MTD CBITS: Moving Target Defense for Cloud-Based IT Systems

Alexandru G. Bardas[1]*, Sathya Chandran Sundaramurthy[2]**,
Xinming Ou[3], and Scott A. DeLoach[4]

[1] University of Kansas, Lawrence KS, USA
[2] DataVisor, Mountain View CA, USA
[3] University of South Florida, Tampa FL, USA
[4] Kansas State University, Manhattan KS, USA

alexbardas@ku.edu, sathya.chandran@datavisor.com,
xou@usf.edu, sdeloach@ksu.edu

Abstract. The static nature of current IT systems gives attackers the extremely valuable advantage of time, as adversaries can take their time and plan attacks at their leisure. Although cloud infrastructures have increased the automation options for managing IT systems, the introduction of Moving Target Defense (MTD) techniques at the entire IT system level is still very challenging. The core idea of MTD is to make a system change proactively as a means to eliminating the asymmetric advantage the attacker has on time. However, due to the number and complexity of dependencies between IT system components, it is not trivial to introduce proactive changes without breaking the system or severely impacting its performance.

In this paper, we present an MTD platform for Cloud-Based IT Systems (MTD CBITS), evaluate its practicality, and perform a detailed analysis of its security benefits. To the best of our knowledge MTD CBITS is the first MTD platform that leverages the advantages of a cloud-automation framework (ANCOR) that captures an IT system's setup parameters and dependencies using a high-level abstraction. This allows our platform to make automated changes to the IT system, in particular, to replace running components of the system with fresh new instances. To evaluate MTD CBITS' practicality, we present a series of experiments that show negligible (statistically non-significant) performance impacts. To evaluate effectiveness, we analyze the costs and security benefits of MTD CBITS using a practical *attack window* model and show how a system managed using MTD CBITS will increase attack difficulty.

1 Introduction

Current IT systems operate in a relatively static configuration and give attackers the extremely important advantage of time. Therefore, a promising new approach, called *Moving Target Defense* or MTD [19], has emerged as a potential

* Corresponding author. As of July 2017, Alexandru G. Bardas's affiliation is The University of Kansas. This work was conducted when he was a graduate student and then a visiting assistant professor at Kansas State University.

** As of June 2017, Sathya C. Sundaramurthy's affiliation is DataVisor. This work was conducted when he was a graduate student at University of South Florida.

solution. MTD techniques are expected to increase uncertainty and complexity for attackers, reduce their window of opportunity, and raise the costs of their reconnaissance and attack efforts. There have been a number of MTD-related research efforts such as randomizing memory layouts [3, 13, 31], IP addresses [6, 20, 27], executable codes [8, 24, 53], and even machine instruction sets [9, 29]. These are important steps towards achieving the overall goal of moving target defense, but they focus on individual aspects of a system — IP addresses, code for particular applications, and specific architectures. There has not been much research on how to apply an MTD approach at the entire IT system level. We view an *IT system* as a subset of an enterprise network, a group of one or more machines (physical or virtual) that work together to fulfill a goal. The overall goal and the scope of an IT system are determined by the user (system engineer/administrator) and can range from a one-machine service (e.g., FTP server), to more complex deployments with large numbers of machines with internal dependencies (e.g., multi-host eCommerce setups).

Applying an MTD approach to the entire IT system is important for several reasons. First, system administrators fight the continual and generally losing battle of monitoring their IT systems for possible intrusions, patching vulnerabilities, modifying firewall rules, etc. The complexity of such systems and the time required to maintain them are major reasons why errors creep into system configurations and create security holes. The stagnant nature of IT systems gives adversaries chances to discover security holes, find opportunities to exploit them, gain/escalate privileges, and maintain persistent presence over time. For example, the data released (summer 2016) as a consequence of the Democratic National Committee (DNC [18]) breach resulted after attackers were present in the DNC systems for over a year [15]. According to Mandiant's M-Trends 2016 and 2017 reports [34, 35], the median number of days an organization was compromised before discovering the breach was 146 days in 2015 and 99 days in 2016. Even though this constitutes an improvement, it is still way too long. For instance, Mandiant's Red Team was able to obtain access to domain administrator credentials, on average, within three days of gaining initial access to an environment. On the other hand, Verizon's DBIR 2016 [49] states that, overall, the detection deficit is actually getting worse.

Persistence is a trend that turned into a constant [34, 35]. Introducing changes at the entire IT system level will increase the difficulty for attackers to obtain initial access and, especially, to maintain persistent presence. Persistent malware is given an expiration date as running components of the IT system are constantly being replaced with fresh new instances. This has the potential to change the current attacker mode of operation from *compromise and persist* [15, 33, 34, 35] to the more challenging obligation of *repeated compromise*.

However, there are several challenges for introducing MTD mechanisms at the entire IT system level. Due to the number and complexity of dependencies between IT system components, it is not trivial to carry out proactive changes without breaking the system or severely impacting its performance. Introducing changes proactively, if done improperly, may introduce additional complexities.

Making a complex system more complex is unlikely to increase its security. Thus a practical MTD design must simplify system configuration and maintenance, while enabling the capability of "moving". For this reason, we have leveraged ANCOR [47] proposed in our prior work and extended it to an MTD platform.

ANCOR is a framework for creating and managing cloud-based IT systems using a high-level abstraction (an up-to-date IT system inventory). While ANCOR was focused on creating and managing IT systems in a reliable and automated way, this paper analyzes the feasibility and potential security benefits of an MTD approach based on live instance replacement. A live instance replacement mechanism can be the means to deploying various defenses in an automated way while constantly removing attackers' persistent access. For verification purposes, we have re-created the eCommerce scenario, tested it in a new performance testing setup, and also developed a new scenario that uses a set of operational database dumps and real traffic traces (MediaWiki [36] with Wikipedia database dumps). The main contributions of this paper are as follows:

1. We leverage ANCOR [47] for creating and managing IT systems, and extend it to an MTD platform based on live instance (VM) replacements.
2. We evaluate the practicality of this MTD platform through a series of experiments on two realistic IT system scenarios. The experimental results show that the MTD operations may have negligible impact on the normal operations of the IT systems.
3. We analyze the security benefits brought by the MTD platform through an attack window model, and show how to use the model to quantify the security benefits of a given MTD configuration.

2 Our MTD Approach

Our approach of introducing moving target defense at the entire IT system level is to create a platform where any running component of an IT system can be replaced with a pristine version. A component is simply a virtual machine instance or a cluster of instances. We consider that the MTD approach will be deployed in a cloud environment. Cloud infrastructures (e.g., OpenStack and Amazon Web Services – AWS) made it possible and easy to create bare-metal equivalent virtual machine instances and networks. It appears inevitable that IT systems of all sizes are moving towards the cloud — be it private, public, or hybrid (fog and edge computing).

2.1 Threat Model

In-scope threats are the risks our MTD approach intends to mitigate, by increasing the difficulty on the attackers' side. The risks range from reconnaissance actions to arbitrary code execution, and side-channel attacks.

Attackers are able to perform various reconnaissance actions (e.g., port scanning) on the public facing instances, as well as internal probing if they gain access to an instance on the internal network. Furthermore, they may also execute arbitrary code on an instance. Applications may be poorly configured, misconfigured, or have vulnerabilities that allow arbitrary code execution with administrator/root privileges on an instance which is part of the targeted system, e.g., buffer overflow, unsanitized input. Moreover, a social engineering attack (e.g., phishing) may lead to obtaining the privileged user credentials. Arbitrary code execution can result in an operating system compromise that enables attackers to escalate their privileges and maintain their access through backdoors. In addition, attackers may attempt to pivot through the internal network.

Attacks on the MTD platform itself are out of scope for this paper; this includes the MTD controller, the cloud platform (usually controlled by the cloud provider), and the configuration management tools. Currently, the MTD controller instance is protected using guidelines (e.g., [46]) for securing configuration management tool master nodes. We leave it for future work to study in-depth the security of the MTD platform itself.

We evaluated the feasibility of replacing services and small databases. Since persistent data is stored on different volume types in a cloud (e.g., OpenStack Cinder, Ceph, etc.), attaching the data volume to new instances proved more efficient than synchronizing the data on each new instance.

Attackers might be able to store backdoor information in persistent data that enables them to restore persistent access, making the replacement process less effective. Various approaches have been proposed for different environments to ensure the integrity of the stored data, e.g., [17], [48], [28]. For the purpose of this paper we are relying on existing solutions for ensuring data integrity.

2.2 Background

The advancements in virtualization technologies contributed significantly to the evolution of cloud computing [7]. The following capabilities are commonly available on a cloud platform: provisioning instances with various hardware capabilities, utilizing security groups for network access control, and creating storage volumes. At the same time, configuration management tools (CMTs) have become a well-established solution to managing the applications and services (software stack) in an automated fashion. Popular CMT solutions include Puppet [43] and Chef [12]. Walmart, Wells Fargo, and other companies leverage CMTs to configure tens of thousands of servers in an automated fashion [44].

A CMT works by installing an agent on the host to be managed, which communicates with a controller (called the master) to receive configuration directives. In case the host's current state (e.g., installed packages, customized configuration files, etc.) is different than the one specified in the directives, the CMT agent is responsible for issuing the appropriate commands to bring the system into the specified state.

2.3 MTD CBITS Implementation

MTD CBITS (Figure 1) is based on the ANCOR framework which supports creating and managing cloud-based IT systems using a high-level abstraction. The abstraction allows the system administrator to define the high-level structure of the IT system, without specifying the detailed configuration parameters such as IP addresses, port numbers, and other application-specific settings for each instance. The high-level abstraction explicitly specifies the dependency among the various *roles* — clusters of instances with similar configurations. ANCOR has a “compilation process” that processes this abstract specification, generates detailed configuration parameters for each instance, leverages CMT role implementations, and automatically creates an IT system on a cloud infrastructure. The current implementation targets OpenStack and uses Puppet; it may also be changed to AWS and Chef.

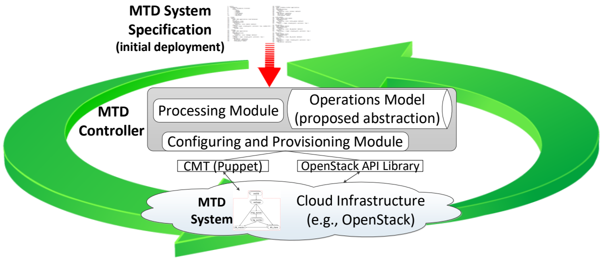

Fig. 1: The MTD platform (MTD CBITS) takes an abstract specification of an IT system as its input, and creates the corresponding concrete system on a cloud infrastructure. In addition to ANCOR, MTD CBITS can perform frequent **live instance replacements** throughout the lifetime of the IT system (green arrows).

In this paper we refer to an *MTD system* as an IT system deployed and managed using our MTD platform that supports dynamically replacing instances. The platform takes an *MTD system specification* (user’s requirements) as its input and automatically creates and manages the corresponding concrete *MTD system* on OpenStack (Figure 1). The configuration parameters are **not** hard-coded; they are generated at run-time from the high-level system specification. The operations model stores the computed parameters and can be viewed as an MTD system inventory — a layer on top of the CMT (Puppet). This data is passed to Puppet through Hiera [45], a key/value look-up tool for configuration data. Whenever a change occurs in the deployed MTD system, it is also recorded in the operations model. Therefore, the operations model always stores up-to-date information about the running IT system.

Most of the MTD CBITS components are stored on the *MTD controller* (see Figure 1). The MTD controller is, basically, used to deploy and manage the MTD systems: it can reach the OpenStack API, hosts the Puppet master, and is able to communicate through the Puppet agents with all instances that are part of the IT system. The MTD controller cannot be reached from the public network and communicates with the agents over an internal isolated network. Moreover, the communication between the Puppet master and the agents is encrypted.

2.4 Instance Replacement Implementation

Using the operations model, MTD CBITS facilitates a variety of *adaptation* operations (movements) for the managed IT systems, creating a moving target defense. In our MTD approach, live instance replacement is carried out through a sequence of adaptations: adding new instances, reconfiguring dependent instances, and removing the old instances.

Reconfiguring Instances. In-place reconfigurations (updated CMT directives) may include internal service changes such as changing service parameters (e.g., credentials), applying service and OS patches, etc., or changes that involve dependent roles. These changes will be accompanied by infrastructure updates (e.g., security group changes).

Adding or Removing Instances. The MTD platform enables the addition and removal of running instances. Both adaptations also involve reconfiguring dependent instances. This happens through a sequence of tasks and in both cases, the affected dependent services will be notified using a set of updated CMT directives. When adding a new instance, the updated configuration directions are sent to the dependent instances (push configuration to dependent instances) after the new instance is ready-to-use (provisioned and configured). In this way, if failures affect the new instances the MTD system's functionality will not be affected during the change process. On the other hand, when removing an instance, first, the dependent instances are notified before the actual deletion happens.

The *instance replacement* process merges the adding of new instances and removing the old instances: one-instance or a cluster of instances may be replaced at once. Creating security groups, provisioning new instances, and configuring them are tasks that can be performed in parallel. Once all these tasks finish, the MTD controller computes the updated CMT directives for all the dependent instances. *Dependent instances will receive only one set of directives that contains all the updates.* Therefore, replacing one instance, or replacing all instances belonging to a role, will take roughly the same amount of time. The new instances may use compatible implementations with different IPs, ports, operating systems or application versions. The roles that instances fulfill in an MTD system can be implemented in numerous ways.

3 Feasibility Analysis

This section summarizes our conclusions after evaluating the impact of instance replacements on real-world IT systems deployed and managed using our MTD CBITS platform. Regardless of potential security benefits, an unreasonable performance overhead would make the approach infeasible. We focused our efforts on the applications, while persistent data (database content) was stored on cloud volumes and reattached to new instances.

Our **hypothesis** was that the performance overhead of instance replacements can be negligible (statistically non-significant) when using MTD CBITS. The experiments were carried out on a cloud testbed consisting of 14 nodes (1 controller and 13 compute nodes) running OpenStack (Icehouse). We focused on two IT system setups: eCommerce deployment and MediaWiki with Wikipedia database dumps. More scenarios are available on our project's webpage.

To test the performance, we used http-perf [2] for the eCommerce system and WikiBench [51] for the MediaWiki deployment. http-perf launches HTTP requests against a server while capturing several metrics, including response times while WikiBench replays real traffic traces against a MediaWiki site. To establish a baseline (i.e., the **control group**), we ran the benchmarking tools without MTD enabled (no instance replacements). Next, we ran the benchmarking tools while replacing various instances. During the replacement process, saving and restoring the active sessions was handled at the application level (e.g., `eCommerce_webapp`) or by a dedicated component in the system (i.e., `memcached` in the MediaWiki/Wikipedia scenario). We observed that depending on the component that is being replaced all or the vast majority of the active sessions were successfully restored. In all setups, caching features were disabled and configurations were reloaded without restarting the services. For this reason, we did not focus on the performance measurement values per se but on the difference (Δ) between the baseline and the replacement measurements. With caching enabled, requests are answered from the cache and not from the system component under test (e.g., webapp) [47]. Thus, there is little or no impact of component replacement. *Using MTD CBITS to manage the above-mentioned scenarios, we were able to show that our hypothesis holds.*

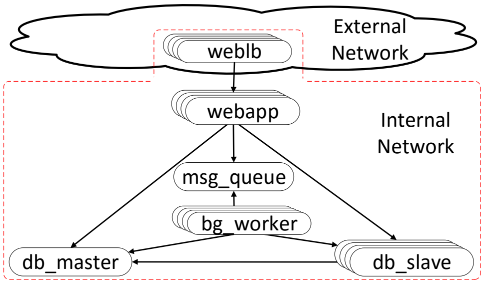

Fig. 2: Scalable and highly available eCommerce website blueprint. `db_master, msg_queue` are single instances while `weblb, webapp, bg_worker, db_slave` are implemented by a homogeneous cluster of instances.

3.1 eCommerce Deployment

Let us consider a scalable and highly available architecture of an eCommerce website with various clusters of services as shown in Figure 2: web load balancers (Nginx or Varnish), web application (Ruby on Rails with Unicorn), database (MySQL), messaging queue (Redis), and worker app (Sidekiq). A cluster can be implemented by one or more homogeneous instances. Arrows indicate dependency between clusters of instances. Each cluster consists of multiple instances implementing the same services.

The website implements the basic operations (i.e., read and write from and to the database, or submit a worker task) needed in an eCommerce setup. The baseline performance (Table 1) was determined by performing read operations on the eCommerce website. Similar to Unruh et al. (our previous work), we focused our efforts on the web application and database clusters, but tested them using a different benchmarking tool (http-perf) and an increased load on the database.

As it can be observed in Table 1, under baseline conditions the eCommerce deployment was able to handle 150,000 requests originating from 70 connections without any errors. Each request was reading 50 entries from the database. Replacing database or web application instances can be performed in a comparable

Aggregated results from **20** experiment runs
Each experiment run: **150,000** requests sent using **70** concurrent connections

	Response time (sec)		Total time		Server Processing Rate (req/sec)		HTTP Error Responses	
	Avg.	stdev	Avg.	stdev	Avg.	stdev	Avg.	stdev
Baseline	0.408	0.069	14min 48sec	160 sec	175.352	36.924	0	0
Replacing one webapp	0.425	0.050	15min 17sec	119 sec	166.340	22.236	1.50	4.66
Replacing webapp cluster	0.424	0.047	15min 16sec	110 sec	166.032	18.887	42.60	37.57
Replacing one db_slave	0.426	0.040	15min 31sec	91 sec	162.675	16.481	588.10	62.84
Replacing db_slave cluster	0.439	0.035	15min 55sec	73 sec	158.051	12.320	913.75	113.57

Table 1: eCommerce website – average performance overhead of carrying out **one** replacement operation: replacing one instance and replacing the whole cluster.

amount of time (within approximately a minute of the baseline measurements). Next, we tried to assess the overall impact of the instance replacement process under the same high load used in the baseline. We performed one-instance and whole-cluster instance replacement on the web application cluster, and then on the database cluster (specifically database slaves). The differences between replacement and the baseline measurements are, in general, statistically non-significant and the performance loss is insignificant during the replacement process (see Table 1). When replacing the webapp, there were very few HTTP error responses. On the other hand, when replacing the database slaves, as shown in Table 1, the performance is slightly impacted by this change and on average 913.75 out of 150,000 requests failed, amounting to 0.61% of total number of requests. We observed that requests are dropped when dependent instances are establishing connections with the new (fresh) instances due to the received configuration updates, while still processing incoming requests.

3.2 MediaWiki with Wikipedia DB Dumps

Unlike the eCommerce scenario that utilized synthetic workloads, WikiBench is a web hosting benchmark that leverages actual Wikipedia database dumps and generates real traffic by replaying traces of traffic addressed to wikipedia.org.

Similar to Moon et al. [37] we utilized the traces from September 2007 and the corresponding Wikipedia database dumps [52]. Our setup consists of a load-balancer (Nginx), three MediaWiki backends, a database hosting the Wikipedia dumps, and a Memcached instance for sharing sessions (the state) among the backends (Figure 3).

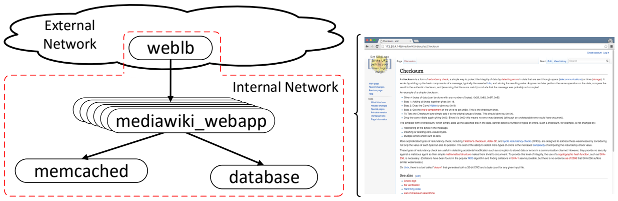

Fig. 3: MediaWiki with Wikipedia database dumps.

Aggregated results from **10** experiment runs
Each run: around **4150** requests, **50** threats, **1** worker, max. timeout **200** ms

	Response time (sec)		Total time		Server Processing Rate (req/sec)		HTTP Error Responses	
	Avg.	stdev	Avg.	stdev	Avg.	stdev	Avg. Diff.	stdev
Baseline	0.054	0.001	10min 1sec	0.0003 sec	6.914	0.004	N/A	1.26
Replacing one webapp	0.053	0.001	10min 1sec	0.001 sec	6.910	0.006	0	1.12
Replacing webapp cluster	0.053	0.001	10min 1sec	0.001 sec	6.910	0.005	+3	1.77

Table 2: WikiBench (WikiBench (MediaWiki with Wikipedia database dumps) – average performance overhead of carrying out **one** replacement operation: replacing one `mediawiki_webapp` instance and replacing the whole `mediawiki_webapp` cluster.

In establishing the baseline, we ran WikiBench (replayed real traces) on our deployment. MTD CBITS did not interfere in an any way when performing the baseline measurements. Next, we replayed the same traces while replacing one `mediawiki_webapp` instance and then the whole cluster. We recorded the averages and standard deviations over ten different runs (see Table 2). We did not focus on the overall errors per se, however, we directed our attention on the difference in the number of errors between the baseline and the replacement actions. We noticed that the difference between the replacement operations averages and the baseline is very small, statistically non-significant. However, in case of the one-instance replacement, we recorded an outlier that displayed a much lower number of HTTP 200 responses than the rest of the experiment runs: 608 compared to 855, which was the average over nine experiment runs. Including the outlier we would still have only 27 errors with a stdev of 90.55 errors.

4 Security Analysis

In general, quantifying the security of an IT system is a challenging task [26]. Quantifying the benefits of constantly changing a system is even more demanding [23]. While there have been numerous attempts [16, 22, 26, 41], the proposed security metrics are usually at a higher abstraction level that enables them to capture a wider range of IT systems. Thus, most of the time, it is hard to validate them in an objective manner on a concrete (production-like) IT system.

We propose to measure the effectiveness of an MTD system in terms of the meaningful interruptions it creates for attackers and the cost associated with those interruptions. In a nutshell, this section is focused on determining *when* instance replacements should happen (**strategy**), *how many* replacements in a given time period (**cost**) and *what* this means in terms of **attack windows**, **persistence**, and **pivoting options**.

4.1 Attack Windows and Attack Surface

An *attack window* is a continuous time interval an attacker may leverage without being interrupted by system changes. System changes refer to reconfigurations that would not happen on a regular basis (every few minutes, hours, or days) in a static system, e.g., changing internal IPs, ports, applications, or credentials.

A system's attack surface can be viewed as the subset of the IT system's resources that an attacker can use to attack the system. This subset of resources is composed of methods, channels, and untrusted data items [32]. Methods refer to the codebase entry and exit points of the IT system's software applications, channels are used to connect and invoke a system's methods, while untrusted data items are used to send or receive data into or from the target system. Strategies to harden the system and reduce the attack surface include reducing the amount of running code (methods), eliminating unneeded services, running updated applications, and reducing the channels available to untrusted users [32].

Reconnaissance and Pivoting Options. MTD CBITS manages an IT system's internal communication channels by leveraging OpenStack's security groups as a per-instance fine-grained firewall. A security group is automatically configured to allow only ingress and egress traffic from and to the dependee and dependent instances. Moreover, traffic will be allowed only to and from the ports (TCP and/or UDP) stored in MTD CBITS's operations model (including related connections). Specifically, MTD CBITS reduces the attack surface of the deployment through *reducing the entry points available to untrusted users* and *limiting the number of channels to the predetermined ones.* Instances can initiate connections to dependent instances only on specific port numbers (stored in the operations model, Figure 1).

The limited pivoting options constitute an important security benefit if an attacker is able to compromise one or more instances in the deployment. For example in the eCommerce deployment (Figure 2), if the `weblb` instances were compromised, an attacker would be able to reach only the three `webapp` instances through the internal network and not all the instances belonging to the other nodes. A node represents a role in the IT system – a single unit of configuration that corresponds to one instance or a high-availability cluster of instances. (Here, a *role* as presented in Section 2.3 corresponds to a node in the security analysis.) Without the possibility of creating new communication channels, attackers are forced into using existing channels in order to advance or to exfiltrate data (specifically, only over related connections).

Attacker's Presence – Persistent Access. Attackers usually exploit somewhat unpredictable occurrences on the targeted IT systems e.g., software bugs, misconfigurations, or user actions. Exploits and other actions may not have the same outcome every time they are executed. Although reducing the attack surface in a non-MTD-CBITS environment helps to prevent security failures, it does not mitigate the amount of damage an attacker could inflict once a vulnerability is found. In an MTD CBITS environment, even if the same flawed node/role implementation (with the same vulnerabilities) is used on a new instance, configuration parameters (e.g., IP, ports, credentials, cryptographic keys) will be updated forcing attackers to adjust their attack in order to potentially re-compromise the instance. Installed malware is not really "persistent" anymore and needs to be re-installed on new instances. This process can be noisy since it needs to be performed repeatedly in order to maintain access.

Attack Window Terminology. We have defined the following terminology to describe the proposed model. An *attack attempt* is an effort to gain unauthorized privileges and data on a system. An attack path may include several nodes that are part of the targeted IT system. These nodes can be:

1. *Transparent nodes.* Replacing the instances of such a node will most probably not influence an ongoing attack. Load balancers (`weblbs`) are transparent nodes if they simply relay requests to `webapp` instances without altering them regardless of the `weblb` implementation (e.g., Varnish or Nginx). Replacing a transparent node on the attack path will **not** influence an ongoing attack, e.g., replacing a load balancer should have the same effect on all requests (benign or malicious) to be passed to the `webapps` in the eCommerce website (Figures 2). We note that under different attack assumptions, `weblb` could be attacked directly and in this case it will not be a transparent node.

2. *Stepping-stone nodes.* Different outcomes for benign and malicious requests. For example, in the eCommerce website (Figure 2), an attack on `db_master` to possibly succeed, usually, requires a vulnerable or misconfigured `webapp`. Changing `webapp` to a different implementation will most likely disrupt the ongoing attack on `db_master`. Thus replacing a stepping-stone node on the attack path will impact an ongoing attack. There are two types of stepping-stone nodes:
 a) *Compromised.* Attackers have root/admin privileges.
 b) *Misconfigured.* Attackers don't have complete control over the node. One or more vulnerabilities and misconfigurations allow attackers to perform an attack on a node down the way, e.g., a misconfiguration on the `webapp` instances allows unsanitized user input that results in a SQL injection which leads to compromising the database node, `db_master` (see Figure 2).

An *adaptation point* is the moment when new (fresh) instances start being used in the deployment. New instances use a compatible implementation with different IP addresses, passwords, and port numbers. Due to these configuration changes, attacks are generally interrupted at adaptation points of stepping-stone or target nodes and the attacker must restart the attack attempt.
A few definitions are needed to determine the length of attack windows.

Definition 1 *We define $T_p(X)$ to be the period of time taken into consideration i.e., extent of time when attacks might be launched against node X.*

Definition 2 *$T_r(X)$ is the interval between adaptation points on node X.*
We have $T_r(X) = ch(X) + d(X) + a(X)$, where

$ch(X)$ - time interval to bring a new instance that implements X in a *ready-to-use* state, e.g., provision and configure the new instance(s);

$d(X)$ - duration to change to the ready-to-use new instance(s), $d(X) > 0$ e.g., pushing configuration to dependent nodes; and

$a(X)$ - delay specifically introduced by the user, $a(X) \geq 0$.

Definition 3 *$T_a(X)$ is the duration of an attack attempt on node X.*

Provisioning and configuring new instances can be performed in parallel by MTD CBITS. However, changing to the new instances belonging to dependent nodes (parameter d for each node) must be completed sequentially in order not to disrupt the communication between the dependent services. Therefore, the adaptation points (T_r's) of two dependent nodes cannot be fully aligned (coincide) as such. There will always be a very short delay between the two adaptation points. However, because the duration of d was usually around 1 second in our testing scenarios, we consider this type of alignment as efficient as a full alignment.

One adaptation point does not necessarily create one meaningful interruption for an attacker. If there are several adaptation points that are aligned, we consider this as only one meaningful interruption from an attacker's perspective. A *meaningful interruption* is a disruption that forces attackers to restart an attack attempt (redo a significant number of the steps that are part of the attack attempt). We consider that one adaptation point creates a meaningful interruption if it is at least one time measurement unit away (1 minute in our case) from other adaptation points. Also, we view an *adaptation moment* as one adaptation point or several aligned adaptation points that create a meaningful interruption.

4.2 Adaptation Points Placement

Assuming X is the targeted node and Y_1 ... Y_{l-1} are the stepping-stone nodes on the path to X, our goal is to determine the lengths of potential attack windows. For this reason, it is vital to determine the moments when adaptation points are aligned. First, the individual replacement-process starting time for each node must be taken into consideration. Thus, the earliest starting time can be considered moment 0, while the placement of the other starting times captures the difference related to moment 0. Let us state the following:

$t_{min} = \texttt{min}(start\_time_{T_r(X)}, start\_time_{T_r(Y_1)}, ...)$,
while $t_X = start\_time_{T_r(X)} - t_{min}$, $t_{Y_1} = start\_time_{T_r(Y_1)} - t_{min}$, ... [5]

Now, the problem can be defined and solved using the *Chinese Remainder Theorem.* Using this theorem one can determine integer m that, when divided by some given divisors, leaves given remainders. In our scenario the given divisors are $T_r(X)$, $T_r(Y_1)$ ...$T_r(Y_{l-1})$, the given remainders are t_X, t_{Y_1}, ..., $t_{Y_{l-1}}$, and m represents the moment when the adaptation points are aligned. We can derive the following cases:

Case 1
`If` $T_r(X)$, $T_r(Y_1), ..., T_r(Y_{l-1})$ are pairwise coprime `then`:

- Integer m exists and can be calculated
- All solutions for m are congruent $\texttt{lcm}(T_r(X), T_r(Y_1), ..., T_r(Y_{l-1}))$ [6]

Case 2
`If` $T_r(X)$,$T_r(Y_1), ..., T_r(Y_{l-1})$ **not** pairwise coprime `then`:

```
If $\forall i, j \in \{X, Y_1, ..., Y_{l-1}\}$, $t_i \equiv t_j \mod \texttt{gcd}(T_r(i), T(j))$ is TRUE, then:
    • Integer $m$ exists and can be calculated
Else:
    • Integer $m$ does not exist
```

[5] `min` is the minimum

[6] `lcm` stands for "least common multiple" and `gcd` is the "greatest common divisor"

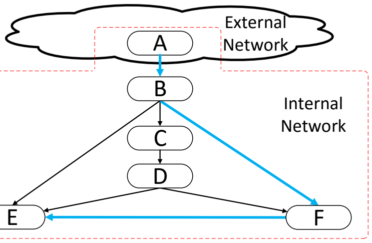

Fig. 4: Possible IT system architecture. Arrows indicate dependencies between nodes.

Case 3

If $T_r(X), T_r(Y_1), ..., T_r(Y_{l-1})$ are **not** pairwise coprime
AND $\forall i, j \in \{X, Y_1, ..., Y_{l-1}\}$, $t_i \equiv t_j \mod \texttt{gcd}(T_r(i), T(j))$ is FALSE, then:

- No pair of adaptation points will be aligned
- Integer m does **not** exist

Case 4

If $T_r(X), T_r(Y_1), ..., T_r(Y_{l-1})$ **not** pairwise coprime AND $\exists i, j, a, b \in \{X, Y_1, ..., Y_{l-1}\}$,
$t_i \equiv t_j \mod \texttt{gcd}(T_r(i), T(j))$ is FALSE,
$t_a \equiv t_b \mod \texttt{gcd}(T_r(a), T(b))$ is TRUE, then:

- Some of the adaptation points will be aligned
- Integer m does **not** exist

4.3 Attack Windows Example

To briefly illustrate the options a user has when managing their deployment using MTD CBITS, let us consider a possible IT system architecture as pictured in Figure 4. Replacing one or all instances belonging to a node takes roughly the same amount of time (see Section 2.4). The architecture pictured in Figure 4 can serve as a concrete eCommerce website (as shown Figure 2).

Based on an improved version (with faster replacements) of the concrete eCommerce scenario, the replacement times for the nodes in Figure 4 are $T_r(B) = 10$ minutes, $T_r(F) = T_r(E) = 11$ minutes, $T_r(A) = T_r(C) = T_r(D) = 3$ minutes and $d(B) = d(F) = d(E) = d(A) = d(C) = d(D) = 1$ second. T_r values are at their lowest bound for the current environment. In other words, ch's and d's are at their minimum and a's are equal to 0.

There are two possibilities to reach node E: A, B, F, E or A, B, E (Fig. 4). For the purpose of this example we will focus on the first path, A, B, F, E. Node A is transparent (e.g., `weblb` in the eCommerce scenario), and therefore $T_r(A)$ will not be taken into consideration.

Assuming the replacements start at the same time, the maximum attack window available to an attacker is $\texttt{min}(T_r(E), T_r(B), T_r(F)) = \texttt{min}(10, 11, 11) = 10$ minutes. For example, over a period of one day, the MTD system will keep the maximum attack window for the instances belonging to node E to 10 minutes while in a static system an attack window can be as long as the entire day.

Figure 5 illustrates three possible attack windows distributions over one day (24 hours). To generate these distributions 407 adaptation points are needed in each case. As observed in Figure 5, for the same cost, the outcome may

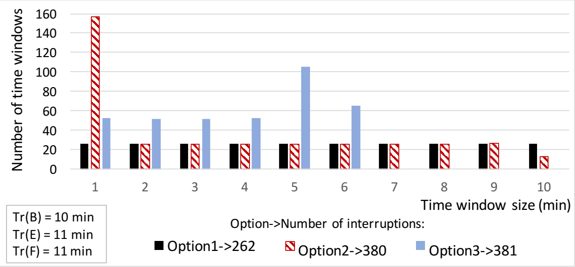

Fig. 5: Attack windows distribution over one day. The cost is 407 adaptation moments in all three cases: Option1 – 262 interruptions with starting times $(t_B, t_E, t_F) = (0, 0, 0)$, Option2 – 380 interruptions with $(t_B, t_E, t_F) = (0, 0, 1)$, and Option3 – 381 interruptions with $(t_B, t_E, t_F) = (0, 1, 6)$.

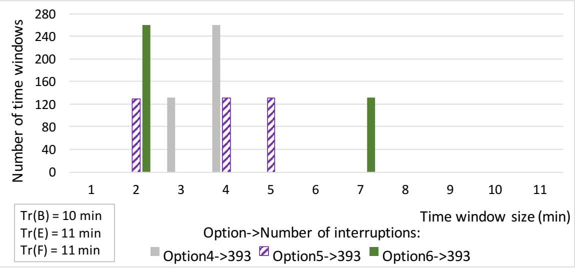

Fig. 6: Attack windows distribution over one day when no two adaptation points coincide. The cost is 393 adaptation moments for 393 interruptions in all three cases: Option4 with starting times $(t_B, t_E, t_F) = (0, 4, 7)$, Option5 with $(t_B, t_E, t_F) = (0, 4, 9)$, Option6 with $(t_B, t_E, t_F) = (0, 2, 9)$.

be very different. For instance, Option1 – 262 interruptions and 26 ten-minute attack windows when starting at (0,0,0) might not be the best option; a user can get 380 interruptions and fewer ten-minute windows for the same number of adaptation moments (cost).

In order to increase the number of interruptions while maintaining a comparable cost (number of adaptations), adaptation points should not pairwise coincide. For this reason, we can opt for a set of parameters that fall under Case 3 in Section 4.2. By setting $a(B)$ to 1 minute we have $T_r(E) = T_r(B) = T_r(F) = 11$ minutes. Next, we chose different starting times that fulfill the requirements in Case 3. Figure 6 illustrates three different such starting time options that result in the same number of interruptions, 393, for the same cost. Furthermore, we have more attack windows with the same length while the length of the maximum window is also shorter compared to Figure 5. *What if attackers learn the parameters over time?* A user may use multiple parameter sets for T_p. Moreover, T_p can also be changed.

In case of a successful attack, the maximum time an attacker may spend on an instance belonging to E, is equal to the difference between the maximum attack window and the duration of the successful attack attempt, $T_a(E)$. Thus, in the worst case scenario an attacker may spend between 4 and 10 minutes on an instance belonging to node E depending on the parameter choice (e.g., Figures 5, 6). While there are numerous options for starting times and other parameters (e.g., parameter a), a user will always be able to calculate the cost and predict the outcome in terms of number of adaptation moments.

The cost of an adaptation point is quantified in terms of the needed resources and the performance overhead (degradation) the environment can accept. The resources may include the cost for the hardware, electricity, and everything else needed to reach the desired values for the ch and d parameters. On the other hand, a (delay introduced by the user) is the parameter that can be easily changed. While increasing a has no upper bound, once $a = 0$, decreasing T_r values involves changing ch and/or d.

5 Discussion and Limitations

Numerous organizations embraced the DevOps adventure in an effort to automate their systems. An integral part of DevOps is focused on a CMT [43]. Even though MTD CBITS is not a "blanket"-like solution that simply covers existing running IT systems, adopting it is well within reach.

CMT-driven automation is the key, but it is not enough. Without an integrated inventory, instance replacements are heavily dependent on manual intervention. Using its operations model, MTD CBITS maintains an up-to-date inventory of the entire IT system and leverages it to reliably automate the instance replacements throughout the lifetime of the IT system.

On cloud infrastructures, the replacements may also constitute an efficient, user-controlled defense against various side-channel attacks. Instead of relying only on the cloud provider, the user controls the replacement operations and can regularly trigger physical host location "refreshes". The physical host where a new instance is placed depends on the cloud provider's scheduler. While public cloud scheduler rules may differ, we used the OpenStack Filter Scheduler with the default settings on our infrastructure. Although we had only thirteen compute nodes, instances "move" between nodes every replacement operation. We have deployed the eCommerce scenario (Figure 2) with 20 web applications, `webapps`. We noticed that between the initial deployment and the first whole `webapp`-cluster replacement only 3 out of 13 hosts were assigned the same number of instances, while between the first and the second replacement only 2 out of 13.

The performance loss on a cloud infrastructure can be compared in a way to Netflix's approach to test the resiliency of their IT systems. They deployed a service (called Chaos Monkey [1]) that seeks out high-availability clusters of services and randomly terminates instances within the cluster. MTD CBITS on the other hand, replaces instances proactively in an organized way for security purposes in virtualized environments (IaaS clouds). Nevertheless, physical hosts may also be managed similar to VMs by using offerings such as MaaS [10].

6 Related Work

Most MTD-related work focuses on specific aspects of system configuration, such as IP addresses [6, 20, 27], memory layouts [3, 13, 31], instruction sets [9, 29], html keywords [14, 50], SQL queries [9], or database table keywords [14]. Software diversity has also been investigated in several efforts [8, 24, 53] as a way to support multiple configurations. Although more comprehensive frameworks [30, 42] for various environments [5, 11] have been proposed, most are still conceptual, and require significant theoretical and practical development. In an attempt to provide a more efficient experimentation support for various pro-active defenses, researchers have proposed VINE [21]. Unlike MTD CBITS which captures the overall IT system and manages it throughout its lifetime, VINE enables users to create an emulated setting of an existing network on OpenStack for training and experimentation purposes.

Narain et al. used high-level specifications for network infrastructure configuration management in the ConfigAssure [39] and DADC [38] projects. Similar concepts have been proposed by Al-Shaer in MUTE [4], which uses binary decision diagrams to achieve dynamic network configurations. On the other hand, SCIT [25] has been used to achieve intrusion tolerance by restoring VM instances to their original state [40]. Our approach achieves the same intrusion tolerance as SCIT and adopts formal models similar to Narain to ensure that instance replacement(s) will not disrupt normal operations.

In terms of metrics, Okhravi et al. [41] quantitatively studied dynamic platforms as a defensive mechanism, while Cybenko and Hughes [16] introduced a quantitative framework to model diversity and showed how it can defend the three core goals of cyber security: confidentiality, integrity, and availability. Our ability to quantify cost while controlling the lengths of attack windows provides a new perspective on measuring security benefits, which may be an important component of the proposed higher-level metrics.

7 Conclusions

We propose and evaluate an MTD platform that captures service dependencies at the entire IT system level, and performs live instance replacements in a reliable way with negligible performance overhead on a cloud infrastructure. We recorded statistically non-significant differences between the baseline measurements (no MTD operations – static system) and the MTD replacement operations.

On the security side, we are able to quantify the outcome (lengths of potential attack windows) in terms of the cost (number of adaptations), and demonstrate that MTD systems managed and deployed using MTD CBITS will achieve the goal of increasing attack difficulty (e.g., restricted reconnaissance and pivoting options, limited persistent access).

MTD CBITS and ANCOR implementations, all scenarios, and auxiliary materials (e.g., supporting proofs for Cases 3 and 4 from Section 4.2, a Python implementation for an "attack windows calculator", more comprehensive benefits descriptions, etc.) are available at https://github.com/arguslab/ancor.

Acknowledgements. We would like to thank the reviewers for their valuable feedback and everyone involved in this research over the years, especially Rui Zhuang, Ali Ali, Simon Novelly, Ian Unruh, and Brian Cain. This work was supported by the Air Force Office of Scientific Research (FA9550-12-1-0106). Opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the agencies' views.

References

[1] Chaos Monkey – accessed 4/2017. https://github.com/netflix/chaosmonkey.

[2] http-perf – accessed 4/2017. https://www.npmjs.com/package/http-perf.

[3] PaX ASLR – accessed 4/2017. https://pax.grsecurity.net/docs/aslr.txt.

[4] E. Al-Shaer. Toward Network Configuration Randomization for Moving Target Def. In *Moving Target Defense*. Springer, 2011.

[5] M. Albanese, A. De Benedictis, S. Jajodia, and K. Sun. A Moving Target Defense Mechanism for MANETs based on Identity Virtualization. In *IEEE CNS*, 2013.

[6] S. Antonatos, P. Akritidis, E. P. Markatos, and K. G. Anagnostakis. Defending against Hitlist Worms using Network Addr. Space Rand. In *ACM WORM*, 2005.

[7] M. Armbust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A View of Cloud Computing. In *ACM CACM*, 2010.

[8] K. Bauer, V. Dedhia, R. Skowyra, W. Streilein, and H. Okhravi. Multi-Variant Execution to Protect Unpatched Software. In *RWS*, 2015.

[9] S. W. Boyd, G. S. Kc, M. E. Locasto, A. D. Keromytis, and V. Prevelakis. On the General Applicability of Instruction-set Randomization. In *IEEE TDSC*, 7/2010.

[10] Canonical, Metal as a Service (MAAS) – accessed 4/2017. https://maas.io/.

[11] V. Casola, A. D. Benedictis, and M. Albanese. A Moving Target Defense Approach for Protecting Resource-constrained Distributed Devices. In *IEEE IRI*, 2013.

[12] Chef – accessed 3/2017. https://www.chef.io/chef/.

[13] P. Chen, J. Xu, Z. Lin, D. Xu, B. Mao, and P. Liu. *A Practical Approach for Adaptive Data Structure Layout Randomization*. Springer International Publishing, Computer Security – ESORICS, 2015.

[14] M. Christodorescu, M. Fredrikson, S. Jha, and J. Giffin. End-to-End Software Diversification of Internet Services. In *Moving Target Defense*. Springer, 2011.

[15] Crowdstrike, Bears in the Midst – accessed 4/2017. https://goo.gl/djML8Q.

[16] G. Cybenko and J. Hughes. No Free Lunch in Cyber Security. In *MTD*, 2014.

[17] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Efficient Integrity Checks for Join Queries in the Cloud. In *IOS JCS*, 2016.

[18] Democratic National Committee – accessed 4/2017. https://goo.gl/nxemkK.

[19] DHS, Moving Target Defense – accessed 4/2017. https://goo.gl/5qXtoH.

[20] M. Dunlop, S. Groat, W. Urbanski, R. Marchany, and J. Tront. MT6D: A Moving Target IPv6 Def. In *IEEE MILCOM*, 2011.

[21] T. C. Eskridge, M. M. Carvalho, E. Stoner, T. Toggweiler, and A. Granados. VINE: A Cyber Emulation Env. for MTD Experimentation. In *ACM MTD*, 2015.

[22] D. Evans, A. Nguyen-Tuong, and J. Knight. *Effectiveness of Moving Target Defenses*. Springer New York, 2011.

[23] T. Hobson, H. Okhravi, D. Bigelow, R. Rudd, and W. Streilein. On the Challenges of Effective Movement. In *ACM MTD*, 2014.

[24] A. Homescu, T. Jackson, S. Crane, S. Brunthaler, P. Larsen, and M. Franz. Large-scale Automated Software Diversity–Prog. Evol. Redux. In *IEEE TDSC*, 2015.

[25] Y. Huang, D. Arsenault, and A. Sood. Closing Cluster Attack Windows through Server Redundancy and Rotations. In *Workshop on Cluster Security*, 2006.

[26] J. Hughes and G. Cybenko. Quantitative Metrics and Risk Assessment: The Three Tenets Model of Cybersecurity. In *Tech. Innovation Management Review*, 2013.

[27] J. H. Jafarian, E. Al-Shaer, and Q. Duan. An Effective Address Mutation Approach for Disrupting Reconnaissance Attacks. In *IEEE Transactions on Info. Forensics and Security*, 2015.

[28] N. Karapanos, A. Filios, R. A. Popa, and S. Capkun. Verena: End-to-End Integrity Protection for Web Applications. In *IEEE S&P*, 2016.

[29] G. S. Kc, A. D. Keromytis, and V. Prevelakis. Countering Code-Injection Attacks With Instruction-Set Randomization. In *ACM CCS*, 2003.

[30] A. D. Keromytis, R. Geambasu, S. Sethumadhavan, S. J. Stolfo, J. Yang, A. Benameur, M. Dacier, M. Elder, D. Kienzle, and A. Stavrou. The MEERKATS Cloud Security Architecture. In *IEEE DCS*, 2012.

[31] C. Kil, J. Jun, C. Bookholt, J. Xu, and P. Ning. Address Space Layout Permutation (ASLP): Towards Fine-grained Rand. of Commodity Soft. In *IEEE ACSAC*, 2006.

[32] P. K. Manadhata and J. M. Wing. An Attack Surface Metric. *IEEE TSE*, 2010.

[33] Mandiant, APT1 Report – accessed 3/2017. https://goo.gl/Cx3wz2.

[34] Mandiant, M-Trends 2016 Report – accessed 4/2017. https://goo.gl/PmJdEZ.

[35] Mandiant, M-Trends 2017 Report – accessed 4/2017. https://goo.gl/ISs8tX.

[36] MediaWiki – accessed 4/2017. https://www.mediawiki.org.

[37] S.-J. Moon, V. Sekar, and M. K. Reiter. Nomad: Mitigating Arbitrary Cloud Side Channels via Provider-Assisted Migration. In *ACM CCS*, 2015.

[38] S. Narain, D. C. Coan, B. Falchuk, S. Gordon, J. Kang, J. Kirsch, A. Naidu, K. Sinkar, S. Tsang, S. Malik, S. Zhang, V. Rajabian-Schwart, and W. Tirenin. A Science of Network Configuration. In *Journal of CSIAC-CSIS*, 4/2016.

[39] S. Narain, S. Malik, and E. Al-Shaer. Towards Eliminating Config. Errors in Cyber Infrastructure. In *IEEE SafeConfig*, 2011.

[40] Q. Nguyen and A. Sood. Designing SCIT Architecture Pattern in a Cloud-based Env. In *DSN-W*, 2011.

[41] H. Okhravi, J. Riordan, and K. Carter. Quantitative Evaluation of Dynamic Platform Techniques as a Defensive Mechanism. In *Research in Attacks, Intrusions and Defenses*. Springer, 2014.

[42] G. Portokalidis and A. D. Keromytis. Global ISR: Toward a Comprehensive Defense Against Unauthorized Code Exec. In *Moving Target Defense*. Springer, 2011.

[43] Puppet – accessed 4/2017. https://puppet.com/ and https://goo.gl/r1WcKm.

[44] Puppet Blog – acc. 4/2017. https://goo.gl/TSRTS0 and https://goo.gl/9Z1YhK.

[45] Puppet Hiera – accessed 4/2017. http://docs.puppetlabs.com/hiera/1/.

[46] Puppet, os_hardening – accessed 4/2017. https://goo.gl/vjkCgZ.

[47] I. Unruh, A. G. Bardas, R. Zhuang, X. Ou, and S. A. DeLoach. Compiling Abstract Spec. into Concrete Sys. - Bringing Order to the Cloud. In *USENIX LISA*, 2014.

[48] US Patent US6917930 – accessed 4/2017. https://goo.gl/KYMT9a.

[49] Verizon, 2016 DBIR – accessed 4/2017. http://goo.gl/E0OSr7.

[50] S. Vikram, C. Yang, and G. Gu. NOMAD: Towards Non-intrusive MTD Against Web Bots. In *IEEE CNS*, 2013.

[51] Wikibench – accessed 4/2017. http://www.wikibench.eu/.

[52] Wikipedia DB dumps – accessed 4/2017. https://goo.gl/8jfhkk.

[53] D. Williams, W. Hu, J. W. Davidson, J. D. Hiser, J. C. Knight, and A. Nguyen-Tuong. Security through Diversity: Leveraging VM Tech. In *IEEE S&P*, 7/2009.