2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)

2020 IEEE 22nd International Conference on High Performance Computing and Communications | 978-1-7281-7649-9/20/$31.00 ©2020 IEEE | DOI: 10.1109/HPCC-SmartCity-DSS50907.2020.00104

ConfigRand: A Moving Target Defense Framework against the Shared Kernel Information Leakages for Container-based Cloud

Tong Kong*†,Liming Wang*‡,Duohe Ma*,Kai Chen*, Zhen Xu*, Yijun Lu⋆
* Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China,100093
† School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China,100049
⋆ Alibaba Cloud Computing Co. Ltd., Hangzhou, China,310099
{kongtong,wangliming,maduohe,chenkai7274,xuzhen}@iie.ac.cn,yijun.lyj@alibaba-inc.com
‡ Corresponding Author

***Abstract*—Lightweight virtualization represented by container technology provides a virtual environment for cloud services with more flexibility and efficiency due to the kernel-sharing property. However, the shared kernel also means that the system isolation mechanisms are incomplete. Attackers can scan the shared system configuration files to explore vulnerabilities for launching attacks. Previous works mainly eliminate the problem by fixing operating systems or using access control policies, but these methods require significant modifications and cannot meet the security needs of individual containers accurately. In this paper, we present ConfigRand, a moving target defense framework to prevent the information leakages due to the shared kernel in the container-based cloud. The ConfigRand deploys deceptive system configurations for each container, bounding the scan of attackers aimed at the shared kernel. In design of ConfigRand, we (1) propose a framework applying the moving target defense philosophy to periodically generate, distribute, and deploy the deceptive system configurations in the container-based cloud; (2) establish a model to formalize these configurations and quantify their heterogeneity; (3) present a configuration movement strategy to evaluate and optimize the variation of configurations. The results show that ConfigRand can effectively prevent the information leakages due to the shared kernel and apply to typical container applications with minimal system modification and performance degradation.**

***Index Terms*—Cloud Computing Security; Container; Moving Target Defense; Network Security**

I. Introduction

Container technology has gained ground as a lightweight virtualization solution to replace the Virtual Machine (VM) in some new cloud infrastructures [1], such as Kubernetes [2], ECI [3], and ECS [4]. Different from the VM, container works at the operating system level and the containers running on one host can share the same kernel. This is because container technology virtualizes the operating environment by using some key modules in the Linux kernel such as system isolation technology Namespace [5], resource control technology Cgroup [6], and security systems (e.g., Capabilities [7]). By characteristics of their design, containers have better resource utilization and lower virtualization overhead. These benefits also promote the fast development of microservice architecture [8] and serverless computation (e.g., Amazon Lambda).

With the extensive usage of container technology, there always exist concerns that whether the extension of container services would affect the security of cloud systems. To support multi-tenancy on cloud services, container technology intensifies the isolation among co-resident containers and supports unprivileged user-level containers. However, some subsystems are not be improved for the applicability of container due to both the low utilization and the difficulty to transform these code bases. Unfortunately, some information about these subsystems are stored in corresponding system configuration files, and they might expose system-wide information to containers. These loopholes could make malicious adversaries much easier to gather sensitive information of cloud systems and launch subsequent attacks, even gaining root-level access on the host (e.g., CVE-2019-5736). Moreover, the information that looks innocuous or useless for most services such as the workload statistics data also might be useful for attackers. They can use the information to identify which containers are running on the same or adjacent servers and launch power attacks, leading to forced shutdowns [9].

A straightforward solution to this problem is to restrict the access of container to these system-wide configuration files. Some previous works focus on this solution by using access control policies or fixing the namespaces of the Linux kernel. These methods could only control the data access on the file-level. However, some subsystems of the Linux kernel are under unified use by the host and containers. In some system files, some contents are essential for the container running, but others are container-oblivious and expose system-wide information. In this case, these methods could only make limited kernel security features available to containers and are not suitable for practical deployment. In our work, we approach this problem from a completely different perspective. We can change these configuration files to provide deceptive configurations, luring attackers to get untrue information and inhibit the attack process. The effectiveness of this method has been demonstrated by Moving Target Defense (MTD) technology [10]. By changing the attack surface of the target system

978-1-7281-7649-9/20/$31.00 ©2020 IEEE
DOI 10.1109/HPCC-SmartCity-DSS50907.2020.00104

dynamically, defenders could disrupt attackers' exploration and cognition to the system. Therefore, the work effort (e.g., the cost and complexity) for the attackers will increase, and the probability of successful attacks will reduce dramatically.

In this paper, we propose ConfigRand, a moving target defense framework using system configuration shift to study and solve these problems mentioned above. The ConfigRand could establish the mapping between the shared system configurations and deceptive system configurations for each container, shift these configurations with maximum diversification, and change the attack surface of the target system to reduce the probability of successful attacks in the container-based cloud. To filter the appropriate system configurations for shifting, we analyze the root causes of correlative information leakages. Meanwhile, we present a theoretical model to formalize these configurations and evaluate the heterogeneity of different configuration combinations. To move the attack surface with maximum diversification, we propose a configuration movement strategy combining cluster analysis to optimize the conversion of deceptive system configurations between different configuration combinations. From this, ConfigRand can dynamically shift the attack surface, make the targeted system more difficult for attackers to strike, and achieve maximum defense benefit.

In summary, our main contributions are listed as follows:

1) We design an MTD framework to defend against the shared kernel information leakages for the container-based cloud by deploying deceptive system configurations for containers, shifting these configurations with maximum diversification, changing the attack surface of the target system.
2) We present a model to formalize the shared system configurations and assess the heterogeneity of different configuration combinations as the basis of configuration movement and cluster analysis.
3) We propose a configuration movement strategy combining the cluster analysis to optimize the shift of these configurations and attack surface, decreasing the success rate of attacks by more than 70%.

The rest of the paper is organized as follows. Section II discusses the root cause of container information leakages and introduces attacks based on this information. Section III analyzes the challenges about solving these problems and introduces the model we formalize. Section IV presents our ConfigRand framework and describes the design of algorithms used in the configuration movement strategy. Section V implements the ConfigRand and evaluates its effectiveness. Section VI gives a survey of previous work on MTD. Finally, we conclude in Section VII.

II. Background

A. Container Isolation Mechanism

Container technology uses OS-level virtualization to make computational resources available to tenants as a service. Different from VM, container needs no hypervisors or an exclusive OS. Instead, it uses the host OS to manage system resources and execute system calls, and uses kernel features to enforce isolation among user-space instances. For example, LXC [11] and Docker [12] use some Linux kernel features such as Namespace [5], Cgroups [6] and Capability [7] to support their container creation.

Here we introduce one of the key technologies, Namespace. The original purpose of Namespace is to isolate system resources for different processes, so specified processes will have a dedicated view of system resources. Any modification to one namespace can only influence the processes related to it. This feature makes the system could modify the configuration of specified processes without system-wide environmental change. Due to this characteristic, namespace is also used to create containers. Now, most container engines use six kinds of namespaces: Ipc, Net, Mnt, Pid, User, and Uts. In brief, these namespaces isolate the interprocess communication, network device, file system mount point, process identifier, uid&gid, and host&domain name among different containers on a same host. However, some Linux subsystems are not (fully) namespaced due to the priority and difficulty consideration. This also leads to security holes about information leakages or co-resident attacks.

B. Information Leakages and Attacks

The memory-based pseudo file system is one kind of controlled interface of Linux, which is used to connect the user mode and kernel mode. Linux uses many memory-based pseudo file systems such as *procfs*, *sysfs*, *devfs*, and *debugfs* to support multiple kernel operations. Some information about theses systems is saved as configuration files in memory, so the kernel data can be manipulated by normal file I/O operations. Moreover, as illustrated in Fig. 1, some file systems have been isolated and others have not [9]. This could lead to diverse problems in the container-based cloud.

(1) Gao et al. validated the system-wide information leakages among host OS and containers due to the system configuration files of memory-based pseudo file systems. They discovered 21 leakage channels about various types of system-wide information including kernel data structures, performance statistics data, kernel events, and hardware information [9].

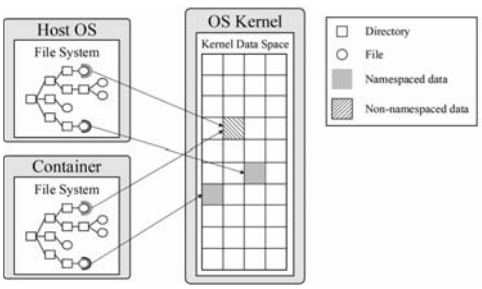

Fig. 1: The reason of information leakages in the container-based cloud

(2) Attackers could use the vulnerability in non-namespaced subsystems to launch attacks, even gaining root-level access on the host. Researchers declared that attackers can inject malicious code to */proc/self/exe*, which is a shared file in *procfs*. When runC executes this file, attackers could compromise the system (e.g. CVE-2019-5736).
(3) Moreover, the information that looks innocuous also might be useful for attackers. For example, the workload statistics data can be used to identify which containers are co-resident or run on adjacent servers that locate on the same rack or power distribution unit. Then, the attacker can launch power attacks, leading to forced shutdowns [9].

III. Problem Overview

A. Problem Analysis

The information leakages and related attacks bring great security threats for container-based clouds. We give two case studies to discuss the challenges of traditional defense methods in solving these problems and analyze the advantages of the MTD framework we propose to overcome them.

Case study I: The most common method is using the access control mechanism to regulate which containers can view or use the system configuration files. However, this method can only prevent unauthorized access but not solve the problem of shared files fundamentally. Meanwhile, the access control mechanism cannot satisfy the needs of some conditions. For example, the */proc/cpuinfo* contains information about all the CPUs on a computer. The information about CPUs that are assigned to a container is essential for the container running, but others will leak system-wide information to this container. The access control mechanism could only control the data access on the file-level, so it's not fine-grained enough to solve this problem.

Case study II: Another straightforward solution to this problem is deleting the configuration parameters that contains system-wide information to prevent the information leakages. These configuration parameters are seemingly useless for containers and applications running on them. For example, due to the cross-platform characteristic of the container technology, the user can create a Ubuntu container on a host that uses CentOS. When the container user accesses the */proc/version*, the Linux release still shows as CentOS. It's because that containers and host share the */proc/version*. This also means that the wrong information would not affect normal operations. However, deleting these configuration parameters is still not acceptable. Because some function calls will extract the whole configuration files to compose a specifies structure and then select the parameters they are interested in. Although some configuration parameters are not used directly by containers and their applications, deleting them will break the structure and cause exception events.

The ConfigRand framework we propose in this paper has some advantages in overcoming the above challenges. Firstly, we create configuration files that leak system-wide information for each container individually. In this way, the problem of shared files will be solved fundamentally. Even attackers compromise a container, they can only modify the files belonging to this container and cannot inject malicious codes to threaten the host OS. Moreover, we replace container-oblivious system configurations with deceptive configurations and shift them periodically based on the MTD philosophy [13]. This method doesn't cut down the information content that containers have, so it would not affect the normal operations. In this way, we could prevent attackers from getting sensitive information and exploring vulnerabilities of the system. Meanwhile, the MTD technology enlarges and changes the exploration surface as shown in Fig. 2, making attackers spend more energy to repeat the exploration and increasing the difficulty of penetrating the system.

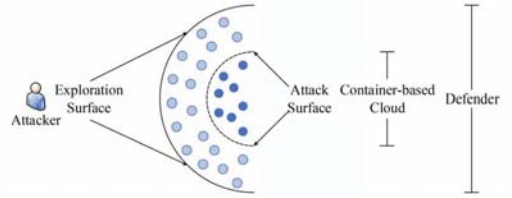

Fig. 2: Attack surface and exploration surface

B. Threat Model

We discuss the purposes and capabilities of adversaries in this subsection. In our threat model, the cloud operator and container image provider are trusted, and adversaries act as tenants or come from the external network. The purpose of adversaries is to compromise containers for stealing important data and launch subsequent attacks in the cloud system. We consider an adversary model with the following characteristics:

Attack targets: The container technology did not isolate some subsystems thoroughly, so the configuration files of these subsystems have potential leakage channels. And the system-wide information is the targets of adversaries. Meanwhile, the cloud system exists some loopholes that could be used by adversaries for launching attacks.

Adversary's Capability: We analyze the capability of adversaries in the following aspects.

- *External Network:* There are some public service applications in containers that could be connected to the external network. We assume that adversaries could discover these applications and launch attacks on them.
- *Internal Network:* If adversaries have the privilege to create containers in the cloud, they could access the configuration files of these containers without any other permission or operation.
- *Limitation:* The information about the cloud system is almost unknown for adversaries before they explore the system. This information includes the construction of the cloud, the location of services in the system, and the replicas of each service. Besides, the security mechanisms of the system and the security strategies of containers are also unknown for adversaries.

TABLE I: Variables definition

Name	Definition
l_i	The number i configuration parameter type
L	The configuration list
$v_{i,j}$	The number j configuration parameter of l_i
V_i	The range of configuration parameter type l_i
C	The configuration combination
S	The configuration space
$D(v_{i,e}, v_{i,f})$	The Levenshtein distance between $v_{i,e}$ and $v_{i,f}$
$H(C_x, C_y)$	The heterogeneity between C_x and C_y

C. Problem Formulation

According to the threat model, we formally make some definitions to cover overall shared subsystem configurations. Then, we introduce the concept of Levenshtein Distance to assess the heterogeneity of different configuration combinations. The basic parameters of the model are defined in Table. I.

According to Sect. II, the configuration files of non-namespaced subsystems may lead to information leakages. We analyze these files and extract container-oblivious configuration parameter types. These configurations are the targets of MTD strategy to shift and also the reflection of attack surface.

Definition 1: Configuration List $L = \{l_1, l_2, l_3, ..., l_n\}$ is a set, where l_i denotes a configuration parameter type.

Actually, each configuration parameter is a variable that could be changed by the system assignment. These parameters are used to denote an attribute of the hardware, OS, or software and have to be valid. In a practical system, these parameters may belong to the string type, numeric type, boolean type, etc. In this model, we transform them into the string type for consistency. These parameters are also the minimal grained elements in our framework for the configuration shift.

Definition 2: $V_i = \{v_{i,1}, v_{i,2}, v_{i,3}, ..., v_{i,m_i}\}$ is the range of configuration parameter type l_i, where $v_{i,j}$ is the value extracted from practical system and be of string type, and $1 \leq i \leq n$, $1 \leq j \leq m_i$.

Although each configuration parameter type has various values in different containers and periods, the value must be certain in a specific state. To capture the configuration state, we define the notion of configuration combination which contains the certain values of every configuration parameter type in L. A configuration combination is also a valid and alternative state for MTD strategy for shifting.

Definition 3: A configuration combination, $C = \{v_{1,j_1}, v_{2,j_2}, v_{3,j_3}, ..., v_{n,j_n}\}$, is a set of configuration parameters that denotes a specific configuration state of a container, and $v_{i,j} \in V_i$.

One significant step of the attack process is exploring the target system's exploration surface. In this work, the configuration space which contains all the valid configuration combinations is a main part of the exploration surface. Extending the configuration space of our target system will lead to the extension of exploration surface.

Definition 4: The configuration space of the MTD system, S, is the domain of configuration combination, where $C \in S$.

To evaluate the heterogeneity of different configuration combinations, we introduce the concept of Levenshtein distance [14]. It's a string metric for measuring the difference between two character strings in information theory, linguistics, and computer science. Specifically, the Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions, or substitutions) required to change one word into the other. In this model, we use Levenshtein distance (D) to calculate the heterogeneity of two configuration parameters, where a, b denote the character strings of two configuration parameters and $a(i)$, $b(j)$ denote the first i characters of a and the first j characters of b.

$$G(a_i, b_j) = \begin{cases} 0 & \text{if } a_i = b_j \\ 1 & otherwise \end{cases} \tag{1}$$

$$D(a,b) = \begin{cases} \max(len(a), len(b)) & \text{if } \min(len(a), len(b)) = 0 \\ \min(D(a(i-1), b(j)) + 1, & \\ \quad D(a(i), b(j-1)) + 1, & \text{otherwise} \\ \quad D(a(i-1), b(j-1)) + G(a_i, b_j)) & \end{cases} \tag{2}$$

After the heterogeneity of each pair of configuration parameters calculated, we use the weighted sum of them to denote the heterogeneity of two configuration combinations. The weight of each parameter α is related to the parameter's influence on information leakages. The heterogeneity (H) between two configuration combinations C_x and C_y is defined by (3).

$$H(C_x, C_y) = \sum_{i=1}^{n} \alpha_i \cdot D(v_{i,x_i}, v_{i,y_i}) \tag{3}$$

The main operation of an MTD system is transforming a container from one configuration state to another. To increase the uncertainty of the system, the strategy is supposed to select a new configuration state that is more different from the old one. In this case, the heterogeneity is used to evaluate the difference between configuration states in the MTD system.

IV. ConfigRand Framework

In this section, we show the design details of the ConfigRand framework, propose a configuration movement strategy to optimize the conversion of deceptive system configurations, and analyze the feasibility of the framework on defending against attackers.

A. Framework Design

We design the ConfigRand framework that can deploy deceptive system configurations for each container, shift these configurations, and change the attack surface of the target system to protect the cloud from attacks. The overall framework architecture is shown in Fig. 3. This framework consists of four modules, Monitor Engine, Configuration Generator, Randomization Controller, and Deployment Handler.

The Monitor Engine is used to read the configuration files and collect configuration states of containers in real time. We only focus on the files which lead to the information leakage problems and these files have been validated by [9]. The result includes many configuration files in *procfs* and *sysfs*. Because these subsystems share the same configuration files. We need

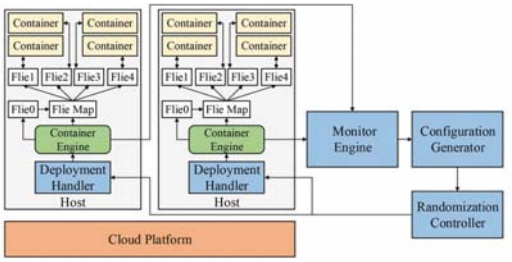

Fig. 3: ConfigRand framework overview

to isolate the integrated condition for providing deceptive system configurations for each container individually. In this framework, we modify the file mapping mechanism of the container engine to achieve this goal. The existing file mapping mechanism is shown in Fig. 1, and we remap the paths of shared configuration files and point them to exclusive configuration files for each container. This modification uses Filesystem in Userspace (FUSE) [15], which is a software interface for Unix and Unix-like computer operating systems. This technology makes non-privileged users can establish their own file systems without editing kernel code. Meanwhile, this module supervises these files, gathers related configurations, and transforms them into a configuration combination.

The Configuration Generator could generate possible configuration combinations for containers. Because all systems and applications have constraints and resource limitations, we must ensure that the new configurations will be valid. This check could be implemented by the expert knowledge base , which contains a collection of information such as rules, facts, and descriptions of the system configurations' dependencies and confliction. It's used to automatically filter the generated configuration combination and reserve the valid one, helping us determine the usages of configurations and the influences if we change them. Firstly, the Configuration Generator generates different valid parameters for each configuration under the guidance of the expert knowledge base. Then, these different valid parameters will be mixed randomly to make multiple diversified and uncertain configuration combinations. Atfer this, the expert knowledge base will filter these configuration combinations again to match the requirement of dependencies and confliction, and determine changing these configurations will not affect the normal running of the current system condition.

Based on the valid configuration combinations generated, the Randomization Controller will select the next configuration state to move to periodically. This operation is essential for an MTD framework to make attackers more difficult to compromise the system. Therefore, deciding what state to move to is a crucial problem for the framework. To solve this problem, we propose a configuration movement strategy. More details on this strategy will be described in the next subsection.

The Deployment Handler automatically deploys and activates the new configuration state selected by the Randomization Controller. This module accepts two parts of information from the Randomization Controller, the configuration combination and corresponding identification of the container. Then, the module extracts configuration parameters from the configuration combination and writes them to the matched configuration files for each container.

B. Configuration Movement Strategy

The configuration movement strategy is based on the problem formulation for evaluating and optimizing the effectiveness of our framework. It is used to select configuration states for containers to move to and works in the Randomization Controller module. More specifically, the strategy's goal is making full use of diversified configuration space to choose more unpredictable configurations to make the attacker's job of compromising the system more difficult. Therefore, we classify multiple configuration combinations for shifting among different kinds of states to accomplish more diversification. We use cluster analysis to do the classification, which is a classic task and common technology in exploratory data mining and statistical data analysis [16]. Moreover, to increase the probability that the configuration state with more difference is chosen, we introduce the weighted random sampling. We calculate the distances among multiple clusters and set these distances as the weights of the sampling associated with each cluster. In this way, we can shift configuration states of containers with both diversification and randomization. More details on this strategy will be described below.

Firstly, the Configuration Generator module generates multiple configuration combinations as the input of the algorithm. Then, we reference the k-means algorithm, one of the most popular methods for cluster analysis in data mining, to cluster these configuration combinations. However, configuration combinations are not normal vectors in the k-means algorithm but consisted of string objects. This means that the commonly used k-means algorithm cannot work well in this model. Hence, we modify the k-means algorithm to fit this model.

The distance: The commonly used k-means algorithm measures the distance between different vectors by the Euclidean distance. But this method is used to calculate the distance between vectors of numerical elements. We use the heterogeneity (H) defined in Eq. (3) as the distance computation method between different configuration combinations to replace the Euclidean distance normally used in the k-means.

The center: The commonly used k-means algorithm defines the cluster's center by the mean of the cluster's vectors. But the configuration combinations are string types that cannot calculate the means. We calculate the mean of H between a configuration combination and other all configuration combinations in the same cluster. And let the configuration combination with the minimum value to denote the center of the cluster.

Based on the above modifications, we design the Clustering Algorithm and show the main part written by pseudocode in Algorithm 1. Moreover, because the initial centers are selected randomly and the number of clusters k is not certain, the clustering result might not be best. We need to iterate the algorithm several times to get the near-optimal result.

Algorithm 1 Clustering Algorithm

Input: k : the number of clusters; max_iter: the maximum value of iterations times; $U = \{C_1, C_2, C_3, ..., C_r\}$: the set of alternative states;
Output: $clusters$: the clustering result; $centers$: the centers of clusters;

```
pre_clusters = { }
num_iter = 0
centers = random.sample(U, k)
while num_iter < max_iter do
    clusters = { }
    for i in U do
        dts = [ ]
        for index, point in centers do
            distance = H(i, point)
            dts.append((index,distance))
        dts.sortbydistance()
        min_index,min_dt = dts[0]
        if min_index not in clusters then
            clusters[min_index] = [ ]
        clusters[min_index].append(i)
    for key in clusters do
        centers[key] = find_center(clusters[key])
    if pre_clusters != clusters then
        pre_clusters == clusters
    else
        break
return centers, clusters
```

After getting the clustering result, we will select the configuration combination to move to for each container. This operation will recur at regular intervals T. To achieve both diversification and randomization, we design the Target Selection Algorithm based on weighted random sampling. We calculate the distances between the configuration combination of the current state and each cluster center of the data set and set these distances as the weights of the sampling associated with each cluster. The sampling probabilities w_i of the cluster q_i is shown in (4), where q_i' denotes the center of q_i, Q denotes the number of cluster centers, and C_c is the configuration combination of current state. Moreover, we generate a random number $\beta \in (0, 1)$, select cluster q_i if the following condition are satisfied.

$$w_i = \frac{H(C_c, q_i')}{\sum_{k=1}^{Q} H(C_c, q_k')} \tag{4}$$

$$\frac{\sum_{h=1}^{i-1} w_h}{\sum_{k=1}^{Q} w_k} < \beta \leqslant \frac{\sum_{h=1}^{i} w_h}{\sum_{k=1}^{Q} w_k} \tag{5}$$

After this, we select a configuration combination randomly from the selected cluster as the target and transmit this configuration combination to the Deployment Handler.

C. Security Analysis

In this subsection, we demonstrate the rationality of the MTD strategies generated by ConfigRand in theory.

1) Feasibility Analysis: According to Sect. III, we analyze the configuration files of non-namespaced subsystems which may lead to information leakages and extract container-oblivious configuration parameters. The parameters we choose are useless (such as the information about non-corresponding CPUs) or even false for a container originally. Meanwhile, we establish exclusive deceptive system configurations for each container and don't change the host OS. Therefore, the changes of parameters will not affect the normal execution of other essential applications in container and OS level.

2) Validity Analysis: In the threat model, the information about the cloud environment is almost unknown for attackers before they explore the system. For penetrating the system, attackers must scan, detect, and compress the exploration surface which means the configurations in this work. ConfigRand defends against the scan and detection by changing these configurations and expanding exploration surface to limit and reduce the exposure of the system vulnerability. We use Z to denote the number of configuration combinations that the configuration space has and z to denote the number of configuration combinations that contain practicable vulnerabilities. The ratio of logical attack surface that ConfigRand shrinks is $0 < 1 - z/Z < 1$.

To demonstrate the validity of configuration shifting, we calculate the success rate of scanning in theory. We assume that there are x containers in the cloud and y of them have vulnerabilities that cloud be penetrated by attackers. Because attackers don't know the details of the cloud, they select i non-repetitive containers randomly to start the exploration. In the original cloud, attackers find j containers which have useful vulnerabilities conforms to Hypergeometric distribution. The probability can be expressed as $P(X = j) = (C_y^j \cdot C_{x-y}^{i-j})/C_x^i$. Therefore, the success rate of attackers is $P(X > 0) = 1 - C_{x-y}^i/C_x^i$. In the cloud with ConfigRand, we shift the configurations of each container independently, so the probability that a container has useful vulnerabilities is z/Z. This condition conforms to Bernoulli distribution. The probability can be expressed as $P'(X = j) = C_y^j \cdot (z/Z)^j \cdot (1 - z/Z)^{i-j}$. Hence, the success rate of attackers is $P'(X > 0) = 1 - (1 - z/Z)^i$. We could get the result that $P'(X > 0)$ is less than $P(X > 0)$ by comparing. This proves that the ConfigRand can effectively reduce the probability of successful exploration of attackers.

V. Evaluation

In this section, we present the details of our experimental method, analyze the influence factor of the experiment, and compare ConfigRand with other methods.

We deploy multiple containers on our testbed to construct a complete container-based cloud. There are some hosts with different software and hardware conditions in the cloud environment and containers are located on these hosts randomly. These containers also belong to different types to achieve the diversity of the experimental environment. The configurations of these components are shown in Table. II. We use NetworkX to build the experimental cloud environment on our testbed. NetworkX is a Python package to create, manipulate, and study the structure, variation, and functions of complex networks [17]. Besides, we use an individual server to run all the modules except the deployment handler of our framework (the deployment handler located on each host which containers run on). It mainly undertakes the computational tasks and system control of the framework.

TABLE II: Configurations in experiments

Type	CPU cores	Memory size	Image type
Host-1	4	64GB	CentOS
Host-2	6	96GB	Ubuntu
Container-1	2	1GB	CentOS
Container-1	2	2GB	Ubuntu
Container-3	1	256MB	Mysql
Container-4	1	512MB	Nginx

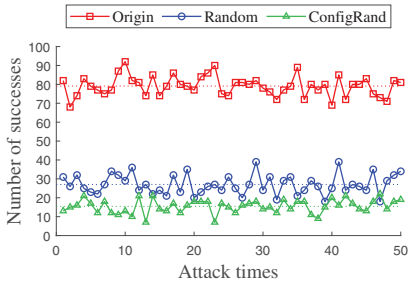

Fig. 4: The number of successful explorations

A. Effectiveness Evaluation

This experiment purposes to observe the number of successful explorations when the attacker tries to find out the vulnerabilities in the cloud environment, and prove the effectiveness of ConfigRand. We design 200 containers in the testbed and each of them includes a CVE vulnerability listed in [18]. In addition, we establish 300 containers without these vulnerabilities in the testbed as the secure applications. Then, we randomly select 200 containers in the testbed to explore these vulnerabilities. The exploration repeats 50 times in three different conditions: the original cloud environment, the cloud environment with pure random MTD strategy (without the Configuration Movement Strategy we propose) deployed, and the cloud environment with ConfigRand deployed.

As shown in Fig. 4, the number of successful explorations is 79.10, 27.04, and 15.44 on average in three different cloud environments. The result shows that ConfigRand framework is effective in defending against vulnerability exploration aimed at the shared kernel system files, and the Configuration Movement Strategy further improves the effect. The success rate is limited to a low percentage and is 80.48% lower than the origin. It doesn't prevent all the explorations, because the deceptive configurations we choose to move to might contain the same vulnerability with the original real configurations coincidently. In this condition, attackers can also find the vulnerability. This problem could be reduced by expanding the configuration space to increase the diversity of alternative configuration combinations in practice.

B. Contrast Experiment

We compare the configuration movement strategy we proposed with the Random strategy to show the validity of ConfigRand in increasing the heterogeneity of shifting. The Random strategy selects configuration combinations from the configuration space randomly for shifting. We execute the shift operation 100 times consecutively in the experiment, observing the variation of configuration combinations.

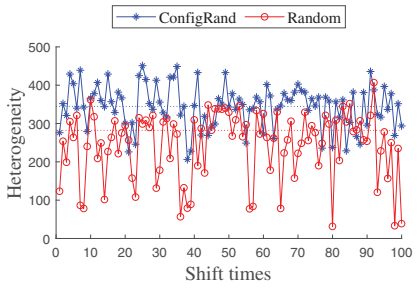

Fig. 5: ConfigRand vs Random

TABLE III: Time costs

Operation	Frequency	Total time	Average time
Configuration generation	500	458.8s	0.917s
Cluster analysis	20	230.7s	11.535s
Configuration selection	100	136.1s	1.361s
Configuration deployment	100	4.3s	0.043s

We compare the heterogeneity between each configuration combination and its previous one in 100 consecutive shifts. As shown in Fig. 5, the average value of heterogeneity in the ConfigRand is higher than the Random, and the ConfigRand always selects the configuration combination with high heterogeneity. This is because the algorithm we present classifies the configuration combinations in advance and would not choose the one which is similar to the previous state. This feature guarantees the effectiveness of ConfigRand and make its performance more robust than the pure random strategy.

C. Time Costs Evaluation

In our system, the time costs come from different operating steps. To demonstrate the usability of ConfigRand in practice, we test and count the time costs after enabling our system. We generate 500 different configuration combinations to compose the configuration space and shift the system configurations (including two operations: selection and deployment) for 100 containers. Table III lists all experimental results. Although the time costs of configuration generation and cluster analysis are obvious, these operations are running on the individual server and will not affect the normal running of the cloud. Besides, these two operations only need to execute one time in the system initialization process, so it's acceptable for a long-running system. The configuration shift operation is triggered periodically, so we pay more attention to its average cost of every single time. The time cost is much lesser than the cycle period of configuration shift, this means that the operation could be completed in advance. As a result, system implementation is practical for the container cloud use case.

VI. Related Work

Different than fixing loopholes of systems to shrink the attack surface, Moving target defense (MTD) is a proactive defense mechanism that aims at extending the exploration surface and shifting the attack surface periodically. MTD can disturb attackers' exploration to the system and make them spend more energy to repeat the exploration. Therefore, the difficulty for attackers to compromise the target

system will increase and the success rate of attacks will reduce [19]. Previous work paid attention to the variation of different aspects of the system to extend the exploration surface or shift the attack surface, including the network configurations [20], instruction set [23], application code [24], or memory address space [21], [22]. These work all moved some surface of the target system, performing the ideology of MTD. More specifically, shifting the configurations (e.g., hardware, OS, configuration files, software, etc.) to establish a dynamic runtime environment is an implementation method of MTD. Some studies focused on the introduction of artificial diversity (e.g., system information, geographic destination, move interval, etc.) and established metrics to determine the benefit of the defensive technique [25]. But they executed the shifting operation based on a customized move which is easy to be found through analysis. [26], [27] modeled computer configurations as chromosomes and used an evolutionary algorithm to discover better computer configurations. This method focused on the security of new configurations, but don't have enough diversification for shifting. Other studies centres around some familiar aspects of system configuration, for example IP shift [20], [28], memory address relayout [21], instruction set randomization [29], html keywords change [30], etc [10], [31]. However, they do not apply to the information leakages in container-based clouds.

VII. Conclusion

In this paper, we discuss the root causes of the shared kernel information leakages and related attacks in the container-based cloud. To solve these problems, we present a MTD framework called ConfigRand that uses system configuration shift. We establish the mapping between the shared system configurations and deceptive system configurations for each container, shift these configurations, and change the attack surface of the target system to reduce the probability of successful vulnerability exploration. Meanwhile, ConfigRand does not require changes to applications, containers, or the Linux kernel besides the file map of the container engine, which brings little performance costs.

Acknowledgment

This work was supported by the National Key Research and Development Program of China (Grant No.2017YFB1010004).

References

[1] S. Soltesz, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization:a scalable, high-performance alternative to hypervisors," Acm Sigops/eurosys European Conference on Computer Systems. 2007.
[2] Kubernetes. http://kubernetes.io, 2020
[3] Ali ECI. https://www.aliyun.com/product/eci, 2020
[4] Amazon ECS. https://aws.amazon.com/cn/containers, 2020
[5] Adya, A., Bolosky, W. J., Castro, M., Cermak, G., Chaiken, and W, R, "Farsite: federated, available, and reliable storage for an incompletely trusted environment," ACM Sigops Operating Systems Review, 2002.
[6] B. Singh and V. Srinivasan, "Containers: Challenges with the memory resource controller and its performance," Routledge Part of the Taylor & Francis Group, 2007.
[7] Capability. https://linux.die.net/man/7/capabilities, 2020
[8] J. Thones, "Microservices," IEEE Software, vol. 32, pp. 116-116, 2015.
[9] X. Gao, Z. Gu, M. Kayaalp, D. Pendarakis and H. Wang, "ContainerLeaks: Emerging Security Threats of Information Leakages in Container Clouds," 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 237-248, 2017.
[10] R. Zhuang, S. A. DeLoach, and X. Ou, "Towards a theory of moving target defense,," In Proceedings of the First ACM Workshop on Moving Target Defense, 2014.
[11] LXC. https://linuxcontainers.org, 2020
[12] Docker. https://www.docker.com, 2020
[13] H. Alavizadeh, D. S. Kim, J. B. Hong, and J. J., "Effective Security Analysis for Combinations of MTD Techniques on Cloud Computing," International Conference on Information Security Practice & Experience, 2017.
[14] ., Li, and L. Bo, "A Normalized Levenshtein Distance Metric," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 29, pp. 1091-1095, 2007.
[15] Develop your own filesystem with FUSE. https://developer.ibm.com/technologies/linux/articles/l-fuse, 2020
[16] Everitt, Brian, "Cluster analysis," Quality and Quantity, vol. 14, pp. 75-100, 1980.
[17] H. Aric, S. Daniel, and S. Pieter, "Exploring network structure, dynamics, and function using networkx," Proceedings of the 7th Python in Science Conference, 2008.
[18] Docker Vulnerability Statistics. https://www.cvedetails.com/vendor/13534/Docker.html, 2020
[19] H. Jin, Z. Li, D. Zou and B. Yuan, "DSEOM: A Framework for Dynamic Security Evaluation and Optimization of MTD in Container-based Cloud," IEEE Transactions on Dependable and Secure Computing, 2019.
[20] M. Dunlop, S. Groat, W. Urbanski, R. Marchany, and J. Tront, "MT6D: a moving target ipv6 defense," Military Communications Conference, pp. 1321-1326, 2011.
[21] C. Kil, J. Jun, C. Bookholt, J. Xu, and P. Ning, "Address space layout permutation (ASLP): Towards fine-grained randomization of commodity software," 22nd IEEE Annual Computer Security Applications Conference, pp. 339-348, 2006.
[22] H. Shacham, M. Page, B. Pfaff, E. J. Goh, N. Modadugu, and D. Boneh, "On the effectiveness of address-space randomization," Proceedings of the 11th ACM conference on Computer and communications security, pp. 298-307, 2004.
[23] S. W. Boyd, G. S. Kc, M. E. Locasto, A. D. Keromytis, and V. Prevelakis, "On the general applicability of instruction-set randomization," Dependable and Secure Computing, vil. 7, pp. 255-270, 2010.
[24] Y. Huang and A. K. Ghosh, "Introducing diversity and uncertainty to create moving attack surfaces for web services," Moving Target Defense, pp. 131-151, 2011.
[25] P. Beraud, A. Cruz, S. Hassell, J. Sandoval, and J. J. Wiley, "Cyber defense Network Maneuver Commander," IEEE International Carnahan Conference on Security Technology, 2010.
[26] B. Lucas, E. W. Fulp, and Da. J. John, "An initial framework for evolving computer configurations as a moving target defense," ACM Cyber & Information Security Research Conference, 2014.
[27] Da. J. John, R. W. Smith, W. H. Turkett, and E. W. Fulp, "Evolutionary based moving target cyber defense," Companion Publication of the Conference on Genetic & Evolutionary Computation, 2014.
[28] S. Groat, M. Dunlop, R. Marchany, and J. Tront, "Using dynamic addressing for a moving target defense," the 6th International Conference on Information Warfare and Security, pp. 84-91, 2011.
[29] G. S. Kc, A. D. Keromytis, and V. Prevelakis, "Countering code-injection attacks with instruction-set randomization," the 10th ACM conference on Computer and communications security, pp. 272-280, 2003.
[30] S. Vikram, C. Yang, and G. Gu, "Nomad: Towards non-intrusive moving-target defense against web bots," IEEE Conference on Communications and Network Security, pp. 55-63, 2013.
[31] T. Kong, L. Wang, D. Ma, Z. Xu, Q. Yang and K. Chen, "A Secure Container Deployment Strategy by Genetic Algorithm to Defend against Co-Resident Attacks in Cloud Computing," IEEE 21st International Conference on High Performance Computing and Communications, pp. 1825-1832, 2019.