in different conditions of ip shuffling.

According to these results, the service scale and the shuffling frequency have an inconsequential influence in the throughput of web service. This indicates the shuffling proportion is the key factor to influence service performance. When the proportion range from 10% to 50%, the performance degradation of web service can keep within 20% which can be accepted. However, the lower shuffling proportion means higher implementation time costs to shuffle all the instances. Therefore, a balance point is needed between performance degradation and time costs. Based on the observation above, we set frequency of ip shuffling as 30s and 50% instances will be shuffled at each time. In this case, we can finish a complete shuffling process within minimum time costs, meanwhile, the performance degradation is acceptable.

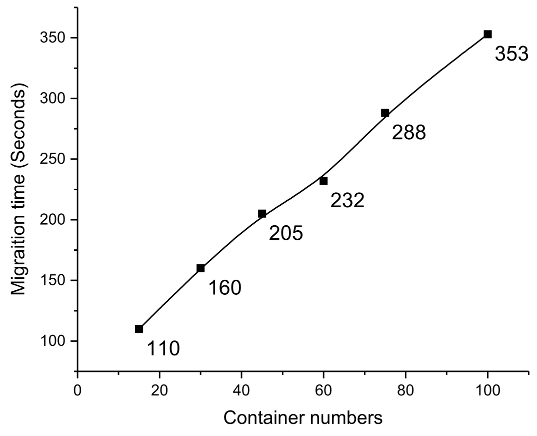


Fig. 4: The time for live migration in different service scale.

6.2.2 Live migration

Different from ip shuffling, live migration wouldn't impact throughput of the web service because of the service access isn't interrupted during migration. However, it introduces a certain delay to workloads handling in the service, which is proportional to the time costs of container migration. In our study, the time costs of migration is mainly decided by the size of checkpoint files which are approximate with the memory usage of a container. When the available memory to the container is limited with 1G, the migration time for different scale of container is depicted in Fig. 4. We observe that the implementation time costs of live migration is nearly proportional to the number of containers. In more detail, the average latency of workload completion introduced by live migration in the *ImageMagick* and the *Memcached* is close to the migration time (7.69s) of a single container.

6.3 Usability Evaluation

To rank the importance of services, we compute the betweenness centrality of each service as shown in Table 4 according to the attack graphs. Next, we launched a real attack with reconnaissance and exploiting phases for penetrating *Mysql* when the MTD technique deployed on different services. During the attack process, we measured the metrics mentioned in Section 6.1 to evaluate the effectiveness of MTD. Fig. 5 shows the total time needed by different attack phases in a successful attack process. It shows that after MTD deploying, the reconnaissance time has significantly increased, and the exploiting time depicted by the shadow area is also increased with a certain. Also, we observe that MTD technique deployed on the services with higher integrated betweenness centrality has a greater effect. By contrast, the service with the highest betweenness centrality in HAG is not the best choice for MTD deployment. In addition, the broken line graph in Fig. 5 describes the time for compromising each single service without MTD technique. It shows that the attack difficulty of the services is proportional to the time needed to compromise them.

TABLE 4: Betweenness centrality of services

Service	HAG	Service-AG	Virtual-AG	Synthesis
Apache	8	3	4	15
Tomcat	1	0	0	1
Memcached	1	1	3	5
ImageMagick	9	2	3	14
MySql	0	0	0	0

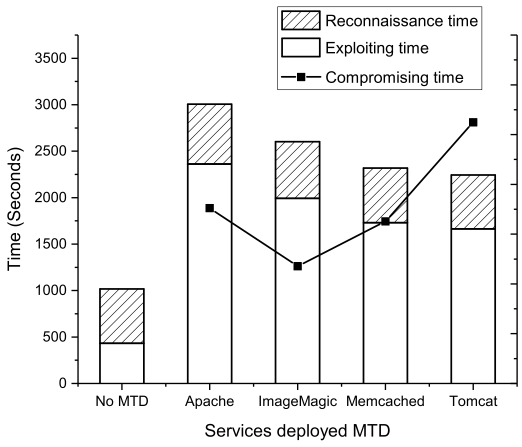


Fig. 5: The comparison of attack time while MTD deployed on different services.

After MTD was deployed on *Apache*, we measured the number of interruptions N_D in the attack process to quantify and compare trends in the effectiveness of MTD strategies with or without DSEOM. Specifically, we generated two kinds of update events, service scaling and service upgrading, for each service. Respectively, these update events will result in the number of service instances increasing (event No. 1, 5, 9, 13) or decreasing (event No. 2, 6, 10, 14), and the attack difficulty of services increasing (event No. 3, 7, 11, 15) or decreasing (event No. 4, 8, 12, 16). Fig. 6 shows the effectiveness of MTD strategies after these update events happened on *Apache* (No.1-4), *Tomcat* (No.5-8), *ImageMagick* (No.9-12) and *Memcached* (No.13-16). We observe that parts of update events make the effectiveness of invariable MTD strategies very volatile, but DSEOM can effectively maintain its effectiveness stable in these cases.

More specifically, in the case of *Apache* instances increasing (event No.4), the increased costs of MTD implementing on *Apache* directly causes the effectiveness of MTD decreasing. Under this situation, the MTD is re-deployed on *ImageMagick* based on optimization of DSEOM to maintain its effectiveness. Also, when the instances of *ImageMagick* decreasing (event No.11), the MTD can also be deployed on it concurrently, which make the MTD strategies are more optimal with the acceptable costs. In other cases (event No.5 and 9), *Tomcat* and *ImageMagick* respectively become new weak point of the system when its attack difficulty sharply declined due to upgrading, which causes the effectiveness of MTD falling badly. Therefore, DSEOM re-deployed MTD on them to maximize the effectiveness of MTD. Nevertheless, the changes introduced by other update events would not alter the weakness of the system, so the effectiveness of MTD strategies just fluctuate within the acceptable range.

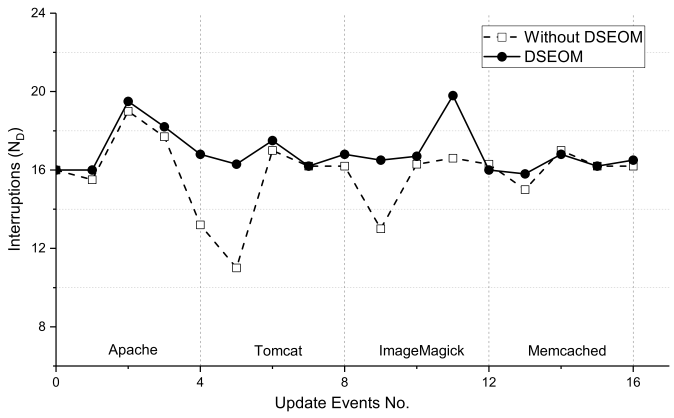


Fig. 6: The average number of interruptions in the attack process after updates occurred.

6.4 Scalability Evaluation

Generally, the microservice has a larger scale and more complex structure in the real world (e.g., Uber's application is composed of over 1,000 microservices [3]). Accordingly, we conducted a set of simulations to evaluate the scalability of DSEOM with different shortest path algorithms in the large scale microservice scenarios. In our simulation, we randomly generate a set of microservice architectures where the number of microservices and the topology of microservice structure are different. Fig. 7 shows the scalable properties of DSEOM for different microservice scale ranging from

100 to 1,000. In different microservice scale, we randomly generated 300 updates and measured the average time needed by DSEOM for optimizing MTD strategies. And the frequency of these updates for microservices follows the Poisson distribution. Based on these results, we can determine that using the modified fully dynamic algorithm (D-KIN) in DSEOM has a better effect and can support the case with 1,000 microservices.

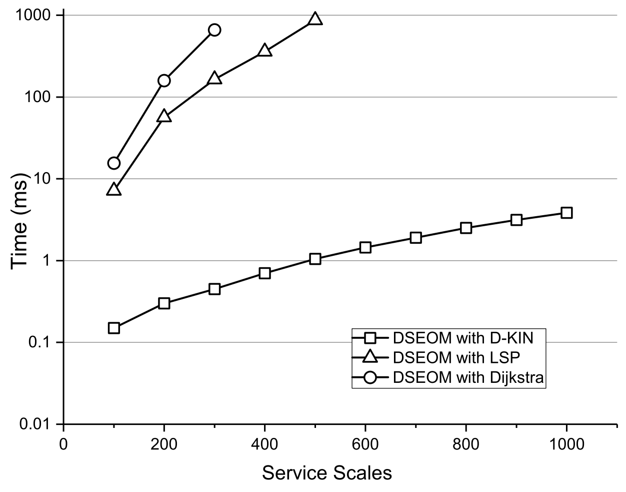


Fig. 7: Scalability for different shortest path algorithms.

7 Discussion

In this section, we discuss limitations of our work that will be resolved in our future works.

Performance impact of MTD techniques. In this paper, we have only taken into account the performance impact on the services where MTD techniques deployed. However, MTD techniques will also indirectly influence other services because its implementation needs to consume some extra system resources. To this limitation, we will incorporate this performance parameter into our model and perform an evaluation in a real system to determine the performance impact for all services caused by MTD techniques. Due to the complexity of microservice architecture, there is also a challenge on how to effectively conduct the evaluation.

Various MTD techniques hybrid deployment. Our framework only supports the fixed deployment pattern for MTD techniques on the cloud platform. But sometimes different MTD techniques should be flexibly deployed on different kinds of services to defeat specific attacks. For example, ip shuffling usually focus to help web services defend against the attack from the network, such as network sniffing and remote code execution etc. However, it is difficult to decide what kind of MTD techniques applies to given targets, as different targets may suffer from various attacks. Another reason is that the different MTD techniques being mixed-used may have conflicts with each other. We will analyze and solve these problems in our future works.

8 Related Work

Container Security: The security of the container-based cloud environment is always an important research area as container technology has been widely adopted in mainstream cloud platforms. Tak et al. [23] performed an analysis on the security issues brought by the high degree of agility, reusability, and portability of containers. This research reveals that the high dynamism of container-based cloud environment may cause disruptive drift problems which will introduce vulnerabilities unintentionally and make service provider lose control of the deployed software. Moreover, the security issue for isolation of container is widely concerned, Gao et al. [46] systematically identified the information leakage problem caused by the weak isolation between the container and the hosts. Based on these leakage channels, they investigated potential container-based power-attack threats in the cloud platform. To solve the isolation problems, Arnautov et al. [19] used the SGX to enhance the isolation of container for protecting container processes from outside attacks.

MTD Evaluation: Since the MTD techniques are applied in various IT system in recent years, how to accurately evaluate the effectiveness of MTD implementation has become a key issue. Some researches use security metrics to quantitatively assess the effectiveness of MTD techniques. Okhravi et al. [47] regarded the coverage, unpredictability, and timeliness as metrics to evaluate effective movement. Taylor et al. [48], [49] designed a collection of metrics, including success, integrity, productivity, and confidentiality, to assess the costs and benefits of MTD. Picek et al. [50] presented several metrics, such as amplitude, relative variation of the amplitude, average gaps of the relative gaps and average length of the walks, to infer more information about the effectiveness of MTD. Other researches use the analytical model to evaluate the effectiveness of MTD techniques. Okhravi et al. [14] quantitatively evaluated the dynamic platform techniques as a defensive mechanism by proposing a generalized evaluation model using the Markov chain. Maleki et al. [51] also used a discrete-time Markov chain to model the interactions between attacker and defender in MTD games. But it cannot predict the performance of an individual MTD. To this limitation, Connell et al. [52] introduced a quantitative analytic model, Continuous Time Markov Chains (CTMC), to assess the resource availability and performance of MTD. Though these methods can accurately evaluate the effectiveness of MTD, the efficiency of them can't be applicable to high dynamic environment.

MTD Strategies Design: Existing works also focus on designing the implementation strategies of MTD techniques to maximize effectiveness. Bardas et al. [12] presented a MTD platform that can capture service dependencies on the cloud environment and find the best strategy for live instance replacements to maximize the attack difficulty. Feng et al. [53] proposed two-player Bayesian Stackelberg game to improve the MTD strategy by disclosing strategic information. Hong et al. [13] used security models, hierarchical attack representation model (HARM), to analyze and assess the effectiveness of different MTD strategies. In addition, they use importance measures (IMs) to determine the important nodes and deploy the MTD techniques in an effective manner. But in their model, they didn't consider the performance damage (e.g. service performance declining caused by the MTD implementing) and the implementation costs of MTD techniques (e.g. resource costs, time costs).

Different from these works, we select targets to deploy MTD techniques according to comprehensive consideration of the importance of targets and the performance issues mentioned above. Moreover, based on the attackers' behavior pattern, we introduce the concept of betweenness centrality to determine the importance of targets, and propose the multi-dimensional attack graphs to revise the betweenness centrality of each target. The importance of targets computed by this method is more accurate than straightly computed by IMs. At last, our framework can continually and rapidly optimize the MTD strategies, which is more applicable to container-based cloud than other works.

9 Conclusion

With the combination of container technology and microservice architecture, the cloud environment is getting more efficient and dynamic. However, the high dynamism makes the effectiveness of defense mechanisms, such as MTD technique, drift over time. Unfortunately, existing solutions for evaluating and optimizing the effectiveness of MTD techniques can't be effectively applied to this dynamic scenario, especially when the complexity of microservices increases.

In this paper, we present a framework called DSEOM for automatically evaluating and optimizing the effectiveness of MTD techniques in the container-based cloud environment at real time. We establish a holistic attack graph model and multi-dimensional attack graph model to formalize attack scenarios in the container-based cloud environment. Based on the attackers' behavior pattern, we introduce *betweenness centrality* combining with the attack graph models to rapidly locate the weak points in these attack scenarios, and optimize the MTD implementation strategies. Further, we incorporate two kinds of MTD techniques, ip shuffling and live migration, into DSEOM and demonstrate the existence of effectiveness drift problem in the container-based cloud environment. Our evaluation results show that DSEOM can rapidly provide the near optimal implementation strategies of MTD techniques under the constraints of overhead after each update.

Acknowledgments

This paper was supported by the National Key Research & Development (R&D) Plan of China under grant No. 2017YFB0802205, the China Postdoctoral Science Foundation funded Project under grant No. 2018M640701, and the First Class Special funding for Postdoctoral Scientific Research of Hubei Province under grant No.2018-G22.

References

[1] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. C. Bavier, and L. L. Peterson, "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors," in *Proceedings of European Conference on Computer Systems*, 2007, pp. 275–287.

[2] J. Thones, "Microservices," *IEEE Software*, vol. 32, no. 1, pp. 116–116, 2015.

[3] A. Panda, M. Sagiv, and S. Shenker, "Verification in the age of microservices," in *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, 2017, pp. 30–36.

[4] N. Dragoni, I. Lanese, S. T. Larsen, M. Mazzara, R. Mustafin, and L. Safina, "Microservices: How to make your application scale," in *Proceedings of the 11th International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, 2017, pp. 95–104.

[5] K. A. Torkura, M. I. H. Sukmana, and C. Meinel, "Integrating continuous security assessments in microservices and cloud native applications," in *Proceedings of the 10th International Conference on Utility and Cloud Computing*, 2017, pp. 171–180.

[6] V. Heorhiadi, S. Rajagopalan, H. Jamjoom, M. K. Reiter, and V. Sekar, "Gremlin: Systematic resilience testing of microservices," in *Proceedings of the 36th IEEE International Conference on Distributed Computing Systems*, 2016, pp. 57–66.

[7] K. Lu, C. Song, B. Lee, S. P. Chung, T. Kim, and W. Lee, "Aslr-guard: Stopping address space leakage for code reuse attacks," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 280–291.

[8] P. Larsen, A. Homescu, S. Brunthaler, and M. Franz, "Sok: Automated software diversity," in *Proceedings of the 35th IEEE Symposium on Security and Privacy*, 2014, pp. 276–291.

[9] L. Wang and D. Wu, "Moving target defense against network reconnaissance with software defined networking," in *Proceedings of the 19th International Conference on Information Security*, 2016, pp. 203–217.

[10] Y. Luo, B. Wang, X. Wang, B. Zhang, and W. Hu, "RPAH: A moving target network defense mechanism naturally resists reconnaissances and attacks," *IEICE Transactions on Information and Systems*, vol. 100-D, no. 3, pp. 496–510, 2017.

[11] C. Huang, S. Zhu, and Y. Yang, "An evaluation framework for moving target defense based on analytic hierarchy process," *ICST Transactions on Security Safety*, vol. 4, no. 13, pp. 1–12, 2018.

[12] A. G. Bardas, S. C. Sundaramurthy, X. Ou, and S. A. DeLoach, "MTD CBITS: moving target defense for cloud-based IT systems," in *Proceedings of the 22nd European Symposium on Research in Computer Security*, 2017, pp. 167–186.

[13] J. B. Hong and D. S. Kim, "Assessing the effectiveness of moving target defenses using security models," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 2, pp. 163–177, 2016.

[14] H. Okhravi, J. Riordan, and K. M. Carter, "Quantitative evaluation of dynamic platform techniques as a defensive mechanism," in *Proceedings of the 17th International Symposium on Research in Attacks, Intrusions and Defenses*, 2014, pp. 405–425.

[15] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. lightweight virtualization: A performance comparison," in *Proceedings of IEEE International Conference on Cloud Engineering*, 2015, pp. 386–393.

[16] "Docker," 2018. [Online]. Available: https://www.docker.com

[17] "Kubernetes," 2018. [Online]. Available: https://kubernetes.io

[18] "Microservice," 2018. [Online]. Available: http://microservice.io

[19] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. Stillwell, D. Goltzsche, D. M. Eyers, R. Kapitza, P. R. Pietzuch, and C. Fetzer, "SCONE: secure linux containers with intel SGX," in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*, 2016, pp. 689–703.

[20] H. Jin, W. Dai, and D. Zou, "Theory and methodology of research on cloud security," *SCIENCE CHINA Information Sciences*, vol. 59, no. 5, pp. 050 105:1–050 105:3, 2016.

[21] N. Dragoni, S. Giallorenzo, A. Lluch-Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: Yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*, 2017, pp. 195–216.

[22] K. A. Torkura, M. I. H. Sukmana, and C. Meinel, "Integrating continuous security assessments in microservices and cloud native applications," in *Proceedings of the 10th International Conference on Utility and Cloud Computing*, 2017, pp. 171–180.

[23] B. Tak, C. Isci, S. S. Duri, N. Bila, S. Nadgowda, and J. Doran, "Understanding security implications of using containers in the cloud," in *Proceedings of USENIX Annual Technical Conference*, 2017, pp. 313–319.

[24] R. Zhuang, S. A. DeLoach, and X. Ou, "Towards a theory of moving target defense," in *Proceedings of the 1st ACM Workshop on Moving Target Defense*, 2014, pp. 31–40.

[25] A. Homescu, T. Jackson, S. Crane, S. Brunthaler, P. Larsen, and M. Franz, "Large-scale automated software diversity - program evolution redux," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 2, pp. 158–171, 2017.

[26] G. Dsouza, S. Hariri, Y. B. Al-Nashif, and G. Rodríguez, "Resilient dynamic data driven application systems (rdddas)," in *Proceedings of the 13th Annual International Conference on Computational Science*, 2013, pp. 1929–1938.

[27] B. Danev, R. J. Masti, G. Karame, and S. Capkun, "Enabling secure vm-vtpm migration in private clouds," in *Proceedings of the 27th Annual Computer Security Applications Conference*, 2011, pp. 187–196.

[28] S. 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*, 2015, pp. 1595–1606.

[29] S. Antonatos, P. Akritidis, E. P. Markatos, and K. G. Anagnostakis, "Defending against hitlist worms using network address space randomization," *Computer Networks*, vol. 51, no. 12, pp. 3471–3490, 2007.

[30] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "An effective address mutation approach for disrupting reconnaissance attacks," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 12, pp. 2562–2577, 2015.

[31] V. Heydari, S. Kim, and S. Yoo, "Scalable anti-censorship framework using moving target defense for web servers," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 5, pp. 1113–1124, 2017.

[32] S. Debroy, P. Calyam, M. Nguyen, A. Stage, and V. Georgiev, "Frequency-minimal moving target defense using software-defined networking," in *Proceedings of International Conference on Computing, Networking and Communications*, 2016, pp. 1–6.

[33] H. Wang, F. Li, and S. Chen, "Towards cost-effective moving target defense against ddos and covert channel attacks," in *Proceedings of the 3rd ACM Workshop on Moving Target Defense*, 2016, pp. 15–25.

[34] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains," *Leading Issues in Information Warfare & Security Research*, vol. 1, no. 1, p. 80, 2011.

[35] N. Poolsappasit, R. Dewri, and I. Ray, "Dynamic security risk management using bayesian attack graphs," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 1, pp. 61–74, 2012.

[36] "Common vulnerability scoring system v3.0: Specification document," 2018. [Online]. Available: https://www.first.org/cvss/specification-document

[37] C. Demetrescu and G. F. Italiano, "Experimental analysis of dynamic all pairs shortest path algorithms," *ACM Transactions on Algorithms*, vol. 2, no. 4, pp. 578–601, 2006.

[38] V. King, "Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs," in *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, 1999, pp. 81–91.

[39] "Swarm mode overview," 2018. [Online]. Available: https://docs.docker.com/engine/swarm

[40] "CRIU Docker," 2017. [Online]. Available: https://criu.org/Docker

[41] "Prometheus overview," 2018. [Online]. Available: https://prometheus.io/docs/introduction/overview/

[42] "National vulnerability database," 2018. [Online]. Available: https://nvd.nist.gov/vuln

[43] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, P. Ingram, E. J. Jackson *et al.*, "Network virtualization in multi-tenant datacenters," in *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation*, 2014, pp. 203–216.

[44] G. Kakivaya, L. Xun, R. Hasha, S. B. Ahsan, T. Pfleiger, R. Sinha, A. Gupta, M. Tarta, M. Fussell, V. Modi *et al.*, "Service fabric: a distributed platform for building microservices in the cloud," in *Proceedings of European Conference on Computer Systems*, 2018, pp. 1–15.

[45] "Apache http server benchmarking tool," 2018. [Online]. Available: https://httpd.apache.org/docs/2.4/programs/ab.html

[46] X. Gao, Z. Gu, M. Kayaalp, D. Pendarakis, and H. Wang, "Containerleaks: Emerging security threats of information leakages in container clouds," in *Proceedings of the 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2017, pp. 237–248.

[47] T. Hobson, H. Okhravi, D. Bigelow, R. Rudd, and W. W. Streilein, "On the challenges of effective movement," in *Proceedings of the 1st ACM Workshop on Moving Target Defense*, 2014, pp. 41–50.

[48] J. Taylor, K. Zaffarano, B. Koller, C. Bancroft, and J. Syversen, "Automated effectiveness evaluation of moving target defenses: Metrics for missions and attacks," in *Proceedings of the 3rd ACM Workshop on Moving Target Defense*, 2016, pp. 129–134.

[49] K. Zaffarano, J. Taylor, and S. Hamilton, "A quantitative framework for moving target defense effectiveness evaluation," in *Proceedings of the 2nd ACM Workshop on Moving Target Defense*, 2015, pp. 3–10.

[50] S. Picek, E. Hemberg, and U. O'Reilly, "If you can't measure it, you can't improve it: Moving target defense metrics," in *Proceedings of the 4th ACM Workshop on Moving Target Defense*, 2017, pp. 115–118.

[51] H. Maleki, S. Valizadeh, W. Koch, A. Bestavros, and M. van Dijk, "Markov modeling of moving target defense games," in *Proceedings of the 3rd ACM Workshop on Moving Target Defense*, 2016, pp. 81–92.

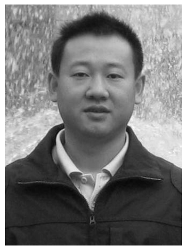
[52] W. Connell, D. A. Menascé, and M. Albanese, "Performance modeling of moving target defenses," in *Proceedings of the 4th ACM Workshop on Moving Target Defense*, 2017, pp. 53–63.

[53] X. Feng, Z. Zheng, D. Cansever, A. Swami, and P. Mohapatra, "A signaling game model for moving target defense," in *Proceedings of the 36th IEEE Conference on Computer Communications*, 2017, pp. 1–9.

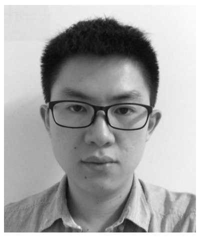
Hai Jin is a Cheung Kung Scholars Chair Professor of Computer Science and Engineering at Huazhong University of Science and Technology. Jin received his Ph.D. in computer engineering from Huazhong University of Science and Technology in 1994. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz in Germany. Jin worked at The University of Hong Kong between 1998 and 2000, and as a visiting scholar at the University of Southern California between 1999 and 2000. He was awarded Excellent Youth Award from the National Science Foundation of China in 2001. Jin is the chief scientist of ChinaGrid, the largest grid computing project in China, and the chief scientists of National 973 Basic Research Program Project of Virtualization Technology of Computing System, and Cloud Security.

Jin is a IEEE Fellow, CCF Fellow, and a member of the ACM. He has co-authored 22 books and published over 700 research papers. His research interests include computer architecture, virtualization technology, cluster computing and cloud computing, peer-to-peer computing, network storage, and network security.

Zhi Li is currently working toward the Ph.D. degree in computer science and technology at Huazhong University of Science and Technology (HUST), Wuhan, China. His research interests include security in system architecture, container technique and cloud computing.



Deqing Zou is a Professor of Computer Science at Huazhong University of Science and Technology. He received his Ph.D at Huazhong University of Science and Technology in 2004. His main research interests include system security, trusted computing, virtualization and cloud security. He has more than 50 high-quality papers, including papers published by NDSS, ACSAC, TPDS, TDSC and so on. He has always served as a reviewer for several prestigious Journals, such as IEEE TPDS, IEEE TOC, IEEE TDSC, IEEE TCC, and so on. He is on the editorial boards of four international journals, and has served as PC chair/PC member of more than 40 international conferences.



Bin Yuan Bin Yuan is currently a postdoc at Huazhong University of Science and Technology (HUST), Wuhan, China. Bin received his B.S. and Ph.D degree in Computer Science and Technology from HUST in 2013 and 2018, respectively. His research interests include SDN security, NFV, Cloud Security, Privacy and IoT security. He has published several technical papers in top journals, such as TSC, TNSM, FGCS and IEEE Access.