EVALUATING AND QUANTIFYING THE FEASIBILITY AND EFFECTIVENESS OF WHOLE IT SYSTEM MOVING TARGET DEFENSES

by

ALEXANDRU GAVRIL BARDAS

B.S., Romanian-American University, Romania, 2008
B.A., Romanian-American University, Romania, 2009
M.S., James Madison University, 2010

AN ABSTRACT OF A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2016

Abstract

The Moving Target Defense (MTD) concept has been proposed as an approach to rebalance the security landscape by increasing uncertainty and apparent complexity for attackers, reducing their window of opportunity, and raising the costs of their reconnaissance and attack efforts. Intuitively, the idea of applying MTD techniques to a whole IT system should provide enhanced security; however, little research has been done to show that it is feasible or beneficial to the system's security.

This dissertation presents an MTD platform at the whole IT system level in which any component of the IT system can be automatically and reliably replaced with a fresh new one. A component is simply a virtual machine (VM) instance or a cluster of instances.

There are a number of security benefits when leveraging such an MTD platform. Replacing a VM instance with a new one with the most up-to-date operating system and applications eliminates security problems caused by unpatched vulnerabilities and all the privileges the attacker has obtained on the old instance. Configuration parameters for the new instance, such as IP address, port numbers for services, and credentials, can be changed from the old ones, invalidating the knowledge the attackers already obtained and forcing them to redo the work to re-compromise the new instance.

In spite of these obvious security benefits, building a system that supports live replacement with minimal to no disruption to the IT system's normal operations is difficult. Modern enterprise IT systems have complex dependencies among services so that changing even a single instance will almost certainly disrupt the dependent services. Therefore, the replacement of instances must be carefully orchestrated with updating the settings of the dependent instances. This orchestration of changes is notoriously error-prone if done manually, however, limited tool support is available to automate this process.

We designed and built a framework (ANCOR) that captures the requirements and needs of a whole IT system (in particular, dependencies among various services) and compiles them into a working IT system. ANCOR is at the core of the proposed MTD platform (ANCOR-MTD) and enables automated live instance replacements. In order to evaluate the platform's practicality, this dissertation presents a series of experiments on multiple IT systems that show negligible (statistically non-significant) performance impacts. To evaluate the platform's efficacy, this research analyzes costs versus security benefits by quantifying the outcome (sizes of potential attack windows) in terms of the number of adaptations, and demonstrates that an IT system deployed and managed using the proposed MTD platform will increase attack difficulty.

EVALUATING AND QUANTIFYING THE FEASIBILITY AND EFFECTIVENESS OF WHOLE IT SYSTEM MOVING TARGET DEFENSES

by

ALEXANDRU GAVRIL BARDAS

B.S., Romanian-American University, Romania, 2008
B.A., Romanian-American University, Romania, 2009
M.S., James Madison University, 2010

A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2016

Approved by:

Co-Major Professor
Xinming Ou, PhD.

Approved by:

Co-Major Professor
Scott A. DeLoach, PhD.

Copyright

Abstract

The Moving Target Defense (MTD) concept has been proposed as an approach to rebalance the security landscape by increasing uncertainty and apparent complexity for attackers, reducing their window of opportunity, and raising the costs of their reconnaissance and attack efforts. Intuitively, the idea of applying MTD techniques to a whole IT system should provide enhanced security; however, little research has been done to show that it is feasible or beneficial to the system's security.

This dissertation presents an MTD platform at the whole IT system level in which any component of the IT system can be automatically and reliably replaced with a fresh new one. A component is simply a virtual machine (VM) instance or a cluster of instances.

There are a number of security benefits when leveraging such an MTD platform. Replacing a VM instance with a new one with the most up-to-date operating system and applications eliminates security problems caused by unpatched vulnerabilities and all the privileges the attacker has obtained on the old instance. Configuration parameters for the new instance, such as IP address, port numbers for services, and credentials, can be changed from the old ones, invalidating the knowledge the attackers already obtained and forcing them to redo the work to re-compromise the new instance.

In spite of these obvious security benefits, building a system that supports live replacement with minimal to no disruption to the IT system's normal operations is difficult. Modern enterprise IT systems have complex dependencies among services so that changing even a single instance will almost certainly disrupt the dependent services. Therefore, the replacement of instances must be carefully orchestrated with updating the settings of the dependent instances. This orchestration of changes is notoriously error-prone if done manually, however, limited tool support is available to automate this process.

We designed and built a framework (ANCOR) that captures the requirements and needs of a whole IT system (in particular, dependencies among various services) and compiles them into a working IT system. ANCOR is at the core of the proposed MTD platform (ANCOR-MTD) and enables automated live instance replacements. In order to evaluate the platform's practicality, this dissertation presents a series of experiments on multiple IT systems that show negligible (statistically non-significant) performance impacts. To evaluate the platform's efficacy, this research analyzes costs versus security benefits by quantifying the outcome (sizes of potential attack windows) in terms of the number of adaptations, and demonstrates that an IT system deployed and managed using the proposed MTD platform will increase attack difficulty.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my co-major professors, Dr. Xinming (Simon) Ou and Dr. Scott DeLoach, for their guidance, patience, and support throughout the years.

Dr. Simon guided me through an incredible journey filled with research projects focused on tackling hard problems, and building an eco-system for teaching and doing research in cybersecurity. I feel very fortunate to have someone who has such a far-reaching perspective in the field of computer science, never avoids real hard problems, and provides crucial support at the most difficult times.

Dr. DeLoach has given me a more software engineering oriented vision on cybersecurity and on computer science in general. I am also very grateful for his support and guidance through the changing times at K-State. He was always able to put a smile on my face.

I would like to thank my committee members: Dr. Eugene Vasserman and Dr. Caterina Scoglio, for helping me improve the presentation of this work and for their helpful suggestions, and Dr. Ethan Bernick for stepping in as the outside chair on a short notice.

I would like to thank Dr. Robby for his help in enhancing this work, for his invaluable advice and for pushing me, at times, out of my comfort zone. I am grateful to the faculty and staff I interacted with throughout the years at K-State. Without their help, knowledge and friendliness, everything would have been a lot harder (if not impossible).

I would like to also thank everyone who contributed to the Moving Target Defense project over the years: Rui Zhuang, Ian Unruh, Trent Novelly, Brian Cain, Gilnei Pellegrin etc.
To my fellow, past or present, Argus members (Sathya, Loai, Xiaolong, Fengguo, Yuping, Su, etc.), I am very grateful for the great times we have spent together.

At last, I am indebted to my wife, parents, and family for their continued and unconditioned support.

This research was supported by the Air Force Office of Scientific Research (AFOSR) award FA9550-12-1-0106.

To my wife, parents, and family

Chapter 1

Introduction

The static nature of current Information Technology (IT) systems gives attackers the extremely valuable advantage of time, as adversaries are able to plan attacks at their leisure.[2] Therefore, a promising new approach to cyber security, called *Moving Target Defense* or MTD,[2;3] has emerged as a potential solution. 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. MTD-related research efforts have included randomizing IP addresses,[4–6] executable codes,[7;8] and machine instruction sets,[9;10] which help achieve the overall goal of moving target defense. These efforts, however, focus on specific aspects of a system (i.e., IP addresses, code for specific applications, and architecture of individual computers) MTD application. Only limited research has studied how to apply the MTD idea to a whole IT system.

We are viewing a *whole IT system* as a subset (component) 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 scope of a whole IT system are determined by the system engineer/administrator and can range from a one-machine service (e.g., FTP server) to more complex deployments such as multi-host eCommerce websites and Hadoop setups.

Applying the MTD idea at the whole IT system level is highly important for two reasons. First, system administrators continually struggle to monitor their IT systems for possible intrusions and compromises, patch potential vulnerabilities, maintain user access lists, or

modify firewall rules. The complexity of such IT systems and the time required to maintain them allow errors to creep into system configurations and create security holes. Creating an MTD mechanism for the whole IT system will support automation of those configuration tasks and reduce the chance for errors. Second, due to the complexity and error-proneness in configuring and maintaining a large IT system, system administrators are generally reluctant to change the system setups once they are deployed. The stagnant nature of the configuration used in the IT system gives adversaries chances to discover security holes, find opportunities to exploit them, gain/escalate privileges, and maintain persistent presence over time.[11] Introducing MTD mechanisms on the whole IT system's configuration will limit or eliminate this advantage.

While it sounds promising, little research has been done to show that MTD systems can work effectively at the whole IT system level and that security benefits can be quantified in realistic IT deployments. In general, the challenges of effective movement, as stated by Hobson et al.[12], can be summarized in three main concerns: are the right components being moved, is the movement performed in a large enough space, and is the movement taking place at the right time?

Moreover, there are a number of more specific challenges to consider. For example, there are many configuration parameters one can change in an IT system with complex dependencies. Introducing random changes will almost certainly render the system unusable. Setting up an IT system and making it function properly is already a time-consuming and complicated job. 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 also simplify system configuration and maintenance, while introducing the capability of *moving*. Changing a system while it is running inevitably introduces overhead, which must be carefully examined to determine if the benefits exceed costs. To address all these questions, it is important to be able to measure the effectiveness of an MTD mechanism at the whole IT system level, which is still lacking today.

1.1 Thesis Statement

Moving Target Defenses for a whole IT system are feasible and can offer several benefits when using a high-level abstraction that captures the objectives and dependencies at the whole IT system level.

In order to be effective, the abstraction and the framework/platform that is leveraging it should exhibit the following properties:

- Users and system engineers (i.e., a more specialized workforce) should be able to quantify the cost of changing a running system in terms of the security and maintenance benefits it presents.
- The abstraction must represent *what* a user needs instead of low-level details on *how* to implement those needs.
- The abstraction must support automatic compilation into valid running (concrete) systems on various infrastructures (e.g., cloud infrastructures). Such compilation should use well-defined knowledge units built by system engineers and be able to translate an abstract specification into different concrete systems based on low-level implementation/platform choices.
- The abstraction should facilitate long-term maintenance of the system, including replacing and reconfiguring live instances. It should also securely and reliably orchestrate those changes and aid in fault analysis and diagnosis.

1.2 Research Approach

This work addresses two main research questions:

1. Can MTD systems work effectively at the whole IT system level?

2. Can security benefits of MTD systems be quantified in realistic IT deployments?

In order to address the first question, we focused on designing and implementing a platform that supports an MTD system: enables the system to *move*, to *change*. Changes should not be noticed by benign users, they should affect only attackers while the overhead is negligible. This work proposes an abstraction that captures *what* a user needs instead of low-level details on *how* to implement those needs. The abstraction is accompanied by a process that automatically compiles the abstraction into a valid running (concrete) system. The proposed solution is packaged in a framework called ANCOR (**A**utomated e**N**terprise network **CO**mpile**R**), as described in Chapter 2.

The purpose of introducing MTD at the whole IT system level is to leverage the ANCOR framework to replace any component of an IT system with a fresh new one. In this work a component is simply a virtual machine instance or a cluster of instances.

Although ANCOR can be configured to work directly with bare-metal machines, the MTD approach is assumed to be deployed in a cloud environment. Advancements in virtualization technology have contributed significantly to the evolution of cloud computing, a movement that has the potential to revolutionize industry, and reshape the way IT systems are designed, deployed, and utilized.[13] Cloud infrastructures (e.g., OpenStack,[14] Amazon Web Services – AWS)[15] made it possible and easy to create bare-metal equivalent instances and networks resulting in the following common capabilities: provisioning instances (VMs) with various hardware capabilities, utilizing security groups, designing the desired networking layout, creating storage volumes, etc. It appears inevitable that IT systems of all sizes are moving towards the cloud, whether private, public, or hybrid.

As previously mentioned, while there are various MTD mechanisms at different levels of a system, this work refers to an *MTD system* as an IT system deployed and managed using an

ANCOR-based MTD platform, or ANCOR-MTD, that supports live instance replacements.

To address the second research question, this work analyzes the main aspects that reflect the practicality and effectiveness of the ANCOR-MTD platform:

- Performance and functionality
- Security

The first objective was to evaluate how an MTD movement process (primarily instance replacement) affects running cloud IT systems in terms of functionality and performance. This research evaluated a series of IT systems (e.g., eCommerce deployments, blogging website, Mediawiki with Wikipedia database dumps, Hadoop scenario) and determined that performance impacts are mostly negligible, statistically non-significant. Research efforts focused on applications and not on the synchronization of large amounts of persistent data.

The study also focused on determining if the movement process brings any security benefits and how these benefits can be quantified by measuring the effectiveness of an MTD system in terms of meaningful interruptions it creates for an attacker and costs associated with those interruptions. For this purpose, we introduced the notion of an *attack window*, or a continuous time interval an attacker may leverage without being interrupted by MTD's system changes. Controlling attack window sizes and their distribution can help valuably quantify potential security benefits MTD adds to the system, while indirectly increasing attackers' efforts and reducing their window of opportunity.

1.3 Contributions

This work supplies the following contributions to cybersecurity:

1. An MTD platform is presented for whole IT systems based on instance replacements via a high-level abstraction-based approach to managing IT systems. The abstraction captures dependencies among the system entities and can be used to calculate the correct values of each instance's (VM) configuration parameters *at any time* – at system creation or while the system is running.

2. The practicality of this MTD approach is analyzed through a series of experiments on realistic IT system scenarios. Experimental results show that MTD operations may have negligible impact (statistically non-significant) on normal operations of the IT systems.

3. Security benefits brought by this MTD approach are analyzed through an attack window model, showing how to leverage the model in order to quantify the security benefits of an MTD configuration.

1.4 Related Work

Although several research efforts in different areas[16;17] have been established, MTD is still in its infancy. Most previous work has focused on specific aspects of a system's configuration, such as IP addresses,[4–6] memory layouts,[18;19] instruction sets,[9;10] html keywords,[20;21] SQL queries,[22] database table keywords,[20] and so on. Additionally a few comprehensive frameworks[23;24] have been proposed, but most are still conceptual and require significant theoretical and practical effort to bring them to fruition.

Previous works introduced software diversity in order to increase the difficulty of exploiting software vulnerabilities.[7;8] Software diversity is an essential type of moving target defense technique and, whereas applied to a different layer, it has many relations to the approach presented in this work. The proposed MTD platform can be considered a compiler for configuration primitives. The process of generating those primitives can be diversified or randomized in the compilation process to further increase the "moving" dynamics of the system. This is similar to software diversity through dynamic compilation. Our work can be viewed as the first step towards this vision, and our instance reconfiguration and replacement mechanism can be extended to accommodate more radical changes of the system, instead of simply regenerating a fresh image with a pre-set configuration set up and a limited number of randomized parameters.

SCIT[25] is a technology for cleansing a machine image to achieve intrusion tolerance, and it has previously been applied in a cloud environment.[26] Our MTD platform's instance replacement process achieves the same intrusion tolerance afforded by SCIT's self-cleansing. However, the proposed ANCOR-MTD platform does this through a higher-level abstraction of the services' dependencies to ensure that instance replacement will not disrupt the cloud services' operation. Moreover we show through a series of experiments that the MTD platform instance replacement introduces a very small runtime overhead and it has the potential to provide important security benefits.

Narain pioneered the use high-level specifications for network infrastructure configuration management in the ConfigAssure[27;28] and DADC[29] projects. DADC attempts to bridge

the gap between requirement and configuration by taking formally specified network configuration constraints and automatically finding acceptable concrete configuration parameter values using a Boolean satisfiability (SAT) solver. The approach has recently been used to achieve moving target defense at the network configuration layer.[30] Similar ideas have also been proposed by Al-Shaer in the MUTE[31] framework that uses binary decision diagrams (BDDs) to achieve moving target defense on network configurations. Our proposed MTD platform adopts the same philosophy of using formal models to facilitate system management, and focuses on the configuration of applications in a cloud, which differs from that of network infrastructure.

In terms of metrics, Okhravi et al.[32] performed a quantitative study of dynamic platforms as a defensive mechanism. Specifically, the paper analyzed and evaluated how diversity, limited duration, multi-instance, and the cleanup effect of an MTD mechanism would impact the attacker's control time over an active platform, given the system vulnerability is successfully exploited. Zhuang et al.[33] also proposed an analytical model for analyzing the effectiveness of moving target defenses in terms of the success likelihood of an intrusion. The model is scalable and provides insight to designers into how the MTD mechanism would impact pivoting-type attacks. Rodes et al.[34] motivate the need for security arguments to facilitate comprehensive security metrics by introducing a framework in which security is measured by the degree of belief in a security claim.

Cybenko and Hughes[35] introduced a quantitative framework to model diversity, and they showed how it can defend the three core goals of cyber security: confidentiality, integrity, and availability. The framework quantifies the security impact in terms of a joint probability function that is described by a sequence of time-to-compromise random variables. The attack window approach described in this dissertation quantifies cost while system components are "moving", thereby providing a new perspective on measuring security benefits of an MTD mechanism. Hence it may constitute an important component for the proposed higher-level metrics frameworks.

Chapter 2

Compiling Abstract Specifications into Running Systems

Cloud computing is revolutionizing industry and reshaping the way IT systems are designed, deployed and utilized.[13] However, every revolution has its own challenges. Already, companies that have moved resources into the cloud are using terms like "virtual sprawl" to describe the mess they have created.[36] Cloud services are currently offered in several models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). While these options allow customers to decide *how much* management they want to perform for their cloud-based systems, they do not provide good *abstractions* for effectively managing those systems or addressing diverse user needs.

IaaS solutions such as Amazon Web Services (AWS) and OpenStack allow cloud users to access the raw resources (compute, storage, bandwidth, etc.); however, it forces users to manage the software stack on their cloud instances at a low level. While this approach gives users tremendous flexibility, it also allows the users to create badly configured or misconfigured systems, raising significant concerns (especially related to security).[37;38] Moreover, offering automatic scalability and failover is challenging for cloud providers because replication and state management procedures are application-dependent.[13] On the other hand, SaaS (also known as "on-demand software") provides pre-configured applications to cloud users (e.g.,

SalesForce and Google Apps). Users typically choose from a set of predefined templates, which makes it difficult to adequately address the range of user needs. PaaS (e.g., Google App Engine, Heroku, and Windows Azure) is somewhere in the middle, offering computing platforms with various pre-installed operating systems as well as services and allowing users to deploy their own applications as well. As PaaS is a compromise between IaaS and SaaS, it also inherits the limitations of both to various degrees. For example, users can be easily "locked in" to a PaaS vendor, like in SaaS, and the configuration of applications is still on the users' shoulders, like in IaaS.

We observe that existing cloud service models suffer from the lack of an appropriate *higher-level abstraction* capable of capturing objectives and functionality of *the whole IT system*. Such an abstraction, if designed well, can help both the creation and the long-term maintenance of the system. While there have been attempts at providing abstractions at various levels of cloud-based services, none have provided an abstraction that both separates user requirements from low-level platform/system details and provides a global view of the system. This has limited the usefulness of those solutions when it comes to long-term maintenance, multi-platform support, and migration from one cloud provider to another. We believe to be effective, the *abstraction* should exhibit the following properties.

1. It must be capable of representing *what* a user needs instead of low-level details on *how* to implement those needs. A major motivation for using cloud infrastructures is to outsource IT management to a more specialized workforce (called *system engineers* hereafter). Communicating *needs* from users to engineers is better served using higher-level abstractions as opposed to low-level system details.

2. It must support automatic compilation into valid concrete systems on different cloud infrastructures. Such compilation should use well-defined knowledge units built by the system engineers and be capable of translating a specification based on the abstraction (i.e., an *abstract specification*) into different concrete systems based on low-level implementation/platform choices.

3. It should facilitate the long-term maintenance of the system, including scaling the system up/down, automatic fail over, application update, and other general changes to the system. It should also support orchestrating those changes in a secure and reliable manner and aid in fault analysis and diagnosis.

We believe such an abstraction will benefit all three existing cloud service models. For IaaS, an abstract specification will act as a common language for cloud users and system engineers to define the system, while the compilation/maintenance process becomes a tool that enables system engineers to be more efficient in their jobs. Re-using the compilation knowledge units will also spread the labor costs of creating those units across a large number of customers. In the SaaS model the system engineers will belong to the cloud provider so the abstract specification and the compilation/maintenance process will help them provide better service at a lower cost. In the PaaS model we foresee using the abstraction and compilation process to stand up a PaaS more quickly than can be done today. This could even foster the convergence to a common set of PaaS APIs across PaaS vendors to support easier maintenance and migration between PaaS clouds.

Nonetheless, such a vision is not only aligned with the objective of building an MTD platform but also very needed. It's flexibility and applicability to multiple environments serve as a vital step towards making a whole IT system MTD approach a potential general security solution.

There are multiple challenges in achieving this vision. The most critical is whether it is feasible to design the abstraction so that it can capture appropriate system attributes in a way that is meaningful to users and system engineers while being amenable to an automated compilation process that generates valid concrete systems.

To demonstrate the efficacy of the proposed vision, we implemented and evaluated a fully-functional prototype of our system, called *ANCOR* (Automated eNterprise network COmpileR). The current implementation of ANCOR targets OpenStack[14] and uses Puppet[39] as the configuration management tool (CMT); however, the framework can also be targeted at other cloud platforms, such as AWS, or even tailored to work with virtualized

infrastructures, such as VMware[40] or bare-metal machines (by taking advantage of MaaS[41] or a similar approach). Other CMT solutions (e.g., Ansible,[42] Chef)[43] may also be leveraged.

This part of the research was published in "Compiling Abstract Specifications into Concrete Systems: Bringing Order to the Cloud"[44] in the USENIX Conference on Large Installation System Administration (LISA), Seattle (WA), November 2014.

2.1 Limitations of Available Automation and Abstraction Technologies

Recent years have seen a proliferation of cloud management automation technologies. Some of these solutions (e.g., , AWS OpsWorks) tend to focus on automation as opposed to abstraction. They include scripts that automatically create virtual machines, install software applications, and manage the machine/software lifecycle. Some are even able to dynamically scale the computing capacity[36;45;46]. Unfortunately, none of these solutions provide a way to explicitly document the dependencies between the deployed applications. Instead, dependencies are *inferred* using solution-specific methods for provider-specific platforms. Not only is this unreliable (e.g., applications may have non-standard dependencies in some deployments), but it lacks the capability to *maintain* the dependency after the system is generated. Ubuntu Juju[47] is a special case that is described and discussed in Section 2.8 (Background: Related Projects).

Recent years have also seen a general movement towards more abstractions at various levels of cloud services, especially in PaaS. Examples include Windows Azure Service Definition Schema (.csdef)[48] and Google AppEngine (GAE) YAML-based specification language.[49] These abstractions are focused on a particular PaaS, thus they have no need to separate the platform from user requirements. Rather, they simply abstract away some details to make it easier for users to use the particular platform to deploy their apps. The abstractions only capture applications under the users' control and do not include platform service structures. As a result the abstractions *cannot* support compiling abstract specifications to different cloud platforms. It appears that these abstractions will likely make it harder for users to move to other cloud providers as they are platform-specific.

Systems like Maestro,[50] Maestro-NG,[51] Deis[52] and Flynn[53] are based on Linux container managers (in this case Docker).[54] Some of the description languages in these systems (specifically Maestro and MaestroNG) can capture dependencies among the containers (applications) through named channels. However, these specifications abstract *instances* (virtual

machines), as opposed to the whole system. There is no formal model to define a globally consistent view of the system, and as a result once a system is deployed it is challenging to perform reliable configuration updates. Current Docker-based solutions are primarily focused on the initial configuration/deployment; maintenance is usually not addressed or they resort to a re-deployment process.

The lack of a consistent high-level abstraction describing the whole IT system creates a number of challenges in configuring cloud-based systems: network deployments and changes cannot be automatically validated, automated solutions are error-prone, incremental changes are challenging (if not impossible) to automate, and configuration definitions are unique to specific cloud providers and are not easily ported to other providers.

2.2 Enabling Technologies

Several new technologies have facilitated the development of our current prototype. In particular, there have been several advancements in the configuration management tools (CMT) that help streamline the configuration management process. This is especially beneficial to our work, since those technologies are the perfect building blocks for our compilation process. To help the reader better understand our approach, we present a basic background on the state-of-the-art CMTs.

Two popular configuration management solutions are Chef and Puppet. We use Puppet but similar concepts exist in Chef and other CMTs as well. Puppet works by installing an agent on the host to be managed, which communicates with a controller (called the master) to receive configuration directives. Directives are written in a declarative language called Puppet *manifests*, which define the *desired configuration state* of the host (e.g., installed packages, configuration files, running services, etc.). If the host's current state is different than the manifest received by the Puppet agent, the agent will issue appropriate commands to bring the system into the specified state.

In Puppet, manifests can be reused by extracting the directives and placing them in *classes*. Puppet classes use parameters to separate the configuration data (e.g., IP addresses, port numbers, version numbers, etc.) from the configuration logic. Classes can be packaged together in a Puppet module for reuse. Typically, classes are bound to nodes in a master manifest known as the *site manifest*. Puppet can also be configured to use an external program such as External Node Classifier (ENC)[55] or Hiera[56] to provide specific configuration data to the classes that will be assigned to a node.

In the current prototype we use Hiera, which is a key/value look-up tool for configuration data. Hiera stores site-specific data and acts as a site-wide configuration file, thus separating the specific configuration information from the Puppet modules. Puppet classes can be populated with configuration data directly from Hiera, which makes it easier to re-use public Puppet modules "as is" by simply customizing the data in Hiera. Moreover, users can publish their own modules without worrying about exposing sensitive environment-specific

data or clashing variable names. Hiera also supports module customization by enabling the configuration of default data with multiple levels of overrides.

Figure 2.1a is an example of a Puppet class for a `worker_queue` based on Redis.[57] Puppet classes can be reused in different scenarios without hard-coding parameters: in this particular example there is only one parameter, `port`. The concrete value of this parameter (`$exports["redis"]["port"]`) is derived from Hiera (Figure 2.1b), which is shown as `6379` but can be computed automatically by a program at runtime. This allows us to calculate parameters based on the up-to-date system model, as opposed to hardcoding them. We use this technology in the compilation process described later.

```
class role::work_queue::default {

  $exports = hiera("exports")

  class { "profile::redis":
    port => $exports["redis"]["port"]
  }
}
```

(a) Puppet Worker Queue Class

```
classes:
  - role::work_queue::default

exports:
  redis: { port: 6379 }
```

(b) Hiera Configuration Data

Figure 2.1: Puppet class and corresponding Hiera configuration data

We should also emphasize that while our current prototype uses Puppet, ANCOR can work with many mature CMT solutions such as Chef, Ansible,[42] SaltStack,[58] Bcfg2,[59] or CFEngine.[60] Two important properties are required for a CMT to be useable by ANCOR. First, the directives an agent receives dictates a *desired state* as opposed to commands for state changes, which allows configuration changes to be handled in the same way as the initial configuration. Second, there is a mechanism for reusable configuration modules (e.g., Puppet classes) that become the building blocks, or the "instruction set," into which ANCOR can compile the abstract requirements model. Depending on the specific CMT features, an orchestrator component might also be needed (especially in case the CMT employs only a pull-configuration model). An orchestrator component can be used on the CMT master node to trigger different actions on the CMT agents (achieve a push-configuration model).

2.3 ANCOR Framework

The three major components of the ANCOR framework are pictured in Figure 2.2: the Operations Model, the Compiler, and the Conductor. The arrows denote information flow.

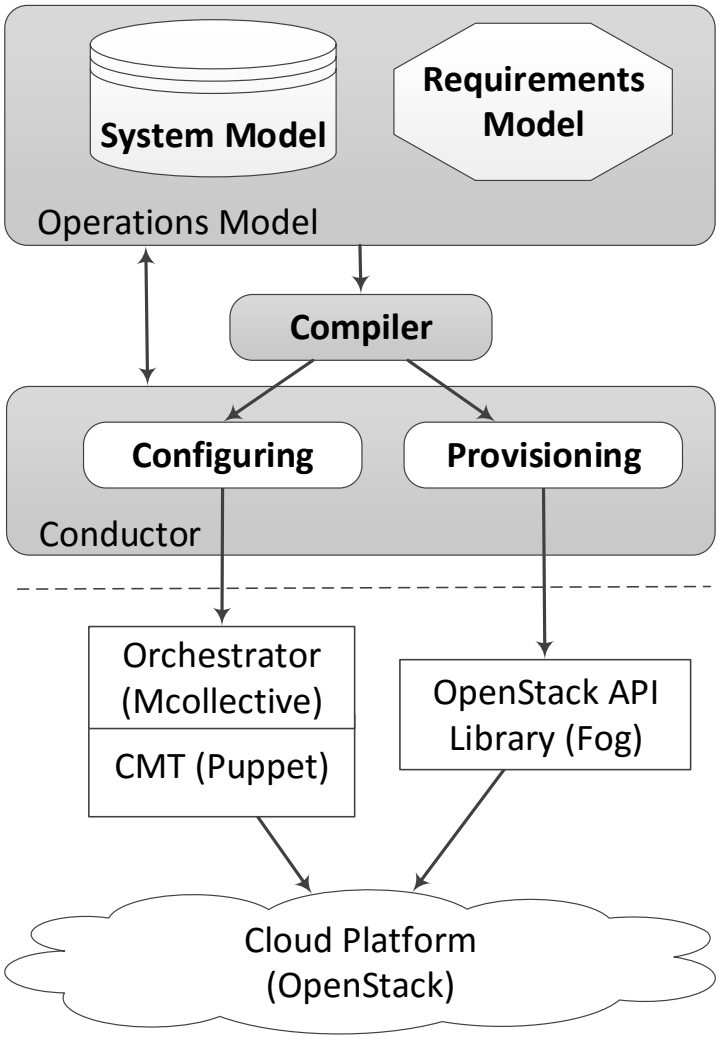


Figure 2.2: ANCOR framework

The key idea behind our approach is to abstract the functionality and structure of IT services into a model that is used to generate and manage concrete systems. This high-level abstraction is captured in the *requirements model*. We also maintain the details of the concrete system in the *system model*. The two constitute the Operations Model. When ANCOR compiles a requirements model into a concrete, cloud-based system, the system

model is populated with the details of the cloud instances and their correspondence to the requirements model. When the system changes, the system model is updated to ensure it has a consistent and accurate view of the deployment.

The Compiler references the requirements model to make implementation decisions necessary to satisfy the abstract requirements and to instruct the conductor to orchestrate the provisioning and configuration of the instances. It can also instruct the conductor to perform user-requested configuration changes while ensuring the concrete system always satisfies the requirements model.

The Conductor consists of two sub-components, Provisioning and Configuring, which are responsible for interacting with the cloud-provider API, the CMT and orchestration tools (shown below the dashed line).

The ANCOR framework manages the relationships and dependencies between instances as well as instance clustering. Such management involves creating and deleting instances, adding/removing instances to/from clusters, and keeping dependent instances/clusters aware of configuration updates. The ANCOR framework simplifies network management as system dependencies are formalized and automatically maintained. Moreover, traditional failures can also be addressed, thus increasing network resiliency.

2.4 The Abstraction

The goal of our abstraction is to maintain an accurate picture of the whole IT system at all times, an up-to-date overview of the services and their dependencies on other services. The overview can then be leveraged in conjunction with dedicated solutions targeted at managing primarily infrastructure resources (e.g., OpenStack Heat, CloudFormation) and CMTs to automatically configure and change an IT system.

As previously mentioned, the key idea behind our abstraction is to separate user requirements from the implementation details, and allow the users (system administrators) to specify what they want in terms of service structure and dependencies in an abstract way (see Figure 2.3). This high-level abstract specification (called the "requirements model") is then automatically compiled into concrete cloud-based systems, leveraging the existing configuration management tools.

The compilation process populates a "system model" that reflects the low-level system details corresponding to the requirements model. For example, the system model will provide the mapping from the individual instances to the high-level roles they play in the overall IT system. The combined system model and requirements model is called the "operations model" (Figure 2.3). The operations model captures and maintains the dependencies among the deployed instances at all times, which facilitates on-going system maintenance such as cluster expansion and contraction, or instance replacement in a reliable manner.

IT systems are described in the operations model in terms of *Goals* and *Roles*. A *Goal* is a high-level business objective whose purpose is to organize the IT capabilities (*Role*s) around business goals. A *Role* can be viewed as a single unit of configuration. Basically it represents a group of similarly configured instances that provide the same functionality. *Roles* are the means that support the accomplishment of the deployed system's *Goal*(s). For example, a user wants to deploy a scalable and highly available eCommerce website which adopts a multiple-layer architecture with various clusters of services, similar to Figure 2.4. In terms of the abstraction, the Goal of the IT system is "eCommerce" and it is supported by several Role structures such as web load balancer, web application, database master,

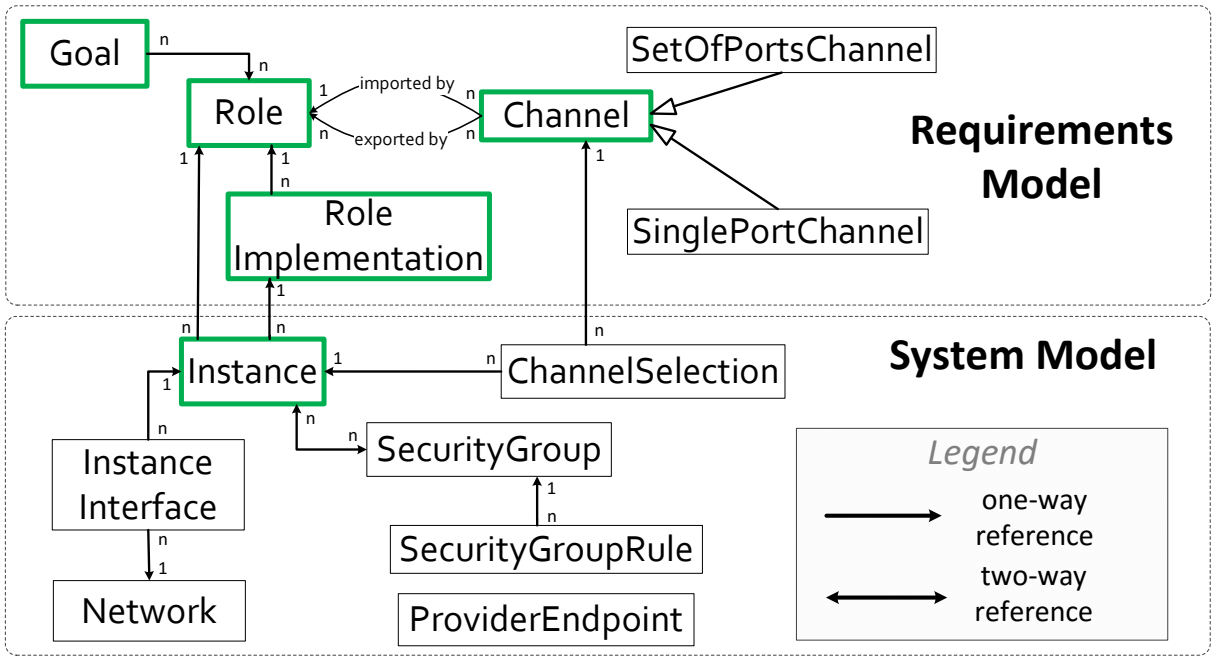


Figure 2.3: Operations model (proposed abstraction): maintains an accurate picture of the whole IT system, an up-to-date overview of the services and their dependencies on other services

database slave, messaging queue and background worker. Roles can be implemented in numerous ways, various applications and operating systems (OSs) can be chosen to fulfill a role, e.g., , web load balancer – installing Varnish[61] on an Ubuntu[62] instance or Nginx[63] on Fedora[64] can achieve the same objective.

In our abstraction, a *RoleImplementation* specifies a concrete way to implement the intended functionality embodied by a role. An implementation points to the CMT building blocks used to build a *Role*; a *Role* has one or more implementations. Moreover, an *Instance* is a virtual machine that fulfills a *Role* by implementing one of the concrete CMT modules specified in *RoleImplementations*. An instance that fulfills a *Role* makes a number of resources, *Channels*, available to other instances. These resources are usually consumed (imported) by the instances belonging to a dependent *Role*. Most of the times, the resource is a single port or a set of ports. A set of ports captures a number of ports that need to be made available at all times for a certain service to work properly. For example, usually an FTP server needs two open ports to work properly; it establishes the data channel on

one port (e.g., port 20) and the command channel on a different one (e.g., port 21). An *Instance Interface* stores the MAC address(es) that belong to an instance and a *Network* stores the network(s) that an instance is connected to. Moreover, an instance has access to the ports that a role consumes or exposes (channels) through *ChannelSelection*. The cloud provider firewall configuration (known as "security groups" in OpenStack) is captured in *SecurityGroup*. One *SecurityGroup* can have multiple configuration entries, *SecurityRules*. *ProviderEndpoint* captures the cloud platform specific API. This component makes it easier to integrate ANCOR with different cloud providers (e.g., AWS).

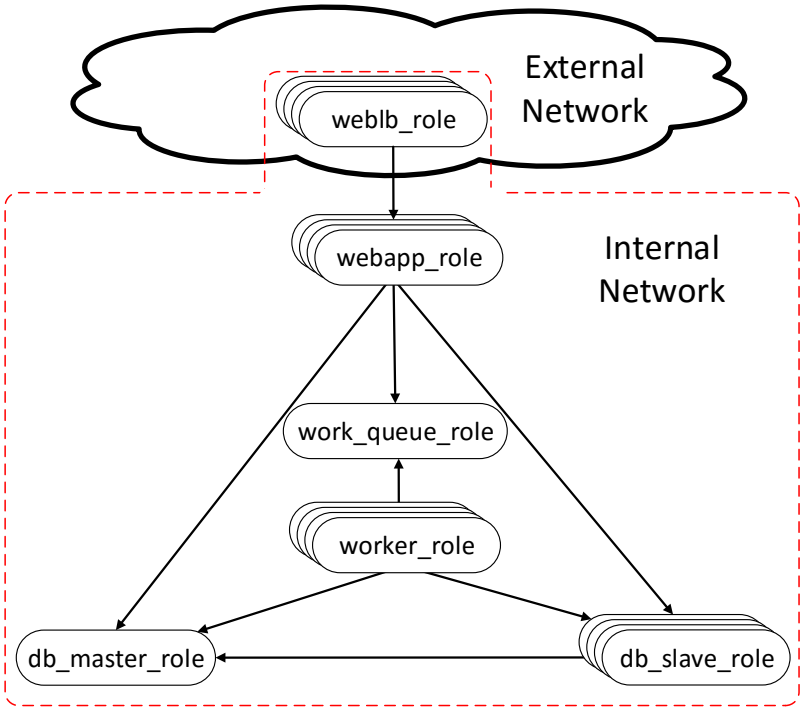


Figure 2.4: eCommerce website

2.4.1 ARML language

We specify the requirements model in a domain-specific language called the ANCOR Requirements Modeling Language (ARML). ARML's concrete syntax is based on YAML,[65] which is a language that supports specification of arbitrary key-value pairs. The *abstract syntax* of ARML is detailed in Figure 2.6.

```
goals:
  ecommerce:
    name: eCommerce deployment
    roles:
      - weblb_role
      - webapp_role
      - worker_role
      - worker_queue_role
      - db_master_role
      - db_slave_role



roles:
  weblb_role:
    name: Web application load balancer
    number_of_instances: 2
    is_public: true
    implementation_requirements:
      default:
        interface: weblb
        os: Ubuntu
        same_implementation: false
    exports:
      http:
        type: single_port
        protocol: tcp
        number: 80
    imports:
      webapp_role: http

  webapp_role:
    name: Web application
    number of instances: 3
    implementation_requirements:
      default:
        interface: webapp_rdb_mht
        os: Ubuntu
        same_implementation: true
    exports:
      http: {type: single_port, protocol: tcp}
    imports:
      db_master_role: rw_query
      db_slave_role: r_query
      work_queue_role: queue

  worker_role:
    name: Worker application
    number_of_instances: 2
    implementation_requirements:
      default:
        interface: background_worker
        os: Ubuntu
        same_implementation: true
    imports:
      db_master_role: rw_query
      db_slave_role: r_query
      work_queue_role: access

  work_queue_role:
    name: Work queue application
    number_of_instances: 2
    implementation_requirements:
      default:
        interface: work_queue
        os: Ubuntu
        same_implementation: true
    exports:
      queue: {type: single_port, protocol: tcp}

  db_master_role:
    name: Database master
    number_of_instances: 1
    implementation_requirements:
      default:
        interface: db_master_rdb
        os: Ubuntu
    exports:
      rw_querying: {type: single_port, protocol: tcp}

  db_slave_role:
    name: Database slave
    number_of_instances: 2
    implementation_requirements:
      default:
        interface: db_slave_rdb
        os: Ubuntu
        same_implementation: true
    exports:
      r_querying: {type: single_port, protocol: tcp}
    imports:
      db_master_role: rw_query
```

Figure 2.5: eCommerce website ARML specification

Figure 2.5 shows an example ARML specification for an eCommerce website. The example is a scalable and highly available eCommerce website on a cloud infrastructure, which adopts a multiple-layer architecture with the various clusters of services shown in Figure 2.4: web load balancer (Varnish), web application (Ruby on Rails[66] with Unicorn)[67], database (MySQL)[68], worker application (Sidekiq)[69], and messaging queue (Redis). Arrows indicate dependency between the clusters. Each cluster consists of multiple instances that offer the same services. Clustering supports scaling via cluster expansion (adding more instances to the cluster) or contraction (removing instances from the cluster). The clustering strategies employed by these applications fall into two main categories: homogeneous and master-slave. In a

```
ReqModel ::= goals ⇒ GoalSpec+
             roles ⇒ RoleSpec+

GoalSpec ::= goalID ⇒ [name ⇒ string]
                      roles ⇒ roleID+

RoleSpec ::= roleID ⇒ [name ⇒ string]
                      [number_of_instances ⇒ integer]
                      [exports ⇒ ChannelSpec+]
                      [imports ⇒ ImportSpec+]
                       implementation_requirements ⇒ ImplementationSpec+

ChannelSpec ::= channelID ⇒  type ⇒ "single_port" | "set_of_ports"
                             protocol ⇒ "tcp" | "udp"
                            [number ⇒ integer | integer, … ,integer]

ImportSpec ::= roleID ⇒ channelID+

ImplementationSpec ::= implementationID ⇒ interface ⇒ interfaceID
                                          os ⇒ string
                                          same_implementation ⇒ "true" | "false"

goalID, roleID, channelID, implementationID, interfaceID are symbols.
integer and string are defined in the usual way.
```

Figure 2.6: ARML's abstract syntax

homogeneous cluster all cluster members have the same configuration. If one of the instances stops working, another instance takes over. In master-slave, the master and slave instances have different configurations and perform different functions (e.g., write versus read). If the master fails, a slave can be promoted to be the master. In this example system, the web load balancer, web application, and the worker application employ the homogeneous clustering while the database employs master-slave (thus MySQL master and MySQL slaves form one cluster). Redis is used as a messaging queue. The clustering approach, mainly replication, supported by Redis is not suited for high-throughput queues.

A requirements model contains the specifications of system *goals* and *roles*. A *goal* is a high-level business goal (e.g., blog website, eCommerce website, etc.) whose purpose is to organize the IT capabilities (*roles*) around business objectives. In Figure 2.5 there is a single system goal `ecommerce` that is supported by six roles.

A *role* defines a logical unit of configuration. Examples include a database role, a web application role, a message broker role, and so on. In essence, a role represents a group of

similarly configured instances that provide the same functionality. In our model we use a single role to represent all the instances that achieve that functionality. For example, the web application instances in Figure 2.4 are configured identically (except for IP addresses, ports, and credentials) and multiple load balancers dispatch incoming web requests to the instances in the web application cluster. We have a single role `webapp_role` for all the web application instances, and a `weblb_role` for all the load balancer instances. The role-to-instance mapping is maintained in the system model.

A role may depend on other roles. A role uses a *channel* to interact with other roles. A channel is an interface *exported* (provided) by a role and possibly *imported* (consumed) by other roles. Channels could include a single network port or a set of ports. For instance, the `webapp_role` exports an `http` channel, which is a TCP port (e.g., , 80). `weblb_role` imports the `http` channel from the `webapp_role`. A role is a "black box" to other roles, and only the exported channels are visible interfaces. Using these interfaces the requirements model captures the dependencies between the roles.

The `webapp_role` also imports three channels from various other roles: `querying` from `db_master`, `querying` from `db_slave`, and `redis` from `work_queue`. This means the `webapp_role` depends upon three other roles: `db_master`, `db_slave`, and `work_queue`. The `number_of_instances` field indicates the number of instances that should be deployed to play the role. If `number_of_instances` is not specified it's default value is 1. The requirements model addresses instance clustering naturally by requiring multiple instances to play a role. For homogeneous clusters this is easy to understand. For master-slave clusters, at least two roles are involved in the cluster, the master and the slave. The dependency information captured in the export/import relationship is sufficient to support calculating configuration changes when, for example, the master is removed from the cluster and a new node is promoted to be the master. So far we have not found any real-world clustering strategies that require explicitly modeling the cluster structure beyond the dependency relationship between the roles that form the cluster. If more general clustering strategies are needed, the requirements model can be extended to support them.

2.4.2 Role Implementation

Role names are system-specific and are chosen by the user or system engineers to convey a notion of the role's purpose in the system; there are no pre-defined role names in ARML. However, to automatically compile and maintain concrete systems, system engineers must provide the *semantics* of each role, which is specified in the role specification's *implementation_requirements* field. The implementation requirements field defines how each instance must be configured to play the role. The implementation information is processed by the a module within the compiler (e.g., constraint model, described in Section 2.5). Next the conductor is informed to which of the actual implementations it should point to properly configure and deploy the concrete instances. The information about the actual implementation is thus dependent on the CMT being used. This process is similar to traditional programming language compilers where abstract code constructs are compiled down to machine code. The compiler must contain the semantics of each code construct in terms of machine instructions for a specific architecture.

The analogy between our ANCOR compiler and a programming language compiler naturally begs the question: "what is the *architecture-equivalent* of a cloud-based IT system?" In other words, is there an interface to a "cloud runtime" into which we can compile an abstract specification? It turns out that a well-defined interface between the requirements model and the "cloud runtime" is well within reach if we leverage existing CMT technologies. As explained in Section 2.2, there has been a general movement in CMT towards encapsulating commonly-used configuration directives into reusable, parameterized modules. Thus, one can use both community and custom modules to implement roles and populate those reusable knowledge units with parameters derived from our high-level requirements model.

Potential role implementations must be specified in a role's "implementation_requirements" field (see Figure 2.5). A role may have multiple implementations since there could be more than one way to achieve its functionality. The compiler then selects an appropriate role implementation from those that satisfy all constraints levied by existing role implementations in the system. More details on the selection process are presented in Section 2.5.

An important challenge was structuring the knowledge units so that they could be easily reused in different requirements models. Failing to have a proper role implementation design model would lead to rewriting every single role implementation from scratch. We adopted an approach similar to that used by Dunn.[70] We name role implementations based on their functionality and/or properties and use "profiles" to integrate individual components to embody a logical software stack.

The software stack is constructed using community and custom modules as lower-level components. In other words, profiles can be viewed as reusable custom-made classes that aggregate these lower-level community and custom modules to implement a functionality that might be needed in various role implementations across multiple scenarios. All role `implementations` used with ANCOR are available on GitHub: https://github.com/arguslab/ancor-puppet.

For instance, in case of the load balancer, let us assume that the `weblb_role` points to the `role::weblb::default` role implementation. Figure 2.7a is a Puppet class that shows the implementation that was defined as `default` for the `weblb_role`. Figure 2.7b pictures a sample of possible parameters that Puppet is getting through Hiera from the compiler for configuring one of the `weblb_role` instances. There are two parts in each role implementation (see Figure 2.7a). The code before "`---`" imports operations model values from Hiera (e.g., see Figure 2.7b). The statements `hiera("exports")` and `hiera("imports")` query Hiera to find all the channels the web load balancer will consume (*imports*) and the channels that it will make available to other roles (*exports*). These channels will be stored in two variables, `"exports"` and `"imports"`. The web load balancer will be instructed to expose an `http` channel on a particular port (in this case port 80, see "exports" in Figure 2.7b), and will be configured to use all instances that are assigned to play the `webapp_role`, from which it imports the `http` channel. Two different roles may use the same name for a resource they are exporting, even though there may be no relation between those resources (e.g., `weblb_role` and `webapp_role both` use `http` to name their exports).

```
class role::weblb::default {

  $exports = hiera("exports")
  $imports = hiera("imports")
---
  class { "profile::varnish":
    listen_port => $exports["http"]["port"] }

  $backends = $imports["webapp_role"]

  file { "default.vcl":
    ensure  => file,
    content =>
     template("role/weblb-varnish/default.vcl.erb"),
    path    => "/etc/varnish/default.vcl",
    owner   => root,
    group   => root,
    mode    => 644,
    require => Package["varnish"],
    notify  => Exec["reload-varnish"], }
}
```

(a) Web load balancer role implementation

```
{
 "exports": {
   "http": {
     "port": 80,
     "protocol": "tcp"
   }
 },
 "imports": {
   "webapp_role": {
     "webapp_role-ce66a264": {
       "ip_address": "10.118.117.16",
       "stage": "undefined",
       "planned_stage": "deploy",
       "http": {
         "port": 42683,
         "protocol": "tcp"
       }
     },
     "webapp_role-84407edd": {
       "ip_address": "10.118.117.19",
       "stage": "undefined",
       "planned_stage": "deploy",
       "http": {
         "port": 23311,
         "protocol": "tcp"
       }
     },
     "webapp_role-1ce1ce46": {
       "ip_address": "10.118.117.22",
       "stage": "undefined",
       "planned_stage": "deploy",
       "http": {
         "port": 10894,
         "protocol": "tcp"
       }
     }
   }
 },
 "classes": [
   "role::weblb::default"
 ]
```

(b) Specific `weblb_role` parameters sample exposed to Hiera by ANCOR

Figure 2.7: Web load balancer requirements and Hiera example of parameters

The default `weblb_role` implementation is based on the reusable Puppet "Varnish profile" (`profile::varnish` - see Figure 2.8). The `profile::varnish` Puppet class uses the necessary specified parameters to customize the Varnish installation. Parameters (e.g., `$listen_address`, `$listen_port`, etc.) are initialized with default values. These values will be overwritten in case they are specified in `role::weblb::default`. In the current example, `$listen_port` is the only parameter that will be overwritten (see Figure 2.7a), the other parameters will keep their default values defined in `profile::varnish`. The parameters' values (initialized in `role::weblb::default` or in `profile::varnish`) are passed to Fig-

ure 2.9a and Figure 2.9b to generate the customized Varnish configuration files, and this is all done by Puppet automatically at runtime.

```
class profile::varnish(
  $listen_address = "0.0.0.0",

# $listen_port's default value "6081" will be
# overwritten with the value passed
# from role::weblb::default
  $listen_port = 6081,

  $admin_listen_address = "127.0.0.1",
  $admin_listen_port = 6082) {

  apt::source { "varnish":
    location =>
      "http://repo.varnish-cache.org/ubuntu/",
    release => "precise",
    repos => "varnish-3.0",
    key => "C4DEFFEB",
    key_source =>
    "http://repo.varnish-cache.org/debian/GPG-key.txt",
  }
  package { "varnish":
    ensure => installed,
    require => Apt::Source["varnish"], }

  service { "varnish":
    ensure => running,
    require => Package["varnish"], }

  Exec {
   path => ["/bin", "/sbin", "/usr/bin", "/usr/sbin"]
  }

  exec { "reload-varnish":
    command => "service varnish reload",
    refreshonly => true,
    require => Package["varnish"] }

  file { "/etc/default/varnish":
    ensure => file,
    content =>
      template("profile/varnish/default.erb"),
    owner => root,
    group => root,
    mode => 644,
    notify  => Service["varnish"],
    require => Package["varnish"], }
}
```

Figure 2.8: Varnish profile for a web load balancer

```
# Configuration file for varnish
START=yes
NFILES=131072
MEMLOCK=82000
VARNISH_VCL_CONF=/etc/varnish/default.vcl
VARNISH_LISTEN_ADDRESS=<%= @listen_address %>
VARNISH_LISTEN_PORT=<%= @listen_port %>
VARNISH_ADMIN_LISTEN_ADDR=<%= @admin_listen_address %>
VARNISH_ADMIN_LISTEN_PORT=<%= @admin_listen_port %>
VARNISH_MIN_THREADS=1
VARNISH_MAX_THREADS=1000
. . .
```

(a) Web load balancer, Varnish, initialization script: *default.erb* (used in *profile::varnish*)

```
<% @backends.each do |name, backend| %>
backend be_<%= name.sub("-", "_") %> {
  .host = "<%= backend["ip_address"] %>";
  .port = "<%= backend["http"]["port"] %>";

  .probe = {
    .url = "/";
    .interval = 5s;
    .timeout = 1s;
    .window = 5;
    .threshold = 3;
  }
}
<% end %>

director webapp round-robin {
  <% @backends.each_key do |name| %>
  {
    .backend = be_<%= name.sub("-", "_") %>;
  }
  <% end %>
}

sub vcl_recv {
  set req.backend = webapp;
}
```

(b) Web load balancer, Varnish, configuration file: *default.vcl.erb* (used in *role::weblb::default*)

Figure 2.9: Web load balancer, Varnish, configuration files

Thus, a role implementation definition specifies *a concrete way to implement the intended functionality embodied by a role* by describing the invocation of pre-defined configuration modules with concrete parameters computed from the operations model. The use of a high-level requirements model that explicitly captures the dependencies among the various roles is crucial to automating this process. These role implementations are not only useful when generating the system, but also for modifying the system as it changes over time. For example, if a new instance is deployed to play the `webapp_role`, the dependency structure in the operations model allows ANCOR to automatically find all the other roles that may be impacted (those depending on the `webapp_role`) and use their role implementation to direct the configuration management tool to reconfigure them so that they are consistent with the updated operations model.

ANCOR leverages existing CMTs to define the role implementations, to minimize additional work that has to be done by the users. For example, only information in Figure 2.7a is what one needs to write for ANCOR; Figure 2.7b is generated automatically by ANCOR; Figure 2.8, 2.9a, and 2.9b are what one would have to specify anyway using Puppet.

2.5 Constraint Model

The constraint model attempts to find a compatible combination of role implementations (short-handed implementations) that will fulfill the roles defined in the ARML specification. In other words, it will match a role with the implementations that are compatible with the implementation requirements of the role, including its dependent and depended upon (dependee) roles.

The constraint model ensures that role specifications and corresponding implementations adhere to a common structure (e.g., the same number of imports and exports) and that the required operating system is supported by the specific implementation. In addition to the specification that defines the roles (ARML specification), the constraint model requires a list with the available implementations and a way to match the two specifications – using interfaces and schemas.

2.5.1 Inputs

The constraint model takes an ARML specification, a list of schemas and interfaces specs, and a list of the available implementations specs as its input, and attempts to find a compatible combination of implementations that fulfills the requirements. The list of schemas and interfaces stores the characteristics of the roles and implementations; thus the constraint model uses the list of schemas and interfaces to verify and match the implementations with the role requirements specified in ARML format. The list of available implementations describes the actual characteristics and needs of the software applications that are installed using the CMT. Every implementation from this list implements an interface specified in the schema and interface list.

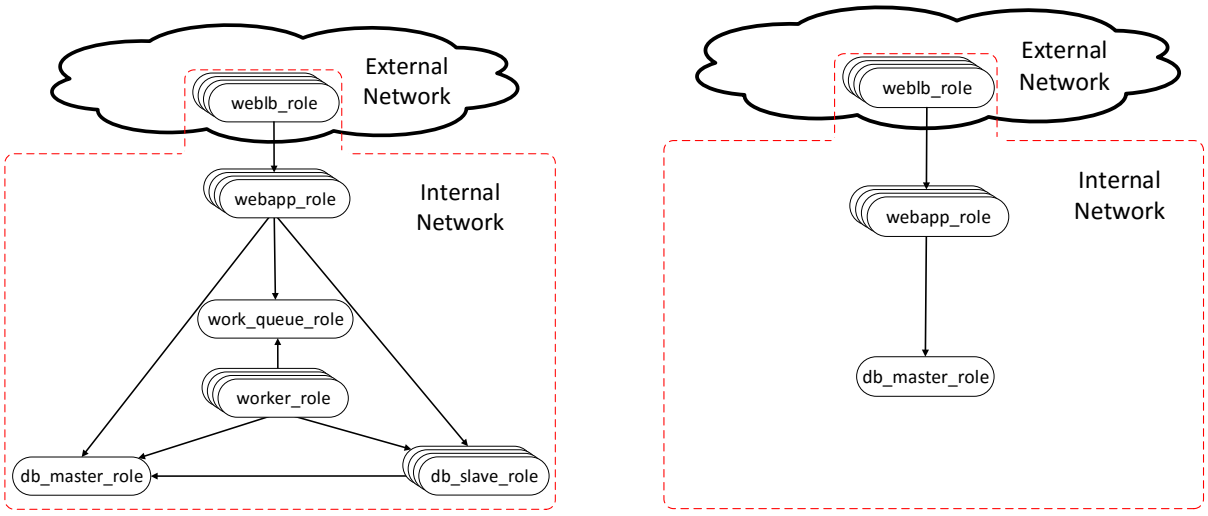


Figure 2.10: Two eCommerce deployments

Let us consider a number of different eCommerce deployments as pictured in Figure 2.10, a list of schemas and interfaces (see Figure 2.11), and the descriptions for the available implementations (Figure 2.14).

The abstract syntaxes that describe the structures of the schemas-interfaces and implementations lists are illustrated in Figures 2.12 and 2.15. Similar to ARML, the concrete syntax is based on YAML in the schemas-interfaces and implementations lists. While an ARML specification describes a whole IT system (one scenario), the list of schemas and interfaces, and the list of available implementations may be used for multiple whole IT systems (e.g., Figure 2.10).

Thus, the *name* tag of an implementation (Figure 2.14) may also be used to point to the location of the CMT modules implementing it (e.g., `role::ecommerce:weblb::nginx` points to the location of the nginx Puppet manifest in the current ANCOR prototype). The corresponding ARML specs for Figure 2.10a are pictured in Figure 2.5, while the requirements for Figure 2.10b are specified in Figure 2.13.

```
# available_schemas_and_interfaces

er exp:
  type: single_port
  protocol: tcp

ir imp

schema no_imp_one_exp:
  exports: exp

schema no_imp_two_exp:
  exports: {exp, exp}

schema one_imp_one_exp:
  exports: exp
  imports: imp

schema two_imp_one_exp:
  exports: exp
  imports: {imp, imp}

schema three_imp_one_exp:
  exports: exp
  imports: {imp, imp, imp}

schema three_imp_no_exp:
  imports: {imp, imp, imp}

interface db_master_rdb:
  conforms: no_imp_one_exp
  exports:
    rw_query: {type: single_port, protocol: tcp}

interface work_queue:
  conforms: no_imp_one_exp
  exports:
    queue: {type: single_port, protocol: tcp}

interface db_slave_rdb:
  conforms: one_imp_one_exp
  exports:
    r_query: {type: single_port, protocol: tcp}
  imports:
    db_master_rdb: rw_query

interface webapp:
  conforms: no_imp_one_exp
  exports:
    http: {type: single_port, protocol: tcp}

interface webapp_rdb:
  conforms:one_imp_one_exp
  extends: webapp
  imports:
    db_master_rdb: rw_query

interface webapp_rdb_ht:
  conforms: two_imp_one_exp
  extends: webapp
  imports:
    db_master_rdb: rw_query
    db_slave_rdb: r_query

interface webapp_rdb_mht:
  conforms: three_imp_one_exp
  extends: webapp
  imports:
    db_master_rdb: rw_query
    db_slave_rdb: r_query
    work_queue: queue

interface weblb:
  conforms: one_imp_one_exp
  exports:
    http: {type: single_port, portocol: tcp, port: 80}
  imports:
    webapp: http

interface background_worker:
  conforms: three_imp_no_exp
  imports:
    db_master_rdb: rw_query
    db_slave_rdb: r_query
    work_queue: queue
```

Figure 2.11: List of available interfaces and schemas. This list may include the schemas and interfaces descriptions used in multiple ARML specs

```
InterfaceSchemaList ::= (schema schemaID ⇒ SchemaSpec)+
                        (interface interfaceID ⇒ InterfaceSpec)+

SchemaSpec ::= [exports ⇒ Er+]
               [imports ⇒ Ir+]

Er ::= er erID ⇒  type ⇒ "single_port" | "set_of_ports"
                  protocol ⇒ "tcp" | "udp"
                 [port ⇒ integer | (integer, … ,integer)]

Ir ::= ir irID ⇒ (interfaceID ⇒ channelID)*

InterfaceSpec ::= conforms ⇒ schemaID
                 [extends ⇒ interfaceID]
                 [exports ⇒ ChannelSpec+]
                 [imports ⇒ ImportSpec+]

ChannelSpec ::= channelID ⇒ type ⇒ "single_port" | "set_of_ports"
                             protocol ⇒ "tcp" | "udp"
                            [number ⇒ integer | (integer, … ,integer)]

ImportSpec ::= interfaceID ⇒ channelID+


roleID, channelID, interfaceID, schemaID, erID, irID are symbols.
integer and string are defined in the usual way.
```

Figure 2.12: Abstract syntax for the list of interfaces and schemas

```
goals:
  ecommerce:
    name: eCommerce deployment
    roles:
     - weblb_role
     - webapp_role
     - db_master_role

roles:
  weblb_role:
    name: Web application load balancer
    number_of_instances: 2
    is_public: true
    implementation_requirements:
      default:
        interface: weblb
        os: Ubuntu
        same_implementation: false
    exports:
      http:
        type: single_port
        protocol: tcp
        number: 80
    imports:
      webapp_role: http

webapp_role:
  name: Web application
  number of instances: 3
  implementation_requirements:
    default:
      interface: webapp_rdb
      os: Ubuntu
      same_implementation: true
  exports:
    http: {type: single_port, protocol: tcp}
    imports:
      db_master_role: rw_query

db_master_role:
  name: Database master
  number_of_instances: 1
  implementation_requirements:
    default:
      interface: db_master_rdb
      os: Ubuntu
  exports:
    rw_querying: {type: single_port, protocol: tcp}
```

Figure 2.13: Lightweight eCommerce deployment

```
# available_role_implementations

  varnish:
    name: "role::ecommerce::weblb::default"
    description: Varnish webapp load balancer
    interfaces: weblb
    compatible_os: [Ubuntu, Fedora]
    imported_role_impls:
      webapp: [rails_super_lite]

  nginx:
    name: "role::ecommerce::weblb::nginx"
    description: Nginx webapp load balancer
    interfaces: weblb
    compatible_os: [Ubuntu]
    imported_role_impls:
      webapp: [rails, rails_crazy]

  rails_super_lite:
    name: "role::ecommerce::webapp::railsSL"
    description: Super lite Rails deployment
    interfaces: webapp_rdb
    compatible_os: [Ubuntu]
    imported_role_impls:
      db_master_rdb: [postgres_master]

  rails_crazy:
    name: "role::ecommerce:webapp:railscrazy"
    description: Tweaked Rails deployment
    interfaces: webapp_rdb_ht
    compatible_os: [Ubuntu]
    imported_role_impls:
      db_master_rdb: [mysql_master]
      db_slave_rdb: [mysql_slave]

  rails:
    name: "role::ecommerce::webapp:default"
    description: Rails with Unicorn and Nginx
    interfaces: webapp_rdb_mht
    compatible_os: [Ubuntu]
    imported_role_impls:
      db_master_rdb: [mysql_master, postgres_master]
      db_slave_rdb: [mysql_slave, postgres_slave]
      work_queue: [redis]

sidekiq:
  name: "role::ecommerce::worker::default"
  description: Sidekiq background worker
  interfaces: background_worker
  compatible_os: [Ubuntu]
  imported_role_impls:
    db_master_rdb: [mysql_master]
    db_slave_rdb: [mysql_slave]
    work_queue: [redis, rabbit]

 redis:
   name: "role::ecommerce:work_queue::default"
   description: Redis used as a work queue
   interfaces: work_queue
   compatible_os: [Ubuntu]

 rabbit:
   name: "role::ecommerce::work_queue::rabbit"
   description: RabbitMQ used as a work queue
   interfaces: work_queue
   compatible_os: [Ubuntu]

 mysql_master:
   name: "role::ecommerce::db_master::default"
   description: MySQL database master
   interfaces: db_master_rdb
   compatible_os: [Ubuntu]

 mysql_slave:
   name: "role::ecommerce:db_slave::default"
   description: MySQL slave
   interfaces: db_slave_rdb
   compatible_os: [Ubuntu]
   imported_role_impls:
     db_master_rdb: [mysql_master]

 postgres_master:
   name: "role::ecommerce::db_master::postgres"
   description: Postgres database master
   interfaces: db_master_rdb
   compatible_os: [Ubuntu, Fedora]

 postgres_slave:
   name: "role::ecommerce::db_master::postgres"
   description: Postgres database slave
   interfaces: db_slave_rdb
   compatible_os: [Ubuntu, Fedora]
   imported_role_impls:
     db_master_rdb: [postgres_master]
```

Figure 2.14: List of available role implementations. This list may include the implementations used in multiple ARML specs.

RoleImplList ::= *RoleImplSpec*$^+$

RoleImplSpec ::= *roleImplID* ⇒ **name** ⇒ string
[**description** ⇒ string]
interface ⇒ *interfaceID*
compatible_os ⇒ string$^+$
imported_role_impls ⇒ {(*interfaceID* ⇒ *roleImplID*$^+$)$^+$}

roleImplID, *interfaceID* is a symbol.
integer and string are defined in the usual way.

Figure 2.15: Abstract syntax for the list of role implementations

Definition 1. *A* **schema** *is a structure (outline) that is common to all adhering members.*

Definition 2. *We view an* **interface** *as a contract that specifies certain characteristics that will apply to any implementing entity. The structural constraints for an interface are specified in a schema, an interface always conforms to a schema. An interface may also inherit characteristics from another interface. Inheritance implies a union operation between the sets of characteristics in the child interface and the parent interface.*

The `webapp_role` from Figure 2.10a may have multiple implementations but although many implementations are webapps (e.g., rails, rails_super_lite, rails_crazy) that support the required OS, only `rails` is compatible with the `webapp_role` (implements the interface – `webapp_rdb_mht`). The number of `imported_impls` in `rails` differs from the number of `imported_impls` in `rails_crazy` or `rails_super_lite`. Therefore their implementations point to different interfaces (implement different contracts). However, the `webapp_role` as defined in the ARML spec in Figure 2.13 and in Figure 2.10b conforms to the `webapp_rdb` interface, and only `rails_super_lite` may be able to implement it.

An interface conforms to a schema and adheres to the structural restrictions specified in the schema description (Figure 2.11). Usually restrictions refer to the number of exports and imports – resources made available to other implementations and resources needed to implement the desired functionality. For instance, `weblb` adheres to the `one_imp_one_exp` schema and, thus, it has one import and one export. In other words, all role specifications (e.g., `weblb_role` from Figure 2.10a) and implementations specs (e.g., `nginx, varnish` from Figure 2.14) that adopt the interface `weblb` will have one import and one export.

2.5.2 Roles-Interfaces Relationship

A role implements an interface if

1. the interface name is specified in the role specification;
2. the interface and the role have the same number of `imports` and `exports`;
3. role `exports` have the same structure (same type, protocol, and port number) as the interface `exports`;
4. imported roles implement (or inherit) the interfaces stated in the interface `imports`;
5. role `imports` have the same structure (same type, protocol, and port number) with the interface `imports`.

Figure 2.16 shows an example of a role implementing an interface.

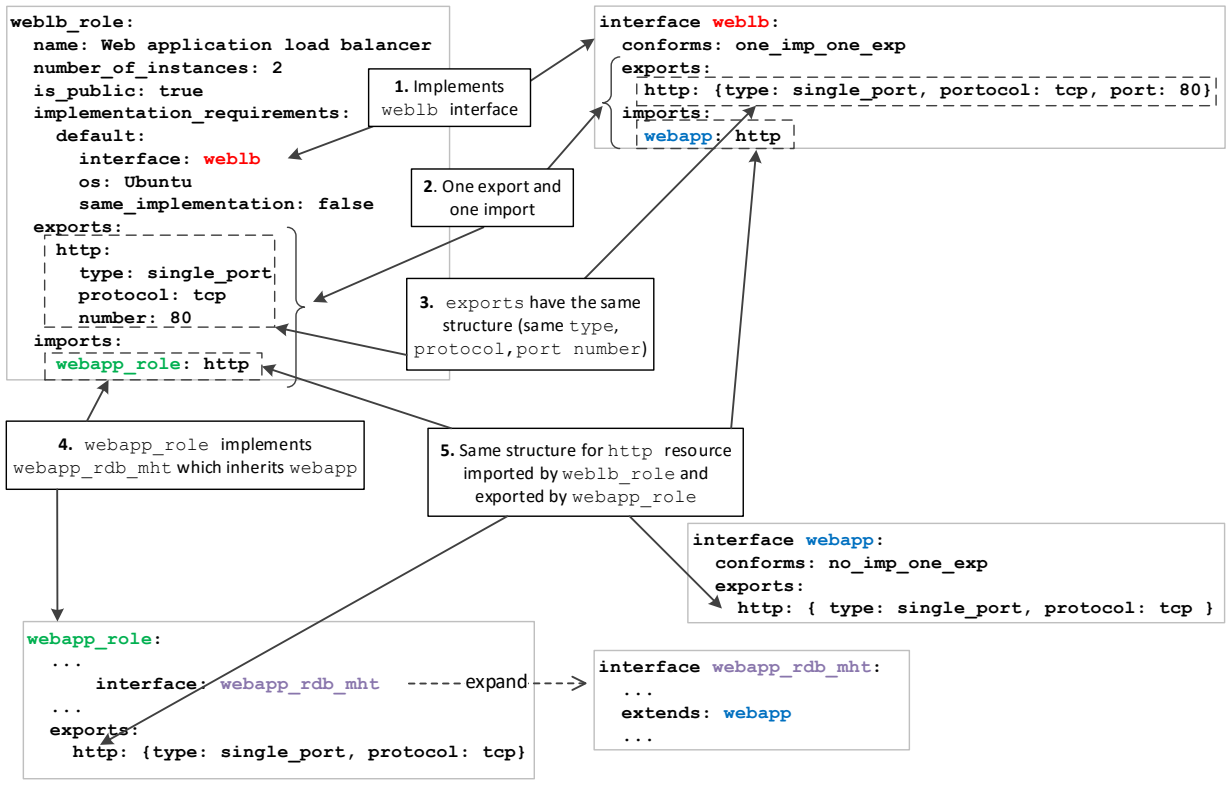


Figure 2.16: Role complies with interface example

2.5.3 Role Implementations-Interfaces Relationship

An implementation complies with an interface if

1. the interface name is specified in the implementation spec;
2. the number of interface `imports` is equal to the number of `imported_impls` from the implementation's specification;
3. the `imports`' names from the interface spec are the same with the `imported_impls` from the implementation's specification;
4. implementations (values) specified in `imported_impls` need to implement (or inherit) their corresponding interface (key) name.

Figures 2.17 shows the implications of an implementation complying with an interface.

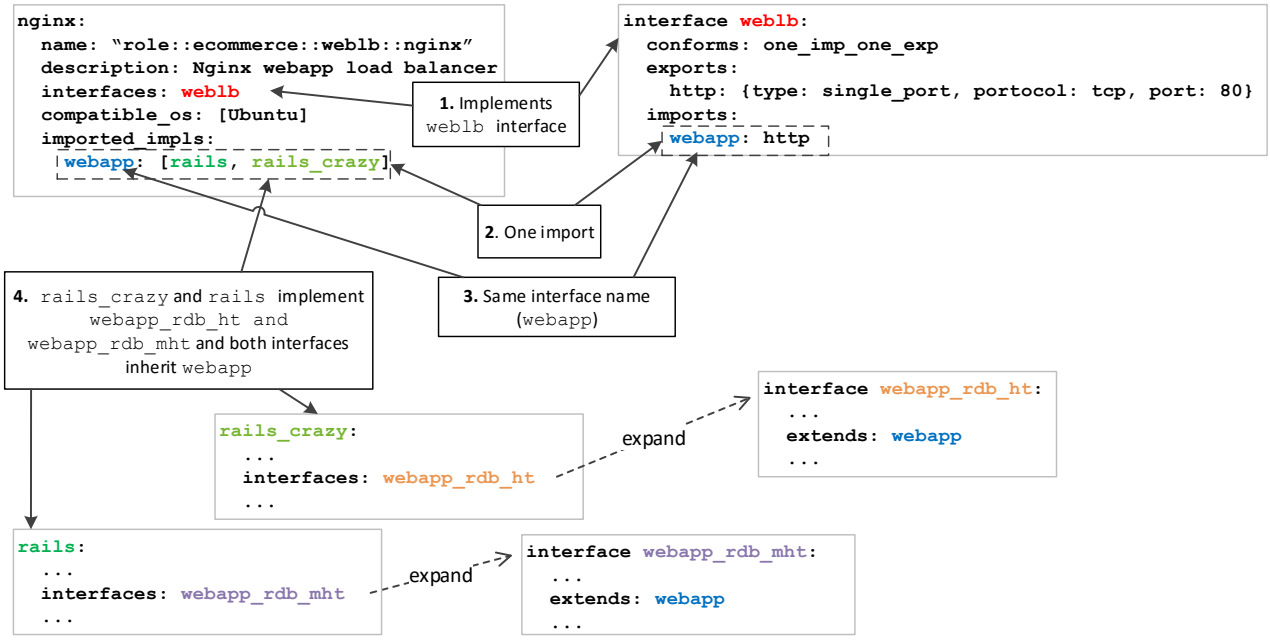


Figure 2.17: Implementation complies with interface example

2.5.4 Assigning Role Implementations to Roles

Definition 3. *A* **candidate implementation** *for a role X implements the same interface as X and supports the operating system required by X. Role X is specified in ARML format.*

Definition 4. *Implementation A* **depends on** *implementation B if B's interface is present in A's* `imported_impls`*.*

Definition 5. Compatibility *refers to implementations being able to work with each other in the same scenario. Implementation A depends on implementation B. A and B are compatible if A's* `imported_impls` *contains B. Implementations that do not depend on each other are also considered compatible (do not directly influence each other's functionality).*

A candidate implementation for a role, implements the interface and supports the required OS as specified in the role's ARML spec. In the current version of the constraint model, a role implementation supports a required OS if the OS specified in the ARML role is among the OSs specified in the implementation's specification. Such an example is pictured in Figure 2.18. A clean image of an operating system used in the list of available implementations must be supported and present on the targeted cloud infrastructure. More specific OS version names (e.g., Ubuntu14.04_x64) may also be used in the current format.

Once the constraint model was able to find at least one candidate implementation for each role, it must check the compatibility of the implementations with the implementations of the imported and dependent roles. For this purpose, the constraint model can translate the individual compatibility constraints between candidate implementations to boolean formulas and leverage the benefits of a satisfiability solver (SAT solver). Satisfiability constitutes a solution to a boolean formula that is an assignment of values to the formula's boolean variables that result in the formula becoming true[29;71]. If the formula is satisfiable, a SAT solver will produce an example of a truth assignment. The ultimate goal is to find a compatible combination of implementations that will fulfill the roles defined in the ARML specification.

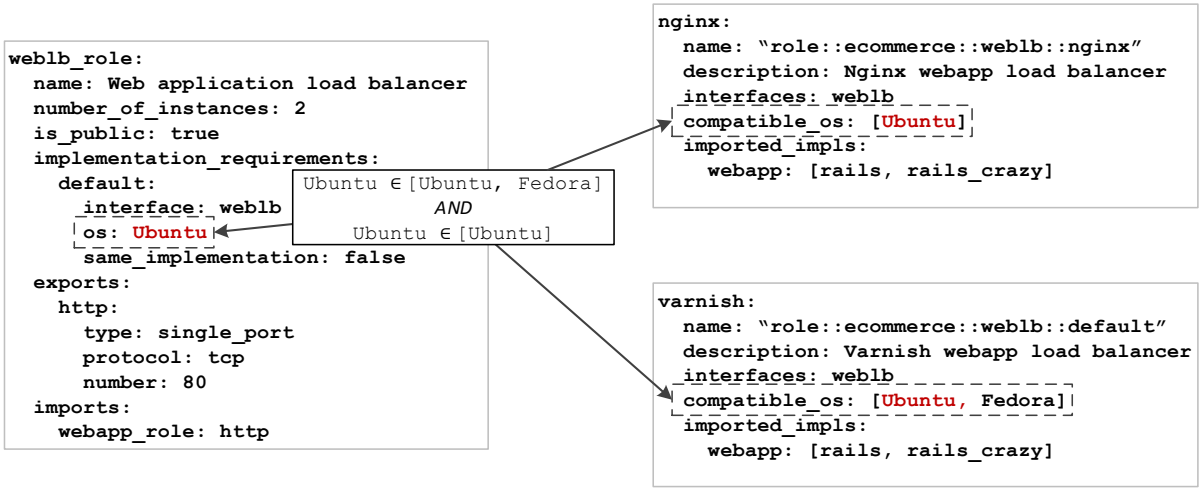


Figure 2.18: Required ARML role OS supported in a role implementation

The current version of the constraint model uses Alloy.[72] Alloy is a language for describing structures and a tool for exploring them.[71;72] Hence the constraint model populates an Alloy template model (see Figure 2.19) with information specific to the three inputs (ARML spec, list of schemas and interfaces, and list of implementations) and passes it to the Alloy Analyzer. The Alloy Analyzer works by reduction to SAT and, thus, it employs a SAT solver to find a solution.

Alloy Model

An Alloy model is built from atoms and relations. An atom is an indivisible, immutable and uninterpreted primitive entity while a relation is a structure that relates atoms.[71]

Figure 2.19 pictures snippets from the Alloy model generated based on the inputs we utilized for the eCommerce website scenario specified in Figure 2.5. Signatures (`sig`) describe the basic data elements we want to reason about and introduce the sets of atoms. We have generated an abstract signature with the appropriate fields for a role (`Role`), an interface (`Interface`) and a role implementation (`RoleImpl`). A field defines relations between the signatures while an abstract signature has no elements except those belonging to its extensions and subsets. Therefore, for each role, interface and implementation from the inputed lists we have generated one signature that extends the corresponding ab-

stract signature (e.g., `one sig Weblb_role extends Role {}; one sig Weblb extends Interface {}; one sig Varnish extends RoleImpl {}`). Extensions of the same signature are mutually disjoint. We have also defined a signature for the imported role implementations (`ImportedRi`) and use facts (`fact`) and predicates (`pred`) constructs to capture the entire eCommerce environment. Facts denote additional constraints on signatures while predicates are named constraints, with zero or more arguments. We packaged the additional constraints on the top-level signatures as predicates and then included the predicates in facts to capture the relationships between roles, interfaces and implementations.

Alloy specifications can also be viewed from an object-oriented perspective. Signatures (e.g., `Role`, `RoleImpl`, `Interface`, `Weblb_role` etc.) are analogous to classes while a field (e.g., `interface`) of a certain type (e.g., `Interface`) holds references to objects of that particular type. Moreover, relations (e.g., `interface one -> some ri`) play a role similar to Ruby hashes[73] or Python dictionaries[74] – they map elements in the domain type (e.g., `interface` from `Interface`) to elements in the range type (e.g., `ri` from `RoleImpl`).

The objective of the Alloy model is to find a compatible combination of role implementations that will fulfill the roles defined in the ARML specification. The solution is stored in the `SoftwareStack` signature and "computed" by running the `findImpl` predicate. For a candidate implementation to be associated to a role the `findImpl` predicate checks for the following conditions:

- Same interface: Role and the candidate implementation implement the same interface (e.g., `role1.interface = ri1.interface`)
- Compatibility with the implementations of the dependent roles: Implementations of the imported role(s) are among the supported implementations of the current implementation (e.g., `s.imp[role1.imported_roles] in ri1.imported_role_impls.ri`)

Optional ("sanity") checks may also be added, and they may verify whether the number of imports is the same in the role and in the candidate implementation or if the interface names of the imported roles match (or are inherited by) the interfaces in the imported role implementations.

```
module constraintModel/dependent_dependee_check

//Roles from the ARML specification
abstract sig Role {
      interface: one Interface,
      imported_roles: set Role
}

one sig Weblb_role extends Role {}
...
one sig Db_slave_role extends Role {}


//Interfaces from list of available schemas and interfaces
abstract sig Interface {}
one sig Weblb extends Interface {}
...
one sig Db_slave_rdb extends Interface {}


//Role Implementations from the list of available implementations
abstract sig RoleImpl {
      interface: some Interface,
      imported_role_impls: set ImportedRi
}

sig ImportedRi {
      interface: one Interface,
      ri: some RoleImpl,
      imported_ris: interface one -> some ri
}

one sig Varnish extends RoleImpl {}
...
one sig Postgres_slave extends RoleImpl {}

fact {

      //parameters' order: current_role, interface
      populateRoleInterface [Weblb_role, Weblb]
      ...
      populateRoleInterface [Db_slave_role, Db_slave_rdb]

      //Populating role imports parameters' order: current_role, role1, role2, etc.
      populateRoleImports [Weblb_role, none, Webapp_role, none, none, none, none]
      ...
      populateRoleImports [Db_slave_role, none, none, none, none, Db_master_role, none]

      ...  }
...

//SoftwareStack stores the solution
sig SoftwareStack {
  imp: Role -> RoleImpl
}

pred findImpl (s: SoftwareStack,role1: Weblb_role, role2: Webapp_role, role3: Worker_role,
               role4: Work_queue_role, role5: Db_master_role, role6: Db_slave_role,
               ri1: RoleImpl, ri2: RoleImpl, ri3: RoleImpl, ri4: RoleImpl,
               ri5: RoleImpl, ri6: RoleImpl) {

      role1.interface = ri1.interface
       #role1.imported_roles = #ri1.imported_role_impls
      s.imp[role1.imported_roles] in ri1.imported_role_impls.ri
      s.imp[role1] = ri1

      ...
}
run findImpl for 1 SoftwareStack, 9 ImportedRi
```

Figure 2.19: Alloy model generated based on inputs from Figures 2.5, 2.11, 2.14. Red text are values populated from the input lists, blue text is generated as needed (e.g., number of roles), black text is part of the default template.

Constraint Model Implementation

Algorithm 1 provides an overview of the constraint model's actions.

Algorithm 1 Constraint model actions. "impls" stands for implementations.

```
1: procedure CONSTRAINT_MODEL(ARML_spec, schema_interface_list, impls_list)
2:     Initialization and parsing the lists: ARML_spec, schema_interface_list, impls_list
3:     // "Type checking" and OS compatibility:
4:     for each role in the ARML spec do
5:         Find all candidate_impls
6:     end for
7:     for each role's candidate_impls do
8:         Drop candidate impls that don't support the required OS
9:         (role.os ∉ impl.compatible_os)
10:    end for
11:    // Impls compatibility with the impls of dependent and dependee roles:
12:    for each role in the ARML spec do
13:        Populate Alloy template with data about the candidate_impls and from the three
           parsed lists (ARML_spec, schema_interface_list, impls_list)
14:    end for
15:    Pass final rendered template to Alloy and run Alloy Analyzer
16:    Retrieve solution from Alloy (the solution is a combination of compatible impls that
       will fulfill the role specs or ∅)
17: end procedure
```

The current prototype uses a Ruby module to parse the YAML[75] inputs and the Ruby Erubis template engine[76] to generate the Alloy model in an automated fashion. Erubis is a fast, secure, and extensible implementation of eRuby, which means "embedded Ruby" in documents.[77] Appendix C contains a sample ERB/Erubis template for generating the Alloy model described in Figure 2.19. The constraint model may also be implemented as a stand-alone module that interacts with ANCOR when compatibility checks are needed.

2.6 ANCOR Workflow

There are four main phases involved in creating and managing cloud-based systems using the ANCOR framework.

1. Requirements model specification
2. Compilation choices specification
3. Compilation/Deployment
4. Maintenance

The first two phases result in the creation of the requirements model while the next phase performs the actual deployment of the cloud-based system. The final phase, maintenance, is performed throughout the lifecycle of the system and may include other phases within it. In a perfect world, the phases are performed sequentially; however, reality is rarely ever that neat. In general, we expect that this is an iterative process that will require the repetition of multiple phases during the lifecycle of a given system. Each of the four phases is discussed in more detail below.

Requirements Model Specification

In this phase, the user and system engineers work together to define the goals of the system, which may require significant input from various stakeholders. Next, they determine the roles required to achieve each goal and the dependencies among the roles. The high-level requirement language ARML provides an abstract, common language for this communication.

Compilation Choices Specification

In this phase, system engineers define role semantics using pre-defined CMT modules. In our current prototype this is accomplished by defining the role implementations that invoke Puppet classes as described in Section 2.4.2. If no appropriate CMT modules exist, system engineers must define new profiles, update the list of available role implementations, and,

if needed, the schemas and interfaces list. In general, system engineers should specify multiple implementation choices using various operating systems for each role to provide the constraint model flexibility in choosing the appropriate combination of implementations.

Compilation/Deployment

Once the requirements model has been defined, the framework can automatically compile the requirements into a working system on the cloud provider's infrastructure. This process has seven key steps:

1. The framework signals the compiler to deploy a specific requirements model.
2. The compiler makes several implementation decisions including the number of instances used for each role, the operating systems, and the role implementations.
3. The compiler signals the conductor component to begin deployment.
4. The conductor interacts with the OpenStack API to provision instances and create the necessary security rules (configure the cloud's internal firewall). The provisioning module uses a package such as *cloud-init* to initialize each cloud instance, including installing the CMT and orchestration tool agents (e.g., the Puppet agent and MCollective[78] agent).
5. Once an instance is live, the message orchestrator (e.g., MCollective) prepares the instance for configuration.
6. The configuration is pushed to the authenticated instances using the CMT agent and, if needed, the orchestrator (e.g., Puppet agent and MCollective).
7. System engineers may check deployed services using system monitoring applications, such as Sensu[79] or Opsview,[80] or by directly accessing the instances.

In the current implementation, the configuration from step 6 is carried out via the Hiera component, while configuration directives (node manifests) are computed on the fly using ANCOR's operations model. This ensures that the parameters used to instantiate the Puppet modules always reflect the up-to-date system dependency information.

Maintenance

System engineers can modify the system once the system is deployed in the cloud. If the change does not affect the high-level requirements model, the maintenance is straightforward. The compiler will track the impacted instances using the operations model and re-configure them using the up-to-date system information. A good example for this type of change is cluster expansion/contraction.

Cluster expansion is used to increase the number of instances in a cluster (e.g., to serve more requests or for high-availability purposes).

1. System engineers instruct the compiler to add instances to a specific role.
2. The compiler triggers the conductor component to create new instances, which automatically updates the ANCOR system model.
3. The compiler calculates the instances that depend on the role and instructs the configuration manager to re-configure the dependent instances based on the up-to-date ANCOR system model.

Cluster contraction is the opposite of cluster expansion. The main goal of cluster contraction is to reduce the number of instances in a cluster (e.g., to lower cost).

1. System engineers instruct the compiler to mark a portion of a role's instances for removal.
2. The compiler calculates the instances that depend on the role and instructs the configuration manager to re-configure the dependent instances based on the up-to-date ANCOR system model.
3. The compiler triggers the conductor component to remove the marked instances.

If the change involves major modifications in the requirements model (e.g., adding/removing a role), ANCOR will need to re-compile the requirements model. Performing "incremental recompilation" involving major structural changes without undue disruption will be a topic for future research.

2.7 Prototype Implementation

We built a prototype (see Figure 2.20) in Ruby (using Rails, Sidekiq and Redis) to implement the ANCOR framework (Figure 2.2) using OpenStack as the target cloud platform. The operations model is stored in MongoDB collections using Rails.

ANCOR employs an initial straight-forward type-checking to ensure that the ARML specification is well-formed (e.g., allowing a role to import a channel from another role only if the channel is exported by that role). More type checking tasks are performed as part of the constraint model as described in the previous sections, 2.5.2 and 2.5.3. The compiler references the MongoDB document collections that store the operations model and interacts with the conductor using a Redis messaging queue and Sidekiq, a worker application used for background processing.

The conductor interacts with the OpenStack API through Fog[81] (a cloud services library for Ruby) to provision the network, subnets and instances indicated by the compiler. Once an instance is live, the configuration module uses Puppet and MCollective to configure it using the manifest computed on the fly based on the operations model. The conductor also interacts with the system model and updates the provided system model database every time it performs a task (provisioning or configuration). Therefore, the system model stored in the MongoDB datastore will always have an updated picture of the system. Obviously, the different role implementation choices (e.g., Sidekiq, Redis or Rails) used to build the eCommerce website example scenario (Figure 2.4) are independent from the components that leverage Sidekiq, Redis and Ruby on Rails in the ANCOR framework prototype (Figure 2.2).

The current implementation uses a workflow model that is based on chained and parallel tasks processing. Once the ARML specification is entered by the user, the specification will be parsed and the requirements model will be encountered. Next, the compiler steps in and based on the requirements model it chooses the number of instances that play a role, the role implementations, the IP addresses, the channels (port number and/or sockets that will be consumed or exposed), etc. Then the compiler populates the system model and creates various tasks that it passes to the worker queue. A task can be viewed as an assignment

that is passed to a background (worker) process. In ANCOR, Sidekiq is used for background processing.

Tasks are stored in the database and have several attributes (e.g., type, arguments, state, context). A task can be related to provisioning (e.g., using Fog) or to configuring an instance (e.g., push configuration from Puppet master to Puppet agent). In case other tasks (e.g., deploy instance) depend on the execution of the current task (e.g., create network) a *wait handle* is created. Wait handles can be viewed as the mechanism used by the tasks to signal dependent tasks when they finished execution. A task creates a wait handle object that stores the ids of the tasks that wait for it to execute. Once the task finished, the wait handle triggers all the dependent tasks to execute. The purpose of a wait handle is to start, resume or suspend the dependent tasks. Using this approach we can resume or suspend a task several times including tasks related to the orchestration tool (MCollective) and the CMT (Puppet). Independent tasks (e.g., two *deploy instance* tasks) will be executed in parallel employing locks on certain shared resources.

The ANCOR prototype code, detailed instructions on how to deploy and run it, and a detailed document containing specific implementation details are available online. [1]

[1]The current ANCOR implementation is available and is distributed under the GNU (version 3) General Public License terms: https://github.com/arguslab/ancor

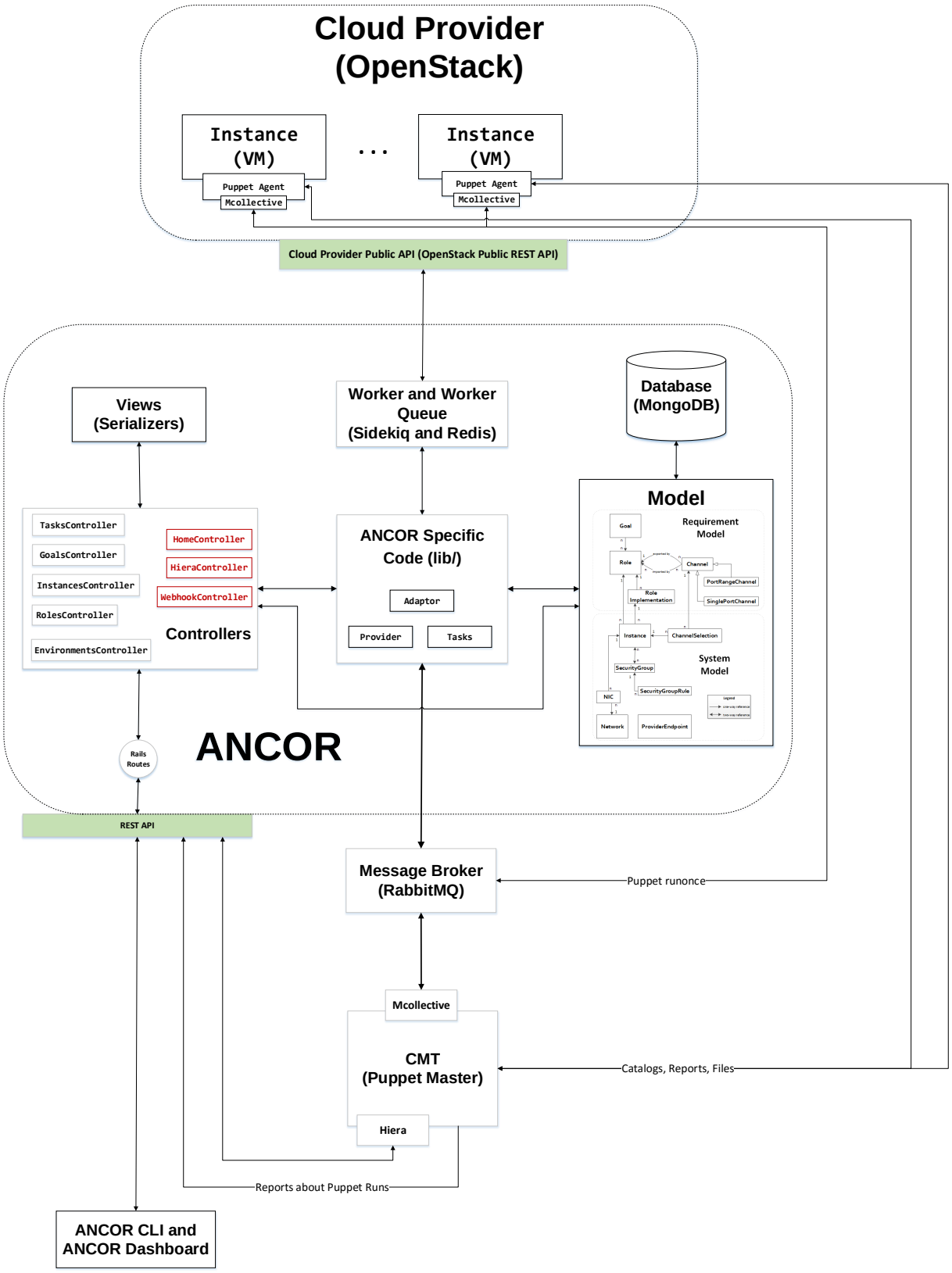


Figure 2.20: ANCOR prototype implementaion

Using ANCOR

The current framework implementation exposes a REST API[82] to it's clients (see Figure 2.20). The current clients include a Command-Line Interface (CLI), a web-browser dashboard and also the Puppet master (specifically the Hiera module). Through the REST API, Hiera is obtaining the specific configuration details (e.g., imported and exported channels - `$exports` and `$imports` arrays, see Figure 2.1a) from the compiler in order to customize the Puppet modules that are part of the chosen role implementation (e.g., see Figure 2.1b). The CLI and the dashboard are used to deploy, manage, visualize (in case of the dashboard) and delete ANCOR deployments.

One can use the CLI to deploy, manage and delete the eCommerce website example using several key commands:

1. `ancor environment plan eCommerce.yaml` – plans the deployment (`eCommerce.yaml` is shown in Figure 2.5)
2. `ancor role list` – lists the current set of roles
3. `ancor instance list` – lists the current set of instances
4. `ancor environment commit` – deploys the environment on the cloud infrastructure
5. `ancor task list` – displays the current progress of the deployment
6. `ancor instance add webapp_role` – used to add a new `webapp_role` instance after all `tasks` are `completed`
7. `ancor environment remove production` – deletes the current deployment

More options and instructions on using the ANCOR CLI and the dashboard are available on the framework's website (https://github.com/arguslab/ancor).

2.8 Background: Related Projects

A general trend has emerged towards creating more abstractions at various levels of cloud service offerings. Some of the solutions even use similar terminologies and features as those in ANCOR. For example, some solutions also use the term "role" in a similar way to ours,[48;70] and others have adopted named channels to describe dependencies in configuration files[48;50;51]. Thus it is important to describe the fundamental differences between the abstraction used in ANCOR and those in the other solutions, so that one does not get confused by the superficial similarities between them.

Abstractions used in the various PaaS solutions such as the Windows Azure service definition schema[48] and Google AppEngine YAML-based specifications,[49] allow users to define their cloud-based applications. These abstractions, while useful for helping users to use the specific PaaS platform more easily, do not serve the same purpose as ARML. In particular, they only define user-provided applications and not the whole (complete) IT system in the cloud, since the important platform components are not modeled. Thus these abstractions cannot be used to compile into different implementation choices or platforms. They are tied to the specific PaaS platform and thus will never separate the user requirements from the platform details. Using these abstractions will lock the users in to the specific PaaS vendor, while ANCOR will give users complete flexibility as to implementation choices at all levels, including platforms.

Docker container-based solutions such as Maestro, Maestro-NG, Deis,[52] and Flynn[53] provide management aid for deploying cloud instances using the Linux containers virtualization approach. Some of them (Maestro and Maestro-NG) also have environment descriptions (in YAML) for the Docker instances that include named channels to capture dependencies. These solutions can automate initial deployment of cloud instances and make it easier to stand up a PaaS, but they take a different approach and do not provide the same level of abstraction that supports the vision outlined at the beginning of this chapter. Specifically, the abstractions provided by their environment descriptions are focused on instances as opposed to the whole IT system, the container is the unit of configuration, and maintenance tasks

Offering	Focus	Platform	Multiple Cloud Infrastructures	CMTs (or similar technologies)
OpenShift	Private PaaS	RHEL	Yes	None
Flynn	Private PaaS	Linux	Yes	Heroku Buildpacks, Docker
Deis	Private PaaS	Linux	Yes	Heroku Buildpacks, Chef Docker
OpsWorks AWS	General	Ubuntu	No	Chef
Maestro	Single-host	Linux	Yes	Docker
Maestro-NG	General	Linux	Yes	Docker
Google AppEngine	PaaS	Linux	No	None
Heroku	PaaS	Linux	No	Heroku Buildpacks
Windows Azure	PaaS	Windows	No	None
Ubuntu Juju	General	Ubuntu, CentOS, Windows	Yes	Charms
ANCOR	General	Any	Yes	Puppet

Table 2.1: Current solutions comparison

are done by progressing through containers. It is not clear how much the container-based solutions can help alleviate the long-term maintenance problem of cloud-based IT systems. We also summarized a few other features to differentiate ANCOR from the other solutions in Table 2.1. ANCOR can be used to deploy systems using other orchestration tools such as Flynn and Deis in conjunction with traditional IT systems. As shown in Table 2.1, ANCOR is currently the most general and flexible management solution available.

Several companies are developing cloud migration technologies. While some appear to internally use abstractions to support migration,[83;84] no details are available for independent evaluation. Our approach is more fundamental in the sense that we *build* systems using the abstraction and, smoother and more reliable migration could be a future product of our approach. Rather than creating technology specifically to replicate existing systems, we aim to fundamentally change the way cloud-based systems are built and managed, which includes enabling dynamic and adaptive system changes, reducing human errors, and supporting more holistic security control and analysis.

Often, solutions like AWS CloudFormation,[85] OpenStack Heat[86] or Terraform[87] may be, mistakenly, viewed as being at the same level of abstraction with ANCOR. These solutions

are primarily focused on building and managing the infrastructure (cloud resources) by allowing the details of an infrastructure to be captured into a configuration file. CloudFormation and Heat manage AWS/OpenStack resources using templates (e.g., Wordpress template,[88] MySQL template,[89] etc.), and they do not separate user requirements from system implementation details. The templates have the potential to integrate well with configuration management tools but there is no model of the structure and dependencies of the system. Thus, it cannot achieve one main objective of ANCOR which is to use the operations model to maintain the system, e.g., updating dependencies automatically while replacing instances. Terraform similarly uses configuration files to describe the infrastructure setup, but it goes even further by being cloud-agnostic and by enabling multiple providers and services to be combined and composed.[90]

Juju[47] is a system for managing services and works at a similar level as ANCOR. It resides above the CMT technologies and has a way of capturing the dependencies between software applications (services). It can also interact with a wide choice of cloud services or bare metal servers. The Juju client works on multiple operating systems (Ubuntu, OS X, and Windows) but Juju-managed services run primarily on Ubuntu servers, although support for CentOS and Windows has been announced but is, currently, not widely available.[91] While we were aware of the existence of Juju when working on ANCOR, the lack of formal documentation on how Juju actually works, the services running only on Ubuntu, and the major changes in the Juju project (e.g., code base was completely rewritten in the Go programming language) kept us away from this project.

We recently reevaluated Juju and discovered fundamental similarities between ANCOR and Juju. Even so, there are subtle differences that make the two approaches work better in different environments. For instance, the ANCOR approach adopts a more "centralized" management scheme in terms of deciding the configuration parameters of dependent services, while Juju adopts a negotiation scheme between dependent services (called relations in Juju) to reach a consistent configuration state across those services. Depending on the need for change in the system, the ANCOR approach may be more advantageous when it comes to a highly dynamic system with proactive changing (e.g., an MTD system).

Our approach has benefited from the recent development in the CMT technologies that have provided the building blocks (or "instruction sets") for our compiler. The general good practice in defining reusable configuration modules such as those advocated by Dunn[70] is aligned very well with the way we structure the requirements model. Thus our approach can be easily integrated with those CMT technologies.

Sapuntzakis et al.[92] proposed the configuration language CVL and the Collective system to support the creation, publication, execution, and update of virtual appliances. CVL allows for defining a network with multiple appliances and passing configuration parameters to each appliance instance through key-value pairs. The decade since the paper was published has seen dramatic improvement in configuration management tools such as Puppet[39] and Chef, which has taken care of specifying/managing the configuration of individual machines. Our work leverages these mature CMTs and uses an abstraction on a higher level. ARML does not need to mention details on host information such as virtual machine configuration files or network interfaces, which is taken care of by the CMT and cloud infrastructure. In particular, ARML specifies the dependency among roles through explicit "import" and "export" statements with the channel parameters, which are translated automatically to concrete protocol and port numbers by the integration of the operations model and the CMT. While CVL does specify dependency among appliances through the "provides" and "requires" variables, they are string identifiers and not tied to configuration variables of the relevant appliances (e.g., the "DNS host" configuration parameter of an LDAP server). In the CVL specification of the virtual appliance network, the programmer would need to take care in passing the correct configuration parameters (consistent with the dependency) to the relevant appliances. In ANCOR this is done automatically by the coordination between the CMT and the operations model (compiled from the high-level ARML specification). This also allows for easy adaptation of the system (e.g., cluster expansion and contraction).

Begnum[93] proposed MLN (Manage Large Networks) that uses a light-weight language to describe a virtual machine network. Like ANCOR, MLN uses off-the-shelf configuration management solutions instead of reinventing the wheel. A major difference between ANCOR and MLN is that ANCOR captures the instance dependency in the requirements model,

which facilitates automating configuration of a whole IT system and its dynamic adaptation. ANCOR achieves this by compiling the abstract specification to the operations model, which is integrated with the CMT used to deploy and manage the instances.

Plush[94] is an application management infrastructure that provides a set of abstractions for specifying, deploying, and monitoring distributed applications (e.g., peer-to-peer services, web search engines, social sites, *etc.*). Although Plush's architecture is flexible, it is not targeted at cloud-based enterprise systems and it is unclear whether system dependencies can be specified and maintained throughout the system life cycle.

Use of higher-level abstractions to improve system management has also been investigated in the context of Software-Defined Networking (SDN). Monsanto et al. introduced abstractions for building applications from independent modules that jointly manage network traffic.[95] Their Pyretic language and system supports specification of abstract network policies, policy composition, and execution on abstract network topologies. Our ARML language and the entire ANCOR system adopts a similar philosophy for cloud-based deployment and management.

2.9 Discussion

Our requirements specification approach and the implemented ANCOR framework offer system engineers the same flexibility as in a typical IaaS model. This means that engineers can keep their workflow using their preferred configuration management tools (e.g., Puppet, Chef, Ansible) and orchestration tools (e.g., MCollective). They have the option to do everything in their preferred ways up to the point where they connect the components (services) together. For example, system engineers have the option of using predefined configuration modules and of leveraging the contributions from the CMT community. Or they can write their own manifests or class definitions to customize the system in their own ways. ANCOR can leverage all of these and does not force the system engineers to use particular low-level tools or languages; rather it provides the ability to manage the whole system based on a high-level abstraction.

The high-level requirements model we developed could also facilitate tasks like failure diagnosis and system analysis to identify design weaknesses such as single point of failures or performance bottlenecks. The system dependency information specified in the requirements model and maintained in the operations model allows for more reliable and streamlined system updates such as service patching. It also allows for a more fine-grained firewall setup (i.e., only allows network access that is consistent with the system dependency), and enables porting systems to different cloud providers in a more organized manner (e.g., one can take the up-to-date requirements model and compile it to a different cloud provider's infrastructure, and then synchronize data from the old one to the new one).

2.10 Summary

Separating user requirements from the implementation details has the potential of changing the way IT systems are deployed and managed. To capture user requirements, we developed a high-level abstraction called the requirements model for defining IT systems. Once users define their desired system in the specification, it is automatically compiled into a concrete cloud-based system that meets the specified user requirements.

We demonstrate the practicality of this approach in the ANCOR framework. ANCOR manages the relationships and dependencies between instances as well as instance clustering. Such management involves creating and deleting instances, adding/removing instances to/from clusters, and keeping dependent instances/clusters aware of configuration updates. The ANCOR framework simplifies network management as system dependencies are formalized and automatically maintained. The current implementation targets a cloud infrastructure (OpenStack) and leverages the Puppet configuration management tool.

Chapter 3

A Moving Target Defense Platform for Whole IT Systems

Our approach of introducing moving-target defense at the whole IT system level is to create a platform where any component of the IT system can be replaced with a fresh new one. 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[14] and AWS) made it possible and easy to create bare-metal equivalent virtual machine instances and networks. Moreover, the cloud is considered more and more for various types of tasks (e.g., offloading jobs[96]) involving all types of network devices. It appears inevitable that IT systems of all sizes are moving towards the cloud — be it private, public, or hybrid. There are a number of security benefits of such replacements.

- By replacing a VM instance with a new one, the OS and applications will be installed at the most up-to-date version, eliminating security problems caused by unpatched vulnerable software.
- If the attacker already compromised an instance, replacing it with a fresh new one will eliminate the attacker's foothold on the instance.
- The fresh new instances will have everything installed from a clean slate. The key

configuration parameters for the new VM, such as IP addresses, port numbers for services, authentication credentials, etc. can be changed to a new value hard to predict. This will increase the difficulty for the attacker to re-compromise the instance – the knowledge about the old system is no longer valid.

- The replacement can happen proactively or be triggered by security events. Mingling the two will make it hard for the attacker to know whether the attack activities have been discovered or not.

- When replacement is triggered by security events (e.g., IDS alerts), it is a low-cost way to deal with potential false positives in such events since the replacement will not disrupt the normal functionality of the whole IT system.

The above shows that even such a simple maneuver of replacing VMs in an IT system can have significant benefit on security. However, such simple maneuvers have been non-trivial to perform. Modern enterprise IT systems have complex dependencies among services, so that changing one instance alone will almost certainly disrupt other instances that depend upon it. Thus replacing VM instances must be carefully orchestrated with modifying configuration settings of the dependent instances. Such orchestration of changes are notoriously error-prone if done manually. Currently there is a limited tool support to automate this process. As a result, many IT systems remain stagnant due to the fear that changing them may break the working systems. Even applications with known patchable vulnerabilities are kept running for long periods of time until the next system maintenance cycle, which often requires down time.[97] It is even beyond the current capability to replace running VM instances proactively. [1]

[1] with perhaps one exception, the Chaos Monkey introduced by Netflix[98]

3.1 ANCOR-MTD Platform

The ANCOR framework provides a solid foundation for the moving target techniques introduced in this work. Therefore, we leverage the ANCOR framework to build an MTD platform for whole IT systems.

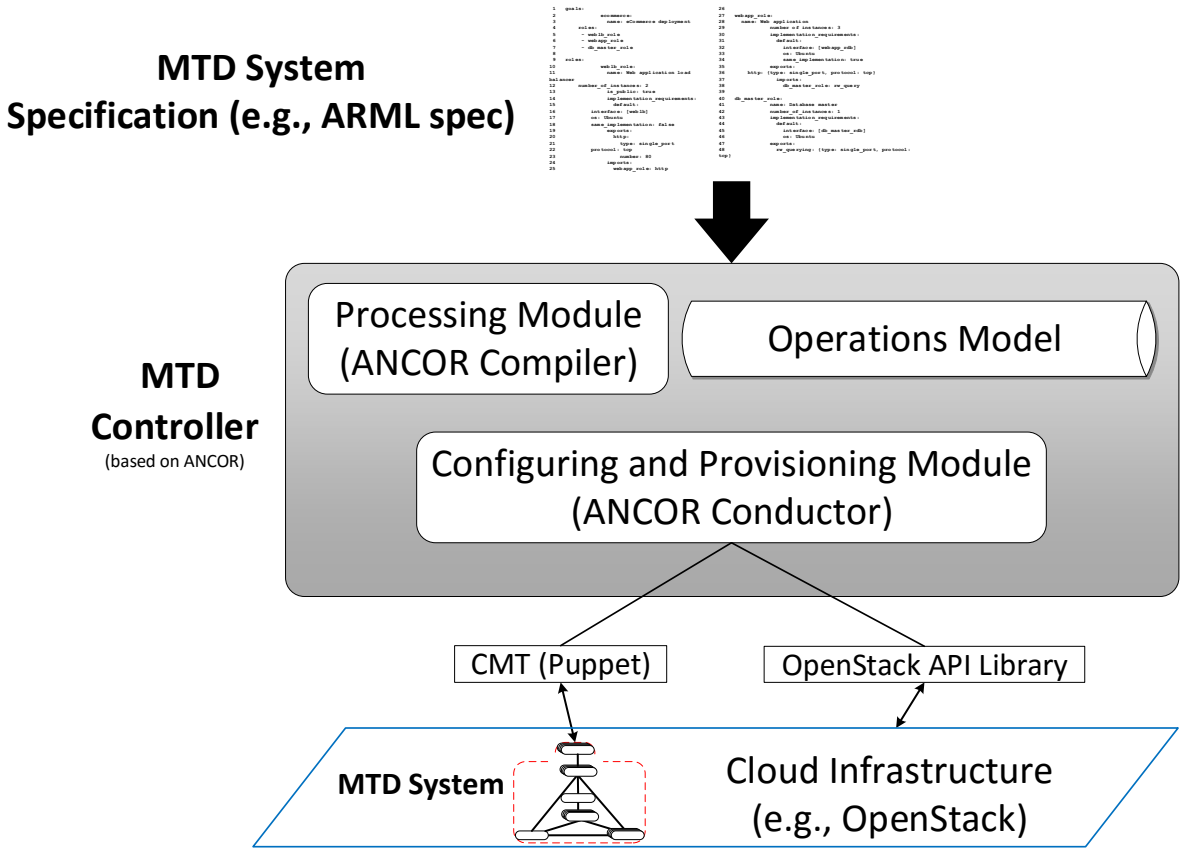


Figure 3.1: ANCOR-MTD platform taking an abstract specification of an IT system as input and creating and managing the corresponding concrete system on a cloud

As shown in Figure 3.1 our current MTD platform is based on the ANCOR prototype, it targets OpenStack and leverages the Puppet CMT. The implementation can be changed to work with other cloud infrastructures (e.g., AWS) and CMTs (e.g., Ansible). In this work, we refer to an *MTD system* as an IT system deployed and managed using our ANCOR-MTD platform that supports dynamically replacing instances. The *MTD controller* is used to deploy and manage the MTD systems: it can reach the OpenStack API, it hosts the

Puppet master, and it 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, so it communicates with the agents over an internal isolated network. Moreover, the communication between the Puppet master and the agents is encrypted.

The ANCOR-MTD 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 3.1). The key feature of the platform is that 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 inventory captures the user's requirements and explicitly documents the dependency among the instances, which nowadays only exists in a system administrator's mind. This data is passed to Puppet through Hiera, a key/value look-up tool for configuration data. Whenever a change happens in the deployed MTD system, it is also recorded in the operations model. This way we ensure that the operations model always maintains up-to-date information about the running IT system. More details about the underlying ANCOR framework are presented in Chapter 2.

3.2 Instance Replacement

Using the operations model, the platform facilitates a variety of *adaptation* operations (movements) for the managed IT systems, creating a moving-target defense. When a user requests an action, e.g., creating new instances, updating configurations, etc., the processing module updates the inventory and triggers the configuring and provisioning module to perform the tasks in an organized manner (see Figure 3.1). The processing module is responsible for computing the concrete configuration parameters, for keeping the inventory up-to-date, and for overlooking the whole process. In addition, every instance in an MTD system has its own security group,[99] an external firewall provided by the OpenStack infrastructure. Security groups are also computed from the inventory by the processing module and configured using the provisioning module, to allow an instance to communicate *only* with instances belonging to dependent and depended-upon (dependee) instances.

In our MTD approach, live instance replacement is achieved by a sequence of adaptations: adding new instances, reconfiguring dependent instances to use the new instances, and removing the old instances. An MTD system may proactively or reactively perform instance replacement. In case the MTD system uses high-availability clusters of services, the overhead and interruptions will be negligible as detailed in Chapter 4.

Reconfiguring Instances In-place reconfigurations (updated CMT directives) may include internal service changes such as changing service parameters (e.g., credentials), updating to a new version, applying service and OS patches, etc., or changes that involve dependent roles (e.g., pointing to updated port numbers and IP addresses). These changes will be accompanied by security group updates.

Adding or Removing Instances The MTD platform enables the addition and removal of running instances. Both adapations 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 (see Figure 3.2). When adding a new instance, the updated configuration directions are sent to the dependent instances (push configuration

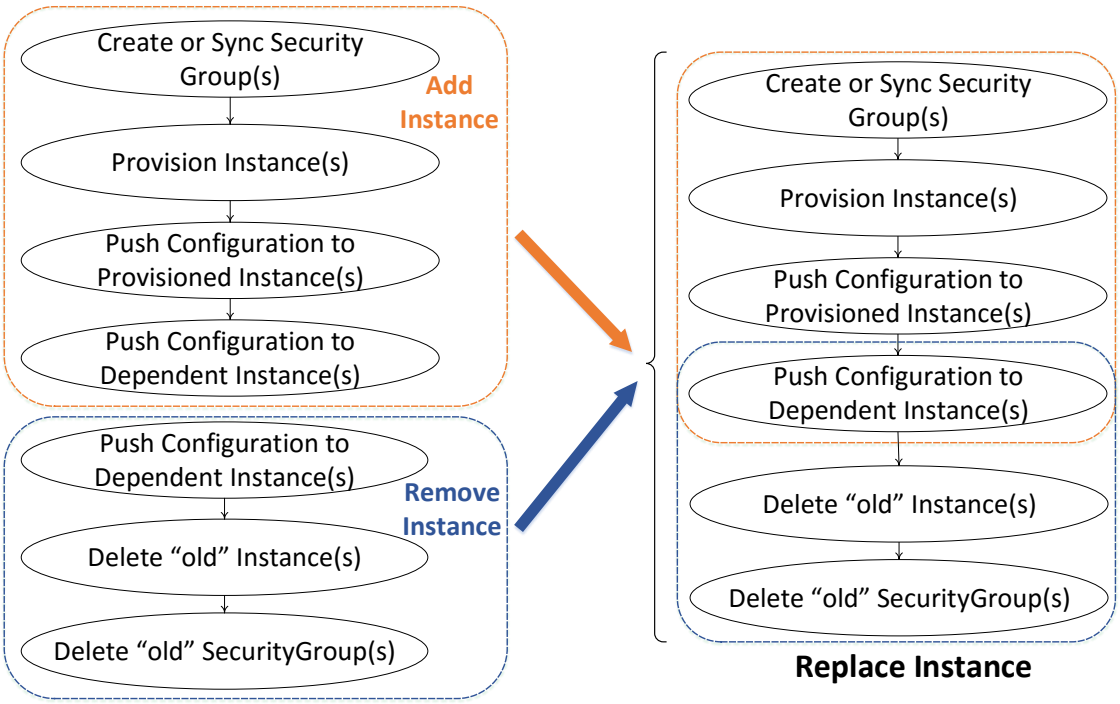


Figure 3.2: The *Instance Replacement Process* merges the *Add Instance* and *Remove Instance* operations through a sequence of tasks carried out via the provisioning component and the CMT. Affected dependent services are notified using a set of updated CMT directives.

to dependent instances) after the new instance was provisioned and configured. On the other hand, when removing an instance, first, the dependent instances are notified about the change before the actual deletion takes place. In this way, the MTD system's functionality will not be affected during the change process. Moreover, this makes it possible for the MTD platform to gracefully recover from failures.

Replacing Instances The instance replacement process merges the adding of new instances and removing the old instances (Figure 3.2): 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 require approximately

the same amount of time. The new instances may use compatible implementations with different IP addresses, passwords, port numbers, operating systems or application versions.

The roles specified in an MTD system can be implemented in numerous ways, by various applications and operating systems e.g., web load balancer – installing Varnish on an Ubuntu instance or Nginx on Fedora can achieve the same objective. As long as Puppet directives (manifests) exist, they can be included in a role implementation.

3.3 Threat Model

The threat model covers the in-scope and out-of-scope risks and also the capabilities of the MTD system to respond to the described threats.

3.3.1 In-scope Threats

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

Reconnaissance

Attackers are able to perform various reconnaissance actions (e.g., port scanning) on the public facing instances, as well as internal probing in case they can gain access to an instance on the internal network.

Arbitrary Code Execution

Attackers may also execute arbitrary code on an instance. This can occur in multiple ways. 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, arbitrary file upload/execution, SQL injection resulting in code running on the database instance, etc. Moreover, a privileged user may be utilized to execute code, e.g., a malicious former employee leaks credentials, a current user's workstation is compromised and credentials are leaked, credentials are obtained using social engineering techniques (e.g., phishing), etc. An instance can also pull and use compromised code from a source controlled by attackers, e.g., compromised packages.

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.

DDoS Attacks or General Downtime

Applications or the underlying cloud infrastructure may experience downtime due to failures or because of attackers' actions (defacement, destruction, misconfiguration, etc.). Attackers may also attempt to perform denial-of-service (DoS) or distributed-denial-of-service (DDoS) attacks by flooding the target system with a huge amount of traffic.

Side-Channel Attacks – Instance Co-Residency

Attackers may gain co-residency with victim instances (perhaps in a similar way to[100]) and use various techniques as described in[101–103] to infer sensitive information about the target (e.g., cryptographic keys).

3.3.2 Out-of-scope Threats

Our MTD systems will not provide any additional defense compared to a "static" IT system when dealing with application vulnerabilities that lead to unauthorized data retrieval and/or modification: SQL injection, cross-site scripting, cross-site request forgery (CSRF), etc.

In case the infrastructure or the MTD controller is compromised, the MTD mechanism will no longer be effective. This can happen because of misconfigurations or vulnerabilities on the MTD controller instance, hypervisor vulnerabilities in OpenStack or AWS, and so on.

3.4 Discussion - MTD System versus Threats

An MTD system will limit attackers' capabilities to perform reconnaissance actions or to pivot through the internal network assuming an instance is compromised and controlled by attackers. The system model is constantly changing, and in case the MTD system is proactively reconfiguring and/or replacing instances, various parameters must be repeatedly discovered by attackers: IP addresses, passwords, port numbers, OS/app versions, etc. Former knowledge about the system may become obsolete in a short amount of time. Lateral movement (pivoting) is also restricted. Security groups allow instances to communicate only with dependent instances on the internal network (granular rules for specific port numbers, protocols, IP addresses, and direction – ingress or egress). Security groups are automatically configured and maintained by the MTD controller. Furthermore, network packets that target IP addresses or port numbers that are not stored in the system model can be flagged as suspicious with high-confidence. Performing reconnaissance actions without being detected may be very difficult, if not impossible.

In order to restrain the effects of arbitrary code execution, the MTD system can be configured to act in proactive and/or reactive ways. Through instance replacement, the patched versions of various applications and OSs can be installed or a new implementation (different OS and application) may be chosen. Recovery from credential leaks may be done faster and with minimal or no downtime using replacement instances with new credentials. Compromised virtual machines can be replaced and persistent access interrupted (especially effective in case of attacks that succeeded because of a rare event). Using different implementations in the replacement process may prove efficient in interrupting automated attacks. In the worst case scenario it will force attackers to put more resources and time to cover different OSs and applications that implement the same role.

In case of DDoS attacks, the MTD system can be expanded by temporarily adding more instances in case the Service-Level Agreement (SLA) is violated. The deployment may also be moved to another infrastructure if the current infrastructure or deployment cannot be fixed (port the system specifications). However, synchronizing large amounts of persistent

data might prove to be a challenge. Solutions similar to IBM Aspera's[104] offerings can be leveraged for moving object storage data. On the other hand, moving block storage data is more provider specific and the effectiveness is highly dependent on the storage driver. For instance, OpenStack has more than twenty proprietary storage drivers (e.g., Netapp Unified Driver, IBM GPFS) and about four free ones (LVM, Ceph RBD, GlusterFS, and NFS)[105].

A sufficiently sized cloud infrastructure collaborated with the instance replacement process can make co-residency-based side-channel attacks against the instances belonging to an MTD system very challenging. The instance distribution across physical nodes may change when instances are replaced. Attackers might need to repeatedly re-locate the new target instance, gain co-residency, or, at least, re-infer specific parameters about the new instance in case it is placed on the same physical host.

3.5 Summary

Our vision of introducing moving-target defense at the whole IT system level depends on building a platform in which any component of the IT system can be replaced with a fresh new one. A component is a virtual machine instance or a cluster of instances. Although the approach can be applied directly to physical hosts, due to the widespread adoption of cloud technologies, we consider that the MTD system will be deployed in a cloud environment.

Our proposed MTD platform (ANCOR-MTD) is based on the ANCOR framework and enables users to perform live changes to their running IT systems in an automated and reliable fashion. Being able to reliably replace instances in a running IT deployment may have a significant impact on the entire security landscape. It has the potential to drastically increase attackers efforts and reduce their windows of opportunity at a very low cost.

Chapter 4

Feasibility and Security Analysis

This chapter aims to show that coordinated changes at the whole IT system level are possible, and that such proactive changes have security benefits. Two main aspects that reflect the practicality and effectiveness of an MTD mechanism:

1. Performance and functionality
2. Security

The main goal of this chapter is to analyze, evaluate, and quantify how the operations supported by our MTD platform affect a running IT system in terms of functionality, performance, and security.

Section 4.1 focuses on testing the performance of various real-world IT systems deployed and managed using the ANCOR-MTD platform. We quantify the performance overhead that adaptation operations impose on an IT system. Regardless of potential security benefits, an unreasonable performance overhead makes the approach infeasible.

Section 4.2 measures the security benefits of an MTD system in terms of the interruptions it creates for an attacker. Thus, we introduced the *attack window* concept that can be validated in an objective way. We are quantifying the security benefits (sizes of potential attack windows) of using the ANCOR-MTD platform in terms of cost (number of adaptations).

4.1 Feasibility Analysis

The central intent of the feasibility analysis is to evaluate how an MTD movement process affects a running cloud-based IT system in terms of functionality and performance. Despite potential security benefits, an unreasonable performance overhead would make the whole approach infeasible.

We have utilized "instance replacement" as our movement process because it is the most resource-intensive operation (including both adding and removing instances) and it also involves reconfiguration. Our efforts were primarily focused on applications. If needed, large amounts of persistent data were stored on cloud infrastructure volumes (e.g., OpenStack Cinder)[106] and re-attached to the new instances.

We tested the following **hypothesis**: The cost of proactive movement can be negligible if the movement is performed in an organized manner.

The experiments were performed on a private cloud testbed consisting of fourteen Dell PowerEdge R620 (2 x CPU@2.20 GHz, 128 GB RAM) servers and a Dell S4810P switch. We installed OpenStack (Icehouse distribution) on these machines using Mirantis' open-source tool Fuel.[107] The infrastructure consists of one controller and thirteen compute nodes.

We deployed and managed a number of IT system setups based on two different architectures: multilayered web services architecture (blogging website, eCommerce deployments, and MediaWiki with Wikipedia database dumps) and a high-performance computing architecture (Hadoop deployment).

To measure the performance, we used http-perf[108] (NPM Node HTTP Server Performance Tool) on our deployed blogging and eCommerce systems, and WikiBench[109] to measure a MediaWiki deployment utilizing a Wikipedia (wikipedia.org) database dump.

http-perf runs an HTTP client that launches HTTP requests against a server, while measuring and recording response times and other metrics. On the other hand, WikiBench is a benchmarking tool that replays real traffic traces. We ran the benchmarking tools on the initially deployed scenarios (no MTD operations interference) and established a baseline for every component in our measurements.

The baseline measurements are viewed as the **control group** in our experiments. http-perf was launched from client machines that were able to access the websites (i.e., connect to the load balancer). After establishing the baseline, we started replacing instances belonging to different roles while running the benchmarking tools. It is important to keep in mind that the way services are configured can greatly affect the performance of an IT system in general and, especially during the replacement process. In all scenarios caching features were disabled. However, the load balancer instances were configured to reload the new configuration without restarting the service. With caching enabled, requests are answered from the cache and not from the system component under test (e.g., webapp). Thus, there is little or no impact of component replacement. With caching, performance will be improved and there will be no performance degradation.

The Hadoop deployment assessment was carried out using Hibench,[110;111] a comprehensive benchmark suite for Hadoop, which consists of a set of Hadoop programs including both synthetic micro-benchmarks (e.g., Sort or WordCount) and real-world applications.

4.1.1 Blogging Website

A basic blueprint of the *blogging website* is pictured in Figure 4.1; arrows indicate dependencies between clusters of instances implementing the defined roles: `weblb`, `blogging_webapp`, and `database`.

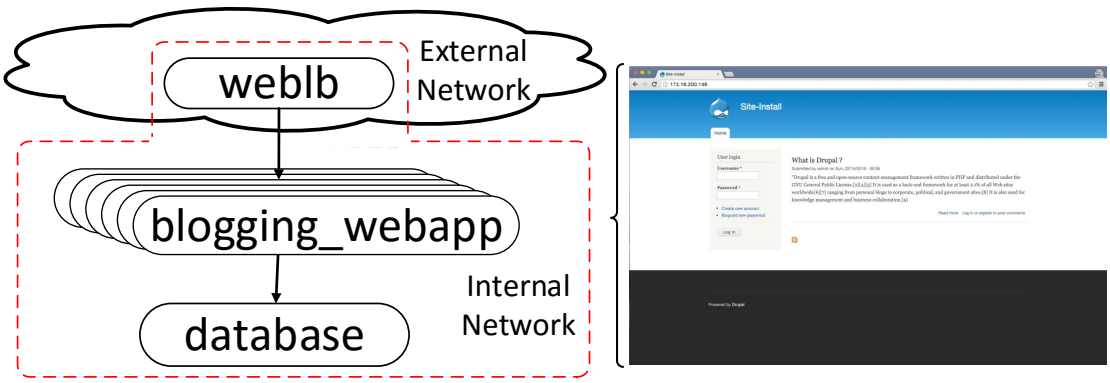


Figure 4.1: Blogging website blueprint: `blogging_webapp` implemented by a homogenous cluster of Drupal instances

Aggregated results from **20** experiment runs Each experiment run: **150,000** requests sent using **150** concurrent connections								
	Response time (sec)		Total time		Server Processing Rate (req/sec)		HTTP Error Responses	
	Avg.	stdev	Avg.	stdev	Avg.	stdev	Avg.	stdev
Baseline	0.787	0.040	13min 8sec	40 sec	190.355	10.369	0	0
Replacing one webapp	0.793	0.047	13min 13sec	47 sec	188.917	11.195	0.25	0.79
Replacing webapp cluster	0.797	0.057	13min 17sec	58 sec	188.206	12.235	13.25	37.57

Table 4.1: Drupal blogging website – performance overhead of carrying out ONE replacement operation: replacing one *webapp* instance and replacing the whole *webapp* cluster

We used a simple but common design that includes a load-balancer (Varnish or Nginx), a cluster of web applications (three instances running Drupal)[112], and a MySQL database. Based on the scenario the only component that runs in a high-availability cluster format is the web application (blogging_webapp). Therefore, we chose the web application cluster (blogging_webapp) to be the component that is changed during this experiment (Table 4.1).

All the measurements are averages of twenty experiment runs. Once we collected the baseline measurements, we started replacing instances while http-perf was running and collecting metrics (Table 4.1). The *Baseline* column captures the system's metrics without performing any replacement actions; specifically the benchmarking tool was performing read operations on the database. One hundred and fifty (150) connections was the maximum number of connections our test scenario was able to handle. A higher number of connections would overwhelm our test scenario and would cause failed requests under baseline conditions. Next we ran the replacement operations under the same http-perf load to see if and how much the replacement process delays the requests' processing (Table 4.1).

It is worth noting that the difference between the baseline and the replacement results is statistically non-significant, and that, overall, there were no or very few HTTP error responses during the replacement operations.

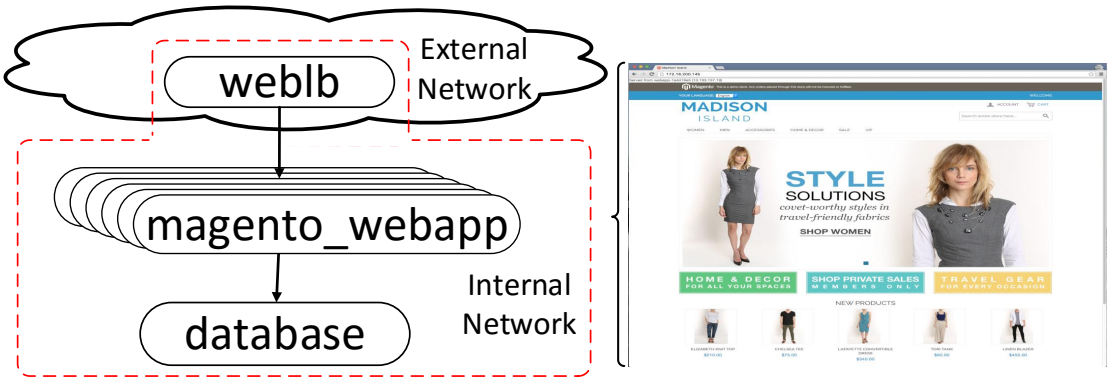


Figure 4.2: Magento: `magento_webapp` implemented by a high-availability cluster of instances running Magento CE

Aggregated results from **20** experiment runs Each experiment run: **12,000** requests sent using **120** concurrent connections								
	Response time (sec)		Total time		Server Processing Rate (req/sec)		HTTP Error Responses	
	Avg.	stdev	Avg.	stdev	Avg.	stdev	Avg.	stdev
Baseline	5.722	0.090	9min 36sec	9 sec	20.833	0.337	0	0
Replacing one webapp	5.732	0.115	9min 39sec	12 sec	20.725	0.436	1.25	3.42
Replacing webapp cluster	5.763	0.404	9min 45sec	46 sec	20.513	1.520	108.25	163.83

Table 4.2: Magento eCommerce website – performance overhead of carrying out ONE replacement operation: replacing one *webapp* instance and replacing the whole *webapp* cluster

4.1.2 eCommerce Deployments

Our second experimental scenario is an eCommerce cloud IT system. We used two different blueprints for this purpose: one based on a well-known eCommerce platform (Magento), and one based on a more scalable and highly available architecture.

Magento

Magento is a flexible, open source commerce platform that powers over 250,000 online stores worldwide.[113] The blueprint for the deployment is somewhat similar to the previous blogging website scenario. A load-balancer (Nginx) distributes the load (round-robin mode) to

the available Magento webapps (three instances) that are connected to a common MySQL database (see Figure 4.2). The database stores the sample data that is available for Magento CE (1.9.1.0-1.9.2.0)[114]. Table 4.2 pictures the results from twenty separate experiment runs. Disabling the caching features had a significant impact on the overall effectiveness of the system (only 12,000 processed requests and a very high response time), however the HTTP error responses rate was less than 1%.

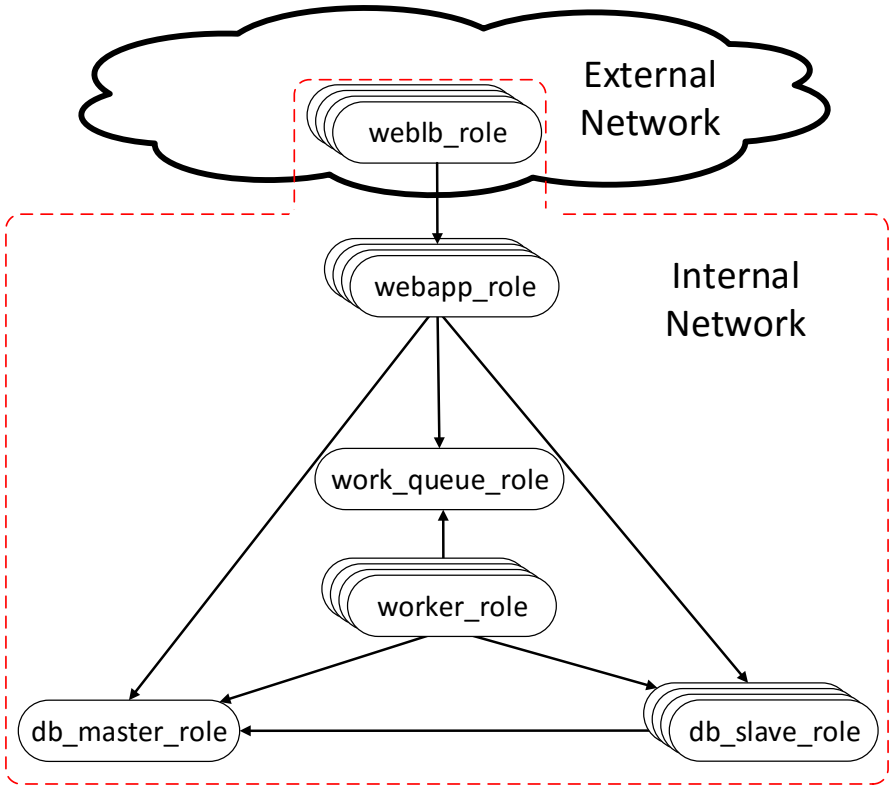


Figure 4.3: Scalable and highly available eCommerce website blueprint: `db_master, msg_queue` are single instances; `weblb, webapp, bg_worker, db_slave` are implemented by a homogeneous, high-availability cluster of instances

Scalable and Highly Available Deployment

This blueprint adopts a multilayered architecture with the various clusters of services shown in Figure 4.3: web load balancers (Nginx or Varnish), web application (Ruby on Rails with Unicorn), database (MySQL), messaging queue (Redis), and worker application (Sidekiq). Arrows indicate dependency between clusters of instances implementing a role.

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	168.919	36.924	0	0
Replacing one webapp	0.425	0.050	15min 17sec	119 sec	163.577	22.236	1.50	4.66
Replacing webapp cluster	0.424	0.047	15min 16sec	110 sec	163.755	18.887	42.60	37.57
Replacing one db_slave	0.426	0.040	15min 31sec	91 sec	161.117	16.481	588.10	62.84
Replacing db_slave cluster	0.439	0.035	15min 55sec	73 sec	157.068	12.320	913.75	113.57

Table 4.3: eCommerce website – performance overhead of carrying out ONE replacement operation: replacing one instance and replacing the whole cluster

Each cluster consists of multiple instances that offer the same services. Our test deployment consisted of two load balancers, three web applications, two database slaves, one database master, two worker instances and one messaging queue.

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 4.3) was determined by performing read operations on the eCommerce website. We focused our efforts on the web application and database clusters. Based on the scenario configuration it can be challenging and/or meaningless to measure the performance of other components. For example, only one load balancer will run-at-a-time; the backup load-balancer instance steps-in only if the main one fails. Furthermore, the worker cluster and the messaging queue will be circumvented by the web applications, in case they become unavailable (Unicorn workers will take over their functionality). Furthermore, if the master fails, a slave may be promoted to act as the database master.

As it can be observed in Table 4.3, 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. We chose this value based on the observation that one of the most-popular eCommerce platforms in the world, amazon.com,

displays a comparable number of items (database entries) on a single page every keyword search.[115] Replacing database or webapp instances can be performed in a comparable amount of time (within approximately 1 minute of the baseline measurements). Similar to the previous scenarios, we tried to assess the overall impact the instance replacement process will have when (for the testing eCommerce scenario) a very high number of requests are processed. Even though caching features were disabled in both scenarios, the rich content (mainly images) of the requested pages from the Magento scenario, contributes extensively to the major performance difference between the two eCommerce deployments.

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 the replacement and baseline measurements are, in general, statistically non-significant and the performance loss is insignificant during the replacement process (see Table 4.3). When replacing the webapp, there were very few HTTP error responses. On the other hand, when replacing the database slaves, as shown in Table 4.3, the performance is slightly impacted by this change and on average 913.75 out of 150,000 requests failed, amounting to 0.61% of the total number of requests.

4.1.3 MediaWiki with Wikipedia Database Dumps

Unlike the previous scenarios that utilized synthetic workloads, WikiBench is a web hosting benchmark that leverages actual Wikipedia database dumps, generates real traffic by replaying traces of traffic that were addressed to wikipedia.org, and targets MediaWiki,[116] the web application used to host Wikipedia. Similar to previous research, Moon et al.[117], in the area of side-channel attacks, we utilized the traces from September 2007 and the corresponding Wikipedia database dump.[118] Our setup consists of a load-balancer (Nginx), three MediaWiki backends, a common database hosting Wikipedia dumps, and one Memcached instance for synchronizing and sharing sessions among the MediaWiki backends (Figure 4.4).

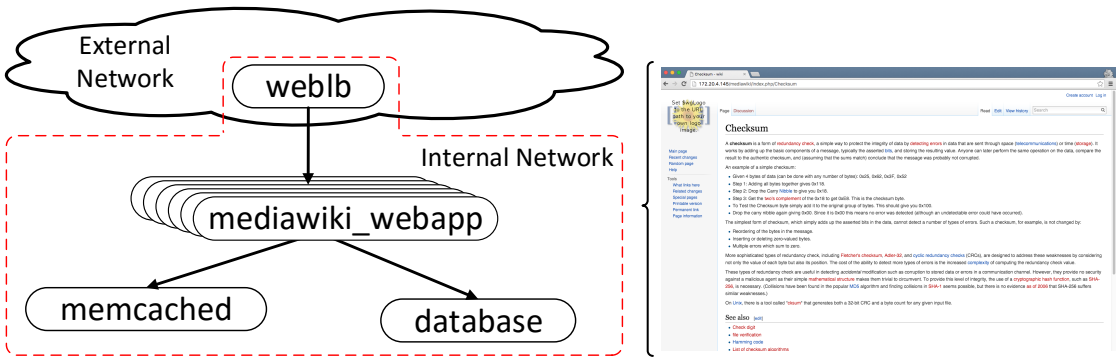


Figure 4.4: MediaWiki with Wikipedia database dump: `weblb, db_master, memcached` are single instances; `weblb, webapp, bg_worker, db_slave` are implemented by a homogeneous, high-availability cluster of instances

Aggregated results from **10** experiment runs
Each experiment run: around **4150** requests (**50** threads, **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	6.903	0.004	N/A	1.26
Replacing one webapp	0.053	0.001	10min 1sec	0.001 sec	6.905	0.006	0	1.12
Replacing webapp cluster	0.053	0.001	10min 1sec	0.001 sec	6.904	0.005	+3	1.77

Table 4.4: WikiBench (MediaWiki with Wikipedia database dumps) – average performance overhead of carrying out ONE replacement operation: replacing one *webapp* instance and replacing the whole *webapp* cluster (the results for "Replacing one webapp" exclude one outlier experiment run)

In establishing the baseline, we ran WikiBench (replaying Wikipedia traces) on the targeted MediaWiki deployment. ANCOR-MTD did not interfere in any way when performing the baseline measurements. Next, we replayed the same trace while replacing one MediaWiki instance and then the whole cluster.

We recorded the averages and standard deviations over ten different runs (see Table 4.4). The overall number of errors per se was not our main focus, we directed our attention on the difference in 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

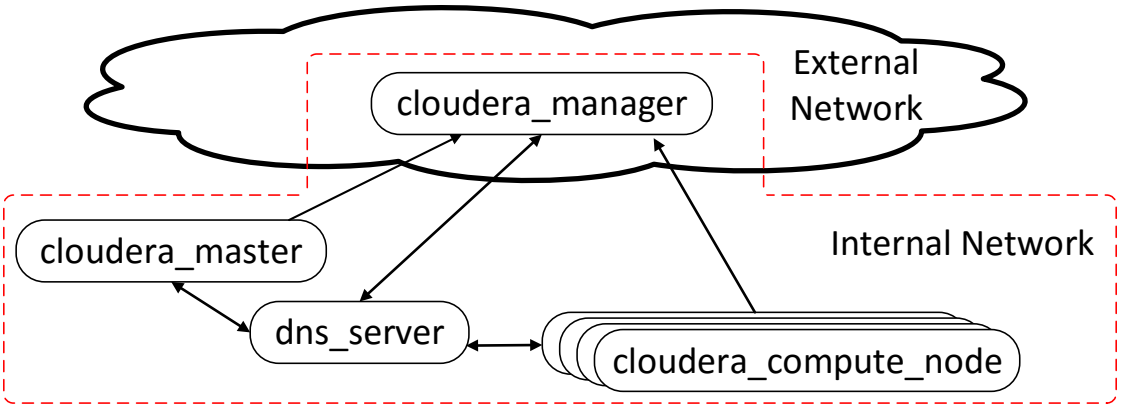


Figure 4.5: Cloudera Hadoop Deployment (CDH5): `cloudera_compute_node` is the only service implemented by a high-availability cluster of instances

small. 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 is the average over nine experiment runs. Including the outlier we would still have only 27 errors (difference compared to the baseline) with a standard deviation of 90.55 errors.

4.1.4 Hadoop Scenario

We deployed and managed a Hadoop setup using Cloudera Manager[119] on top of ANCOR-MTD. Using Cloudera Manager with our MTD platform, we have deployed the CDH5 version (Cloudera's Distribution Including Apache Hadoop version 5) of Hadoop's Distributed File System (HDFS)[120] and YARN[121] (resource negotiator). HDFS is a highly fault-tolerant distributed file system which provides high throughput access to application data and is suitable for applications that have large data sets. Moreover, it is designed to run on commodity hardware.[120] YARN is a resource negotiator responsible for managing and monitoring workloads, implementing security controls, maintaining a multi-tenant environment, and administering high availability features of Hadoop.

Our Hadoop deployment had one master (cloudera_master) and three compute nodes (cloudera_compute_node), see Figure 4.5. The dns_server and the cloudera_manager are

instances utilized to administer the actual Hadoop deployment. Compute nodes are mainly worker nodes that conduct MapReduce jobs, while the master node orchestrates the job execution and ensures that compute nodes are processing distinct parts of the data. Using the *Random Text Writer* job we have generated a total of 15 GB of data. Next we used Hibench to launch the default *Sort* job on the generated data and established a baseline measurement (Table 4.5). Since a Cloudera Hadoop deployment supports only one active master at a time, we have focused our evaluation on replacing the compute nodes: one compute node instance and the whole compute node cluster. When replacing one compute node instance, the MTD platform was instructing the Cloudera manager about the change. Therefore, the Cloudera manager was ensuring the addition of the new compute instance to the Hadoop deployment and the removal of the old instance were done in a smooth way.

We observed the following behavior: If the job started before the new (fresh) compute instance was added to the compute cluster and the old (to-be-replaced) one removed, then the old instance was assigned to participate in processing the job. When the old instance is removed, the data to be processed on it will be relocated to other nodes by cloudera_master. Thus when performing the replacement after the job has started, the "map time" and "shuffle time" (see Table 4.5) were significantly impacted because the relocation process had to complete before the old compute instance could be removed. As a result the Sort job needed a longer time to finish. Once the old instance was removed, the fresh instance did not affect the completion of the current job. In case the replacement process was finished before the job started, there was no compelling difference between the baseline measurements and replacing one compute instance (Table 4.5). We have noticed a similar behavior when replacing all the compute instances (the whole compute cluster). As observed in Table 4.5, a whole compute cluster replacement after the job started would highly impact the completion time of a running Sort job compared to the baseline measurements. However, if the replacement finished before the job execution started, the difference was not significant.

ANCOR-MTD triggered the changes in the Hadoop deployment through Cloudera in an organized way, by notifying the Cloudera manager about the changes it should perform. The manager can be configured to act instantly (similar to experiments in Table 4.5) or it can

wait for certain conditions to be fulfilled (e.g., certain jobs should complete). Therefore, the performance overhead of proactively replacing compute nodes can be negligible or non-existent if it is performed before a job starts. Otherwise the performance impact can be contained by trying to perform the changes when lower-priority jobs are running or during off-peak times.

		Average map time	Average shuffle time	Average merge time	Average reduce time	Elapsed time	State
Baseline		9sec	20sec	8sec	43sec	5min 38sec	SUCCEEDED
Replacing one compute instance	After job started	22sec	1min 2sec	3sec	32sec	8min 28sec	SUCCEEDED
	Before job started	15sec	19sec	4sec	34sec	5min 34sec	SUCCEEDED
Replacing compute cluster	After job started	53sec	1min 36sec	2sec	41sec	13min 21sec	SUCCEEDED
	Before job started	10sec	22sec	5sec	55sec	6min 3sec	SUCCEEDED

Table 4.5: Hadoop deployment – **Sort** job on **15 GB** of data

4.2 Security Analysis

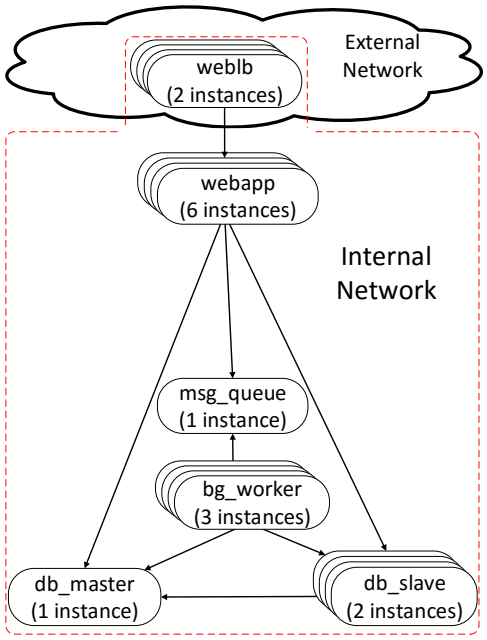
In general, quantifying the security of an IT system is a challenging task.[122] Quantifying the benefits of constantly changing a system is even more demanding.[12] While there have been numerous attempts,[32;34;35;122] the proposed security metrics are usually at a higher abstraction level that enables 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 whole IT system. We propose to measure the effectiveness of an MTD system in terms of the meaningful interruptions it creates for an attacker and the cost associated with those interruptions.

Definition 6. *An* **attack window** *is a continuous time interval an attacker may leverage without being interrupted by system changes.*

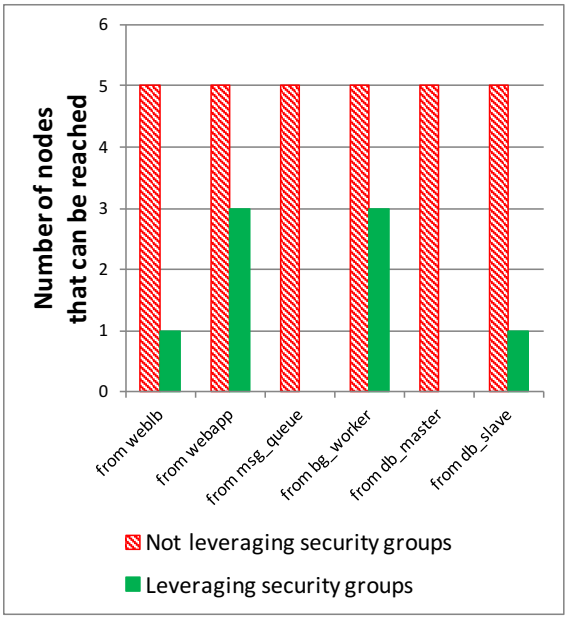
Attackers usually exploit somewhat unpredictable occurrences on targeted IT systems e.g., software bugs, misconfigurations, or user actions. Exploits and other actions may not have the same outcome every time they are executed. However, in case an exploit succeeds and the attacker compromises an instance (or a cluster of instances) on the internal network, his/her lateral movement options (pivoting) would be highly restricted due to the security groups that are automatically configured to allow ingress and egress traffic from and to the dependee and dependent instances. Moreover, traffic will be allowed only to and from the ports stored in ANCOR-MTD's operations model and attackers may follow only the dependency paths in order to possibly advance. For this reason, internal reconnaissance actions on existing or newly created instances can be detected in a straight-forward way with a high confidence (e.g., any attempt to probe a port that is not in the operations model can be flagged as suspicious).

Figure 4.6 pictures the eCommerce example scenario with and without leveraging the security groups. 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, if the `weblb` instances were compromised, an attacker would be able to reach only the six `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

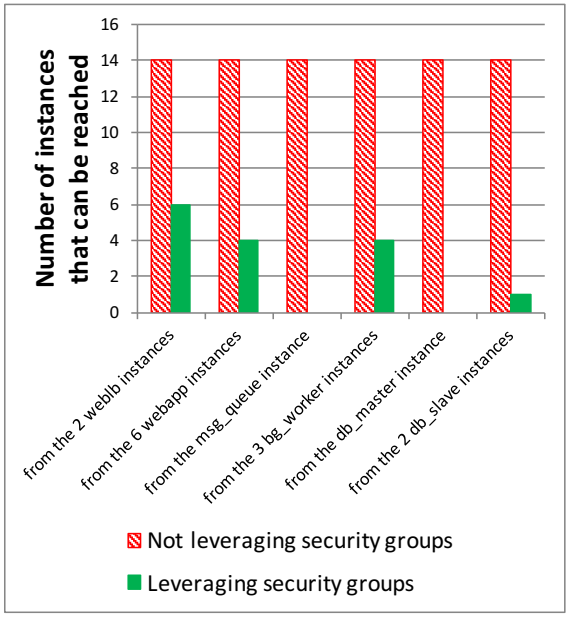
that corresponds to one instance or a high-availability cluster of instances. (Here, a *role* as presented in Chapter 2 corresponds to a node in the security analysis.)



(a) Nodes (roles) and instances



(b) Number of nodes that can be reached from each node belonging to the current eCommerce deployment



(c) Number of instances that can be reached from each instance belonging to various nodes of the current eCommerce deployment

Figure 4.6: eCommerce deployment: Internal reachability options

Controlling the attack window sizes and their distribution indirectly increases attackers' effort, reduces their window of opportunity, and can be a meaningful way to quantify the potential security benefits MTD adds to the system. One of the primary challenges is to find a balance between the budget (number and frequency of changes) and the main objective (controlling the number and sizes of attack windows).

We have defined the following terminology to describe the proposed model. An *attack attempt* is an effort to cause a breach on the targeted IT system. A *breach* can be viewed as a set of unauthorized operations (actions) an attacker is able to perform on a node belonging to the targeted IT system. An attack path may include several nodes that are part of the targeted IT system. These nodes are:

1. Transparent nodes. The load balancers (weblbs) can be considered transparent nodes if they just relay a request to an webapp instance without altering it regardless of the weblb implementation (Varnish or Nginx). Replacing or changing 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 blogging or eCommerce websites (Figures 4.1, 4.2, 4.3).

2. Stepping-stone nodes. In the eCommerce website (Figure 4.3), an attack on db_master to possibly succeed, usually, requires a vulnerable or misconfigured webapp. Changing to a different implementation with an updated application and/or configuration will most likely disrupt the ongoing attack. Thus replacing or changing a stepping-stone node on the attack path will impact an ongoing attack.

An *adaptation point* is the moment when reconfigured or new 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 size of attack windows in a certain time period (Figure 4.7 illustrates some sample inputs).

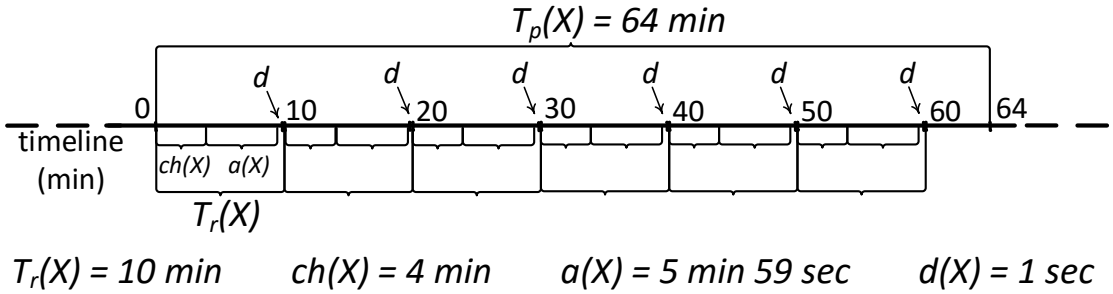


Figure 4.7: Sample inputs for node X

Definition 7. *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 8. *$T_r(X)$ is the interval between adaptation points on node X.*

We defined $T_r(X) = ch(X) + d(X) + a(X)$, where

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

$d(X)$ - 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 to change to the ready-to-use new instance(s), $a(X) \geq 0$ (e.g., specifically introduced by the user or by the adaptation strategy).

Definition 9. *We define $T_a(X)$ to be the duration of an attack attempt on node X, which is the time interval when a system should not change in order for an attack attempt to possibly succeed. In case of a successful attack attempt on a node X, then $T_a(X) < T_r(X)$.*

Definition 10. *We define $T_{target}(X)$ to be the time an attacker can spend on node X after a successful attack, $T_{target}(X) \leq T_r(X) - T_a(X)$*

Although ANCOR-MTD can provision and configure new instances in parallel, changing to the new instances belonging to dependent nodes (duration of parameter d for each node) must be completed sequentially to prevent disruption of communication between dependent services. Therefore, adaptation points (T_rs) of two dependent nodes cannot be fully aligned (coincide) because a very short delay will always be present between the two adaptation points. However, because the duration of d was approximately 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 (coincide), we consider this as only one meaningful interruption from an attacker's perspective. We assume 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. On the other hand, we view an *adaptation moment* as one or more aligned adaptation points that create a meaningful interruption.

4.2.1 Adaptation Points Placement

An attack window is a continuous time interval an attacker might be able to leverage without being interrupted by adaptation points of the targeted node or the stepping-stone nodes on the way.

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. Therefore, we start by determining the moments when adaptation points are aligned.

Intuitively, if there are no stepping-stone nodes on the way to X then the maximum attack window is equal to $T_r(X)$. Moreover, if $T_r(X)$, $T_r(Y_1)$, ..., $T_r(Y_{l-1})$ start at the same time, the maximum attack window is equal to $\texttt{min}(T_r(X), T_r(Y_1), ..., T_r(Y_{l-1}))$, and their adaptation points will be aligned (coincide) at every multiple of $\texttt{lcm}(T_r(X), T_r(Y_1), ..., T_r(Y_{l-1}))$. [1]

[1] `lcm` stands for *least common multiple* `gcd` is the *greatest common divisor*, and `min` is the *minimum*.

If they do not start at the same time, the individual starting times 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.
For this purpose let us state the following:

$$t_{min} = \texttt{min}(start\_time_{T_r(X)}, start\_time_{T_r(Y_1)}, ..., start\_time_{T_r(Y_{l-1})})$$
$$t_X = start\_time_{T_r(X)} - t_{min}$$
$$t_{Y_1} = start\_time_{T_r(Y_1)} - t_{min}$$
$$...$$
$$t_{Y_{l-1}} = start\_time_{T_r(Y_{l-1})} - t_{min}$$

The problem can be defined and solved using the *Chinese Remainder Theorem.*[123] 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 (Figure 4.8).

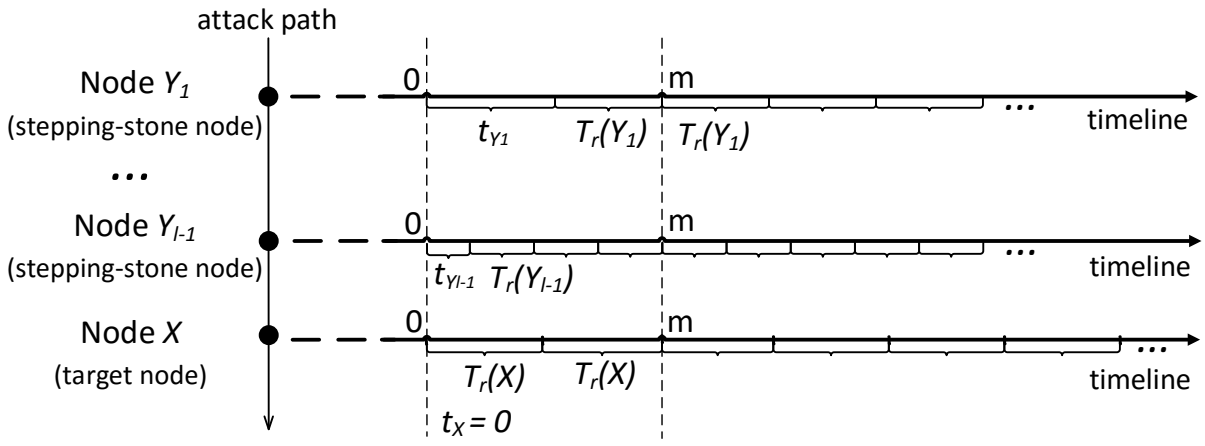


Figure 4.8: Using the Chinese Remainder Theorem to determine common adaptation points

Based on the *Chinese Remainder Theorem* we can derive the following possibilities:

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 $\texttt{lcm}(T_r(X), T_r(Y_1), ..., T_r(Y_{l-1}))$

$$\begin{cases} m \equiv t_X & \mod T_r(X) \\ m \equiv t_{Y_1} & \mod T_r(Y_1) \\ ... & \\ m \equiv t_{Y_{l-1}} & \mod T_r(Y_{l-1}) \end{cases}$$

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

Case 3

`If` $T_r(X), T_r(Y_1), ..., T_r(Y_{l-1})$ are **not** pairwise coprime (**no** coprime pairs)

`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

Appendix A includes more details for this proof.

Case 4

`If` $T_r(X), T_r(Y_1), ..., T_r(Y_{l-1})$ are **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 coincide
- Integer m does **not** exist

Appendix A contains more details for this proof.

4.2.2 Attack Windows Example

To briefly illustrate the options users have when managing their deployment using ANCOR-MTD, let us consider a possible IT system architecture as pictured in Figure 4.9. Nodes may be implemented using high-availability clusters of instances or only by single instances. Replacing one or all instances belonging to a node takes roughly the same amount of time (Section 3.2). The architecture pictured in Figure 4.9 can serve as a concrete highly-available eCommerce website (Figure 4.3).

Based on an improved version (with faster replacements) of the concrete eCommerce scenario, the replacement times for the nodes in Figure 4.9 are $T_r(B) = 10$ minutes, $T_r(F) = 11$ minutes, $T_r(E) = 11$ minutes, $T_r(A) = T_r(C) = 3$ minutes, $T_r(D) = 3$ minutes and $d(B) = d(F) = d(E) = d(A) = d(C) = d(D) = \frac{1}{60}$ minutes. 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 (see Figure 4.9). For the purpose of this example we will focus on the first path, A, B, F, E.

Node A is transparent (corresponds to the weblb in the eCommerce scenario), and thus $T_r(A)$ will not be taken into consideration. Assuming the replacement intervals 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.

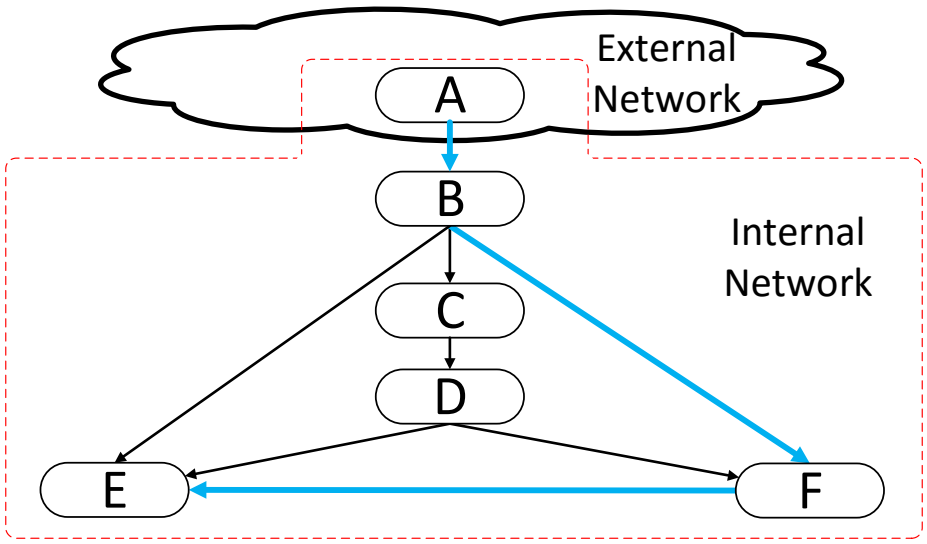


Figure 4.9: Possible IT system architecture. Arrows indicate dependencies and picture the security group configurations, light-colored arrows indicate the attack path from Section 4.2.2.

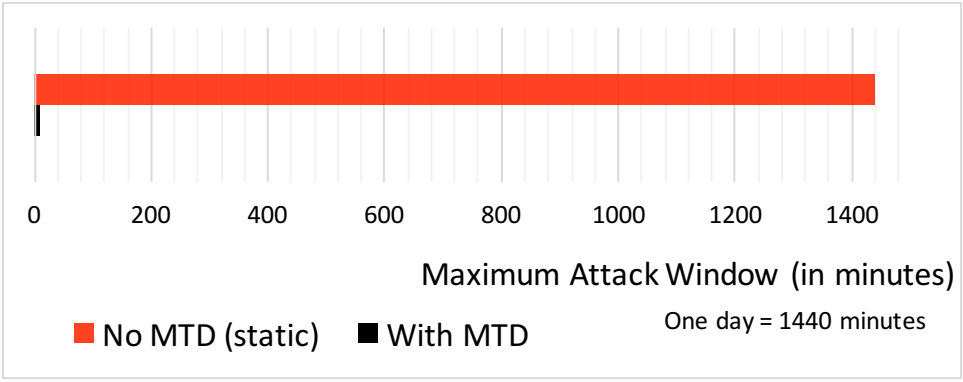


Figure 4.10: Maximum attack windows over one day

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 the maximum attack window can be as long as the whole time period (Figure 4.10).

In the default scenario where adaptations start at the same time, the adaptation points will be aligned every $\texttt{lcm}(T_r(E), T_r(B), T_r(F)) = \texttt{lcm}(10, 11, 11) = 110$ minutes. However, depending on the starting times of $T_r(B), T_r(E)$, and $T_r(F)$, the adaptation points might never coincide and the distributions of the individual attack windows may significantly vary. Figure 4.11 illustrates three possible attack windows distributions over one day (24 hours).

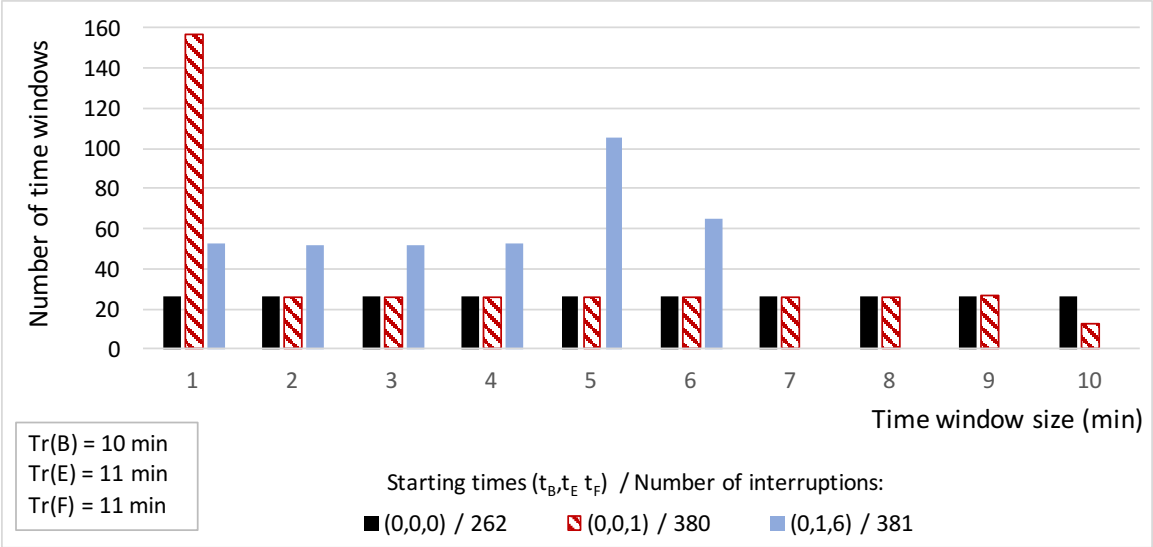


Figure 4.11: Attack windows distribution over one day with a cost of 407 adaptation moments for 262 interruptions with starting times (0,0,0), 380 interruptions with (0,0,1), and 381 interruptions with (0,1,6)

To generate these distributions 407 adaptation points are needed in each case. As observed in Figure 4.11, for the same cost, the outcome may be very different. For instance, 262 interruptions and 26 ten-minute attack windows when starting at (0,0,0) might not be the best option when a user can get 380 interruptions and fewer ten-minute windows for the same cost.

In order to increase the number of interruptions while maintaining the same cost (number of adaptations), adaptation points should not pairwise coincide. For that reason, we can opt for a set of parameters that fall under Case 3 in Section 4.2.1:

$$\begin{cases} T_r(E), T_r(B), T_r(F) \text{ - NOT pairwise coprime} \\ t_E \equiv t_B \mod \texttt{gcd}(T_r(B), T(E)) \text{ is FALSE} \\ t_E \equiv t_F \mod \texttt{gcd}(T_r(F), T(E)) \text{ is FALSE} \end{cases}$$

For instance, 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 above-stated requirements. For example, $T_r(E)$ starts first, $T_r(F)$ starts 4 minutes later, and $T_r(B)$ starts 7 minutes after $T_r(E)$.

In other words, $t_E = 0$, $t_F = 4$, and $t_B = 7$ (see Figure 4.12). Using these parameters, the adaptation points of the instances belonging to the three nodes will not coincide and the maximum attack window is only 4 minutes.

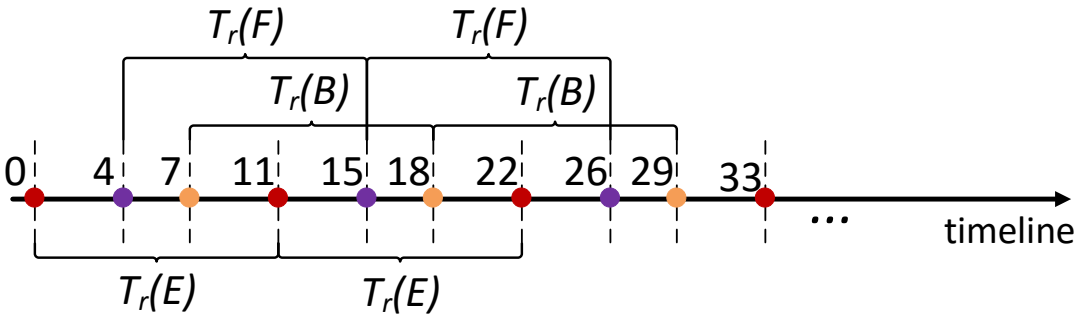


Figure 4.12: Adaptation schedule example

Figure 4.13 illustrates two more starting time options that result in the same number of interruptions, 393, for the same cost. Furthermore, we have more attack windows of the same size while the size of the maximum window is also smaller compared to Figure 4.11.

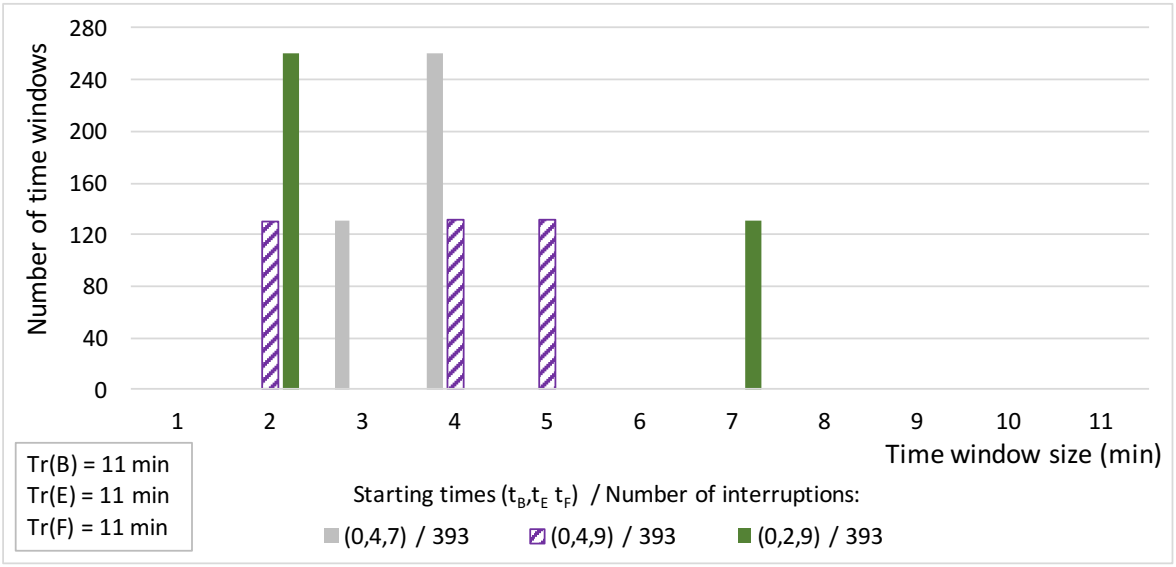


Figure 4.13: Attack windows distribution over one day when no two adaptation points coincide, with a cost of 393 adaptation moments for 393 interruptions in all three cases

In case of a successful attack, the maximum time an attacker may spend on an instance belonging to node E, $T_{\texttt{target}}(E)$, 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 to 10 minutes on an instance belonging to node E depending on the parameter choices (Figures 4.11 and 4.13).

While there are numerous options for starting times, a's, etc., a user will always be able to calculate the cost in terms of number of adaptations. In the following part of this effort we are aiming at providing an ANCOR-MTD user with some guidelines for setting up the parameters.

4.2.3 Goals versus Costs

The overall goal is to create as many meaningful interruptions for an attacker as the environment can afford. The values of the parameters used in the calculation of T_r depend on the resources a system administrator is able to manipulate for each node. As an illustration, let us consider the example described in the previous section (Section 4.2.2). The lower bound in our testing environment is 10 minutes for $T_r(B)$, and 11 minutes for $T_r(E)$ and $T_r(F)$ with $a(B) = a(E) = a(F) = 0$.

In case of node B the duration of $ch(B)$ (provisioning and configuring process) can be shortened by tuning the software applications' setups, adding more hardware to the cloud infrastructure, or by keeping a pool of pre-configured instances for every node. The same applies to nodes E and F. Pre-configured instances may be rapidly synchronized (if needed) and made available much quicker than starting the whole provisioning and configuring process from scratch. Depending on the size of the pre-configured instances pool, the overall cost can be significantly affected. In military environments the cost might be justified, however in other environments it might not be as cost-effective.

Parameter $d(B)$ is mostly dependent on the configuration management tool's settings and represents the duration to push the changes to the instances belonging to the dependent nodes. $d(B), d(E)$ and $d(F)$'s values are in the seconds range (around 1 sec for each one in

our testing scenarios using Puppet) while the other parameters are in the minutes range.

Au contraire, $a(B), a(E)$ and $a(F)$ are the parameters that can be easily manipulated in the vast majority of environments. These parameters capture the delay introduced by the system administrator/user. For example, a system administrator can always increase or decrease $T_r(B)$ by increasing or decreasing $a(B)$. While increasing $a(B)$ has no upper bound, once $a(B) = 0$, decreasing the value of $T_r(B)$ might involve a significant cost.

The cost of an adaptation point is quantified in terms of the needed environment resources and the performance overhead/degradation the environment can withstand (accept). The environment resources may include the cost for the hardware equipment, electricity consumption, and everything else needed to reach the desired values for the ch and d parameters.

As shown in Figure 4.13, for 393 adaptation points with (0,4,7) starting times, the attack windows are 3 and 4 minutes long. If we make the necessary adjustments and bring the values of $T_r(B)$, $T_r(E)$, and $T_r(F)$ down to 6 minutes with (0,2,4) starting times, it would allow us to have only 2-minute attack windows. Even though adaptation points will not pairwise coincide (fall under Case 3 from Section 4.2.1's), the cost increases to 721 adaptation points for 24 hours. Based on the measurements presented in Table 4.3, and assuming that there won't be any errors if no replacement is running, the overall performance overhead/degradation alone would increase from roughly 1.5% errors in case of 393 adaptation points to 2.75% for 721 adaptation points.

4.2.4 Configuration Guidelines

Every environment has a certain budget – restrictions regarding the number and the frequency of adaptation points. The frequency and the number of adaptation points depend directly on the size of the T_r values. Once the lower bounds or the preferred values for the adaptation intervals are known, one can determine the parameter values (starting times and user introduced-delay) along with the adaptation order by generating all potentially-interesting options.

Based on Section 4.2.1, we are proposing a few guidelines to consider when choosing/determining the parameters:

$\texttt{min\_Tr} = \texttt{min}(T_r(X), T_r(Y_1), ..., T_r(Y_{l-1}))$

- If t_X, t_{Y_1},...,$t_{Y_{l-1}} \leq$ `min_Tr` then:
  1. *max attack window* $\leq$ `min_Tr`
  2. the range of window sizes $\in$ *mod* `min_Tr`.
- Parameters that fall under Case 3 ensure that one adaptation point translates to one interruption for attackers.
- Case 3 parameters with the same T_r values for all nodes usually result in a more reduced range of various windows sizes. The sizes of the windows depend on the difference between the starting times of the adaptation intervals.
- Interchanging nodes starting times may also impact the window size distribution.
- The maximum time an attacker may spend on a target node in case of a successful attack: $T_{\texttt{target}}(X) =$ *max attack window* - $T_a(X)$.

4.2.5 Attack Attempts

We have defined an attack attempt as an uninterrupted effort to cause a breach on the targeted IT system. In general, we consider this effort to be meaningful and with a realistic chance to succeed. For example, blindly launching Linux kernel exploits against a Windows machine is not viewed as a meaningful attack attempt and might not be considered a priority. Therefore, when referring to attack attempts we are considering meaningful attack attempts.

As previously mentioned in case of the attack windows, the main objectives on an MTD system is to increase uncertainty and apparent complexity for attackers, reduce their window of opportunity and increase the costs of their attack efforts. Attackers usually exploit somewhat unpredictable occurrences on targeted IT systems and exploits may not have identical outcomes every time they are executed. Being able to determine and limit the number

of attack attempts increases attackers' efforts, helps in determining the appropriate attack window distribution, and can be a meaningful way to further quantify the potential security benefits adaptations add to the system.

Being able to calculate the attack windows distribution enables a system administrator (user) to calculate the maximum number of attack attempts on a node X within $T_p(X)$.

NO MTD. If no MTD mechanism is in place, an attacker might be able to shorten the duration of an attack attempt (T_a) by learning from previous failed attempts. Therefore, the attacker will have at least

$$\text{Attack attempts per } T_p(X) = \lfloor \frac{T_p(X)}{T_a(X)} \rfloor$$

With MTD. Assuming the following information holds and is known to the attacker:

$$\begin{cases} T_r(X) = T_r(Y_1) = ... = T_r(Y_{l-1}) < T_p(X), \\ a(X), a(Y_1), ..., a(Y_{l-1}) \text{ are known to the attacker,} \\ d(X) = d(Y_1) = ... = d(Y_{l-1}); \\ \text{while} \\ Y_1, ..., Y_{l-1} \text{ are stepping-stone nodes on the path to } X \\ l \text{ is the total number of nodes on the attack path} \\ q \text{ is the number of nodes that are aligned to create the interruption} \end{cases}$$

An attack is not disrupted when stepping-stone nodes or the target node are provisioned and configured (during $ch(X), ch(Y_1), ..., ch(Y_{l-1})$) or during the delay introduced by the user until a new instance becomes active ($a(X), a(Y_1), ..., a(Y_{l-1})$). An attack is typically interrupted when the new instance becomes active and changes are sent to the affected instances ($d(X), d(Y_1), ..., d(Y_{l-1})$).

Therefore, for each possible attack window, the number of possible attack attempts (including adjacent and stepping-stone nodes) are computed as

Attack attempts per $T_r(X) =$

$$\begin{cases} \lfloor \dfrac{T_r(X) - q * d(X)}{T_a(X)} \rfloor, T_r(X) > q * d(X) & (\textbf{Case 1}) \\ \\ 0, \quad T_r(X) \leq q * d(X) & (\textbf{Case 2}) \end{cases}$$

$T_r(X) - q * d(X)$ is the amount of time the attacker has to launch attacks on X without being disrupted during a single adaptation interval (Figure 4.14). The disruption happens when changes are pushed to X or to the stepping-stone instances on the attack path. Provisioning and configuring new instances can be performed by the MTD platform in parallel. However, changing to the new instances belonging to dependent nodes must be completed sequentially in order not to disrupt the communication between the dependent services. In our MTD system, attack paths must follow the system dependency due to the security group configuration. Therefore the time required for pushing changes to all the nodes on the attack path leading to X is $q * d(X)$, during which time an attacker will not be able to launch attacks on X.

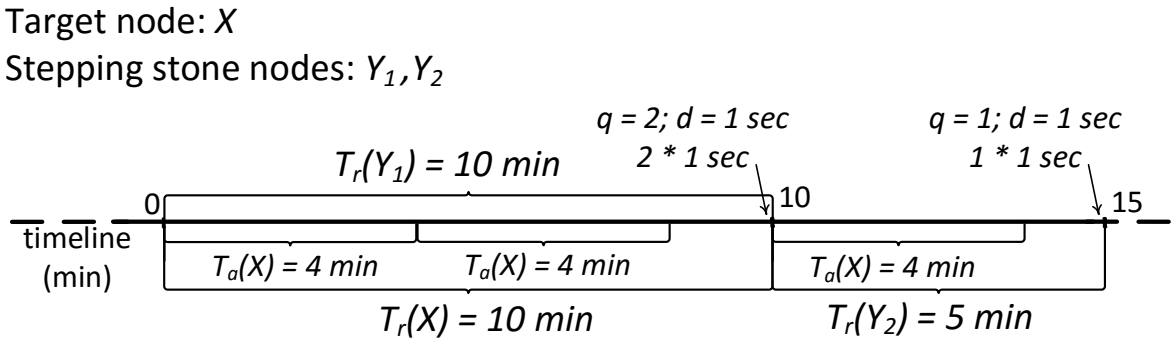


Figure 4.14: Attack attempts within adaptation windows

Thus, $\lfloor \frac{T_r(X) - q * d(X)}{T_a(X)} \rfloor$ captures the number of attack attempts on X that can be launched during one full interval between adaptations (Case 1). In case $T_r(X) \leq q * d(X)$ (Case 2), no attack attempts are allowed. Although Case 2 seams appealing, it might have a significant negative impact on the functionality and the performance of the MTD system.

Based on the eCommerce deployment example described in Section 4.2.2 (Figure 4.9), we have the following replacement times for the nodes: Tr(B) = 10 minutes, Tr(F) = 11 minutes, Tr(E) = 11 minutes, Tr(A) = Tr(C) = 3 minutes, Tr(D) = 3 minutes and d(B) = d(F) = d(E) = d(A) = d(C) = d(D) = $\frac{1}{60}$ minutes.

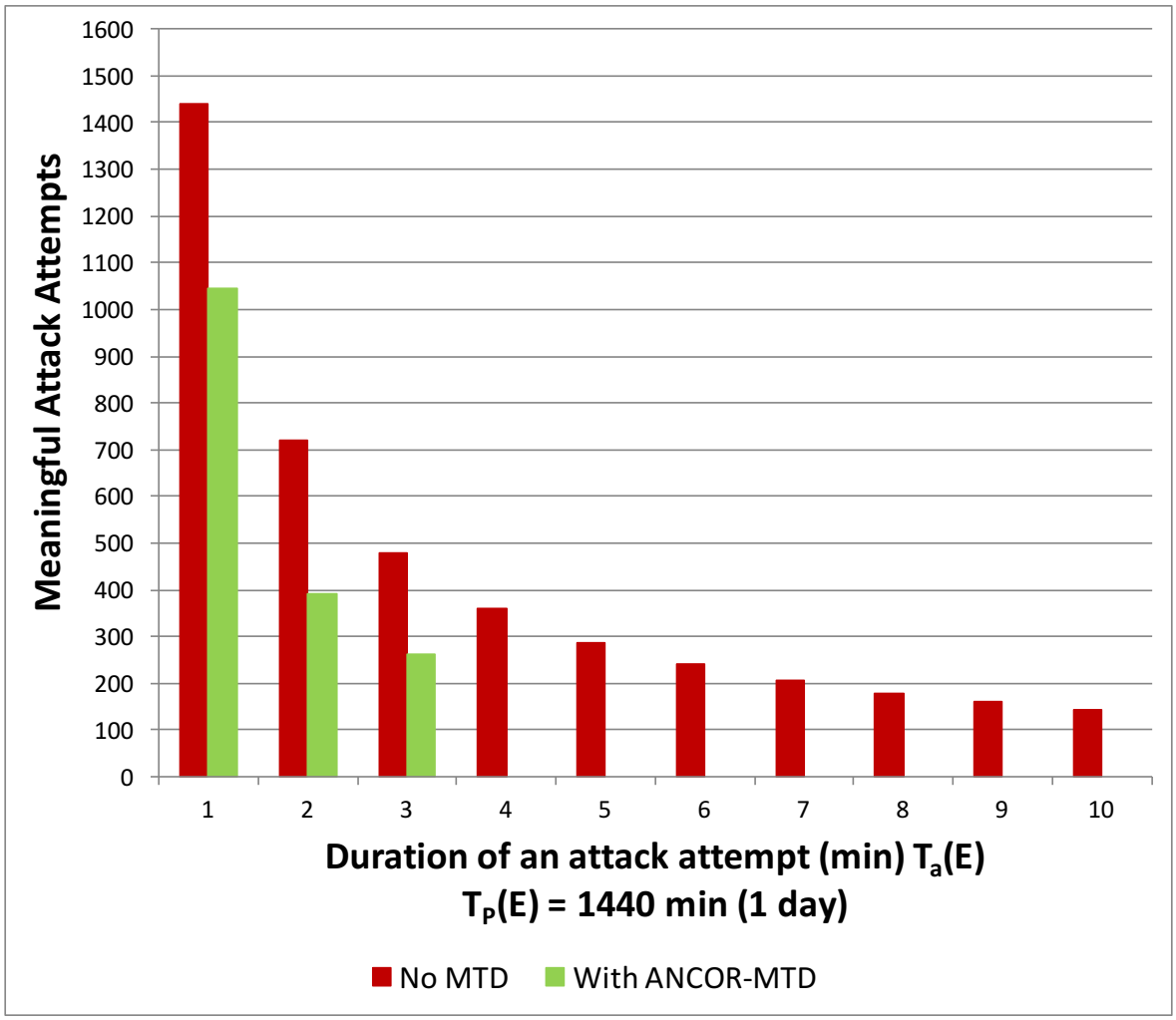


Figure 4.15: Attack attempts on node E from Figure 4.9

Similar to Section 4.2.2, we will focus on node E and the path highlighted in Figure 4.9: A, B, F, E. Based on the guidelines provided in Section 4.2.3, by setting $a(B)$ to 1 minute we have $T_r(E) = T_r(B) = T_r(F) = 11$ minutes, and opt for $t_E = 0$, $t_F = 4$, and $t_B = 7$ (Figure 4.12). Figure 4.15 shows the number of attack attempts on node E. In case of short attack attempts, adaptations help reduce the number of tries an attacker will have. However, as the duration of an attack attempt increases the effectiveness of the adaptations also increases e.g., $T_a(E) \geq 4$ minutes, all attack attempts will be interrupted at least once by the MTD system.

It is worth noting that for static systems (No MTD), the numbers are computed based on the assumption that attackers do not learn from previous attack attempts. In case of an MTD system, an attacker will be forced to re-run reconnaissance actions because new instances will be configured using different parameters (e.g., IP, port, credentials, etc.) – attackers will not be able to learn from previous attack attempts. Moreover, compromised instances will be automatically replaced with clean instances as part of the proactive replacing schema.

4.3 Discussion

In recent years, several researchers have focused on side-channel attacks[101–103;124–126] and co-locating and detecting co-resident instances in public clouds[100;127–130]. In 2015, Varadarajan et al.[128] and Xu et al.[129] concluded that determining co-residency is still possible,[131] but more challenging than just six years before[100] due to, in part, the larger instance pools (more hardware resources) in the current clouds.[129]

ANCOR-MTD provides users a control mechanism to deploy and manage their IT systems on a cloud in a automated and reliable way. Instead of relying on the cloud provider, the user controls the replacement operations and can regularly trigger physical host location "refreshes". New instances get new private IPs, application credentials and ports, and can move to new physical hosts.

The physical host where a new instance is placed depends on the cloud provider's scheduler. While public cloud scheduler rules may differ, Appendix B describes how the OpenStack Filter Scheduler[1] is configured on our cloud infrastructure.

We have deployed on our OpenStack infrastructure the highly-available and scalable eCommerce scenario (described in Section 4.1.2) with twenty web applications (webapps). Figure 4.16 illustrates the distributions of the webapp instances on the thirteen physical compute hosts when initially deployed, and after two whole-cluster webapp replacements. Despite having only thirteen compute nodes, instances "move" between nodes on every replacement operation. We noticed that between the initial deployment and the first 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.

Assigning configurable "expiration" dates to each instance will radically change the landscape against an attacker. While it might seem that more instance replacements provide more chances to time instance deployments and thus achieve co-residency (see[128]), attackers still have to determine whether the cost to achieve co-residency and perform a side-channel attack is justified against instances with limited lifetimes (perhaps only minutes). Moreover, user controlled replacement complements existing and future cloud provider mitigation techniques such as AWS VPC[132] and Nomad.[117]

Figure 4.16: Distribution of 20 webapp instances across 13 physical hosts (initial deployment and two whole-webapp-cluster replacements)

Since a replacement operation can be triggered proactively or by security events, it is difficult for attackers to determine if their activities have been discovered. When triggered by security events (e.g., IDS alerts), replacement is a low-cost way to deal with false positives as replacements do not disrupt the normal operation of the system.

The performance loss can be compared in a way to Netflix's approach to test the resiliency of their services. They deployed a service (called Chaos Monkey)[98] that seeks out clusters of services and randomly terminates instances (VMs). By frequently causing failures, Netflix forces their services to be built in a way that is more resilient. ANCOR-MTD terminates instances during movement, but in a far more organized way.

All scenarios in this work have general-purpose designs and public-facing instances known to both, benign and malicious users. In other environments (e.g., military), a hidden, possibly dynamic, instance known only to the benign users may be plausible by using approaches such as IP hopping. We used public instances because hidden interfaces on public networks are more rightly viewed security through obscurity as opposed to movement.

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. However, if persistent data must be synchronized with other locations, IBM Aspera's[104] offerings could be used to move object storage data. Moving block storage data is more provider-specific (e.g., copying AWS EBS snapshots)[133] and the effectiveness is highly dependent on the storage driver.

Attackers may be able to store backdoor information in persistent data that enables them to restore persistent access, making the replacement process less effective. Orthogonal research is needed to study the data integrity problem. Nevertheless, ANCOR-MTD addresses system-level security and is capable of dramatically reducing the threat to current enterprise network setups.

4.4 Summary

There are two main aspects that reflect the practicality and effectiveness of an MTD mechanism: (1) performance and functionality, and (2) security. This chapter analyzes and quantifies how the operations supported by our MTD platform affect a running IT system.

For this purpose, we have developed a series of real-world IT systems for the proposed ANCOR-MTD platform. ANCOR-MTD performs the movement process in a reliable way with negligible performance (statistically non-significant) overhead. To evaluate the security benefits facilitated through the platform, we analyze costs versus security benefits and demonstrate that an IT system deployed and managed using ANCOR-MTD will increase attack difficulty. We are able to quantify the outcome (sizes of potential attack windows) in terms of the cost (number of adaptations), and show that MTD systems managed and deployed using our platform will achieve the goal of increasing attack difficulty.

Chapter 5

Conclusions and Future Work

Moving Target Defenses are seen as a way to alter the security landscape by increasing the uncertainty and complexity for attackers, reducing their window of opportunity and increasing their reconnaissance costs and attack efforts.

The static nature of current IT systems gives attackers the extremely valuable advantage of time. Therefore, the Moving Target Defense approach has emerged as a potential new solution to cyber security. 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. There have been a number of MTD-related research efforts such as randomizing IP addresses, executable codes, and even machine instruction sets. These are important steps towards achieving the overall goal of moving target defense, but they focus on specific aspects of a system to apply the MTD idea e.g., IP addresses, code for specific applications, individual computer architectures. There has not been much research on how to apply the MTD idea at the whole IT system level.

A whole IT system can be described as a subset (component) 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 scope of a whole IT system are determined by the users (system engineers) and can range from a one-machine service (e.g., FTP server) to more complex deployments (e.g., multi-host eCommerce websites, Hadoop setups, wiki deployments).

Applying the MTD idea at the whole IT system level is essential for two reasons. First, system administrators fight the continual and generally losing battle of monitoring their IT systems for possible intrusions and compromises, patching potential vulnerabilities, maintaining user access lists, modifying firewall rules, etc. The complexity and ramifications of such IT systems, and especially the time required to maintain them, constitute major reasons why errors creep into system configurations and create security holes. Introducing an MTD mechanism for the whole IT system will support automation of those configuration tasks and reduce the chance for errors. Second, due to the complexity and error-proneness in configuring/maintaining a large IT system, once deployed, system administrators are generally reluctant to change the set-ups. The stagnant nature of the configuration used in the IT system gives adversaries chances to discover vulnerabilities, find opportunities to exploit them, gain/escalate privileges, and maintain persistent presence over time. Creating and employing an MTD mechanism on the whole IT system's configuration will eliminate or limit this advantage.

While it seems promising, there has been little research to show that MTD systems can work effectively at the whole system level and the security benefits can be quantified in realistic IT deployments. Nonetheless, there are a number of challenges to consider. For example, there are many configuration parameters one can change in an IT system with complex dependencies. Introducing random changes will almost certainly render the system unusable. Setting up an IT system and making it function properly is already a time-consuming and complicated job. Introducing changes proactively, if done improperly, may introduce additional complexities. Making a complex system more complex is unlikely to increase its security. Modern enterprise IT systems have numerous dependencies among services, so that changing one instance alone will almost certainly disrupt the dependent services. Changing a system while it is running inevitably introduces overhead, the amount of which must be carefully examined to determine if the benefits exceed costs. Thus a practical MTD design must simplify system configuration and maintenance, while introducing the capability of *moving*.

This dissertation presents a Moving Target Defense platform (ANCOR-MTD) at the whole IT system level based on separating user requirements from implementation details. To capture user requirements, we developed a high-level abstraction for defining IT systems (mainly cloud-based IT systems). Once users define their desired system in the specification, it is automatically compiled into a concrete cloud-based system that meets the specified user requirements. We demonstrate the practicality of this approach in the ANCOR framework, the core of the ANCOR-MTD platform.

ANCOR manages the relationships and dependencies between instances as well as instance clustering. Such management involves creating and deleting instances, adding/removing instances to/from clusters, and keeping dependent instances/clusters aware of configuration changes. The ANCOR framework simplifies network management as system dependencies are formalized and maintained in an automated fashion. The current implementation targets a cloud infrastructure (OpenStack) and leverages the Puppet configuration management tool.

Our approach of introducing moving target defense at the whole IT system level was to develop a platform where any component of the IT system can be replaced with a fresh new one. A component is a virtual machine instance or a cluster of instances. Even though the approach can be also applied directly to physical hosts, due to the wide-spread adoption of cloud technologies, we consider that the MTD system will be deployed on a cloud infrastructure. Our proposed MTD platform (ANCOR-MTD) enables users to perform live changes to their running IT systems in an automated and reliable fashion. Being able to reliably replace instances in a running IT deployment can have a significant impact on the whole security landscape.

There are two important aspects that reflect the practicality and effectiveness of an MTD mechanism: (1) performance and functionality, and (2) security. To evaluate the platform's practicality, we developed a series of experiments on multiple IT deployments: eCommerce deployments, blogging websites, Hadoop formation, and Mediawiki with Wikipedia dumps setup. ANCOR-MTD performs the movement process in a reliable fashion with negligible performance (statistically non-significant) overhead.

To evaluate the security benefits facilitated through the platform, we analyze costs versus security benefits and demonstrate that an IT system managed using ANCOR-MTD will increase attack difficulty. We are able to quantify the outcome (sizes of potential attack windows) in terms of the cost (number of adaptations), and show that MTD systems managed and deployed using ANCOR-MTD will achieve the goal of increasing attack difficulty.

One of the future research directions is to construct an adaptation strategies generator that automatically selects strategies based on the MTD systems' budget and goals. This effort may also involve developing a mechanism for incremental recompilation for structural changes and developing a general cost formula that can be customized for various environments. In addition, on the implementation side, the OpenStack AWS compatible API could be leveraged to potentially enable the use of one provider module for various cloud platforms. Future work will also extend and implement the constraint model as an independent component that may consume ANCOR-independent domain specific languages and generate various combinations of compatible software stacks.

Bibliography

[1] OpenStack Documentation – OpenStack Filter Scheduler. http://docs.openstack.org/developer/nova/filter_scheduler.html, April 2015.

[2] Department of Homeland Security (DHS) – Moving Target Defense. https://www.dhs.gov/science-and-technology/csd-mtd, April 2015.

[3] The Networking and Information Technology Research and Development (NITRD) Program – National Cyber Leap Year Summit 2009 Co-Chairs' Report. http://1.usa.gov/1GS5tv3, June 2012.

[4] S. Antonatos, P. Akritidis, E.P. Markatos, and K.G. Anagnostakis. Defending against hitlist worms using network address space randomization. In *Proceedings of the 2005 ACM workshop on Rapid malcode*, WORM '05, pages 30–40, New York, NY, USA, 2005. ACM. ISBN 1-59593-229-1.

[5] Matthew Dunlop, Stephen Groat, William Urbanski, Randy Marchany, and Joseph Tront. MT6D: a moving target ipv6 defense. In *Military Communications Conference, 2011-MILCOM 2011*, pages 1321–1326. IEEE, 2011.

[6] Stephen Groat, Matthew Dunlop, Randy Marchany, and Joseph Tront. Using dynamic addressing for a moving target defense. In *Proceedings of the 6th International Conference on Information Warfare and Security.*, page 84, 2011.

[7] Per Larsen, Stefan Brunthaler, and Michael Franz. Security through diversity: Are we there yet? *IEEE Security and Privacy*, 12(2):28–35, Mar./Apr. 2014.

[8] Daniel Williams, Wei Hu, Jack W. Davidson, Jason D. Hiser, John C. Knight, and Anh Nguyen-Tuong. Security through diversity: Leveraging virtual machine technology. *IEEE Security and Privacy*, 7(1):26–33, 2009.

[9] Stephen W Boyd, Gaurav S Kc, Michael E Locasto, Angelos D Keromytis, and Vassilis Prevelakis. On the general applicability of instruction-set randomization. *Dependable and Secure Computing, IEEE Transactions on*, 7(3):255–270, 2010.

[10] Gaurav S. Kc, Angelos D. Keromytis, and Vassilis Prevelakis. Countering code-injection attacks with instruction-set randomization. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, CCS '03, pages 272–280, New York, NY, USA, 2003. ACM. ISBN 1-58113-738-9. doi: 10.1145/948109.948146. URL http://doi.acm.org/10.1145/948109.948146.

[11] Mandiant – APT1 Report. http://intelreport.mandiant.com/Mandiant_APT1_Report.pdf, October 2014.

[12] Thomas Hobson, Hamed Okhravi, David Bigelow, Robert Rudd, and William Streilein. On the challenges of effective movement. In *Proceedings of the First ACM Workshop on Moving Target Defense*, MTD '14, pages 41–50, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3150-0. doi: 10.1145/2663474.2663480. URL http://doi.acm.org/10.1145/2663474.2663480.

[13] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010. ISSN 0001-0782. doi: 10.1145/1721654.1721672. URL http://doi.acm.org/10.1145/1721654.1721672.

[14] OpenStack – What is OpenStack? https://www.openstack.org/software/, April 2015.

[15] Amazon Web Services – AWS. https://aws.amazon.com/.

[16] M. Albanese, A. De Benedictis, S. Jajodia, and Kun Sun. A moving target defense mechanism for manets based on identity virtualization. In *Communications and Network Security (CNS), 2013 IEEE Conference on*, pages 278–286, Oct 2013. doi: 10.1109/CNS.2013.6682717.

[17] Valentina Casola, Alessandra De Benedictis, and Massimiliano Albanese. A moving target defense approach for protecting resource-constrained distributed devices. In *Information Reuse and Integration (IRI), 2013 IEEE 14th International Conference on*, pages 22–29, Aug 2013. doi: 10.1109/IRI.2013.6642449.

[18] Chongkyung Kil, Jinsuk Jun, Christopher Bookholt, Jun Xu, and Peng Ning. Address space layout permutation (aslp): Towards fine-grained randomization of commodity software. In *Computer Security Applications Conference, 2006. ACSAC '06. 22nd Annual*, pages 339–348, Dec 2006. doi: 10.1109/ACSAC.2006.9.

[19] Team PaX – PaX Address Space Layout Randomization (ASLR). https://pax.grsecurity.net/docs/aslr.txt, August 2015.

[20] Mihai Christodorescu, Matthew Fredrikson, Somesh Jha, and Jonathon Giffin. *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, chapter End-to-End Software Diversification of Internet Services, pages 117–130. Springer New York, New York, NY, 2011. ISBN 978-1-4614-0977-9. doi: 10.1007/978-1-4614-0977-9_7. URL http://dx.doi.org/10.1007/978-1-4614-0977-9_7.

[21] Shardul Vikram, Chao Yang, and Guofei Gu. Nomad: Towards non-intrusive moving-target defense against web bots. In *Communications and Network Security (CNS), 2013 IEEE Conference on*, pages 55–63, Oct 2013. doi: 10.1109/CNS.2013.6682692.

[22] Stephen W. Boyd and Angelos D. Keromytis. *Applied Cryptography and Network Security: Second International Conference, ACNS 2004, Yellow Mountain, China, June 8-11, 2004. Proceedings*, chapter SQLrand: Preventing SQL Injection Attacks, pages 292–302. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. ISBN 978-3-540-24852-1. doi: 10.1007/978-3-540-24852-1_21. URL http://dx.doi.org/10.1007/978-3-540-24852-1_21.

[23] Angelos D Keromytis, Roxana Geambasu, Simha Sethumadhavan, Salvatore J Stolfo, Junfeng Yang, Azzedine Benameur, Marc Dacier, Matthew Elder, Darrell Kienzle, and

Angelos Stavrou. The meerkats cloud security architecture. In *Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on*, pages 446–450, June 2012. doi: 10.1109/ICDCSW.2012.42.

[24] Georgios Portokalidis and Angelos D. Keromytis. *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, chapter Global ISR: Toward a Comprehensive Defense Against Unauthorized Code Execution, pages 49–76. Springer New York, New York, NY, 2011. ISBN 978-1-4614-0977-9. doi: 10.1007/978-1-4614-0977-9_3. URL http://dx.doi.org/10.1007/978-1-4614-0977-9_3.

[25] Yih Huang, David Arsenault, and Arun Sood. Closing cluster attack windows through server redundancy and rotations. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, volume 2, pages 12 pp.–21, May 2006. doi: 10.1109/CCGRID.2006.1630916.

[26] Quyen Nguyen and Arun Sood. Designing scit architecture pattern in a cloud-based environment. In *Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on*, pages 123–128, June 2011. doi: 10.1109/DSNW.2011.5958797.

[27] Sanjai Narain, Gary Levin, Sharad Malik, and Vikram Kaul. Declarative infrastructure configuration synthesis and debugging. *Journal of Network and Systems Management*, 16(3):235–258, 2008. ISSN 1573-7705. doi: 10.1007/s10922-008-9108-y. URL http://dx.doi.org/10.1007/s10922-008-9108-y.

[28] Sanjai Narain, Sharad Malik, and Ehab Al-Shaer. Towards eliminating configuration errors in cyber infrastructure. In *Configuration Analytics and Automation (SAFECONFIG), 2011 4th Symposium on*, pages 1–2, Oct 2011. doi: 10.1109/SafeConfig.2011.6111678.

[29] Sanjai Narain, Dana CheeBrian Coan, Ben Falchuk, Samuel Gordon, Jaewon Kang, Jonathan Kirsch, Aditya Naidu, Kaustubh Sinkar, Simon Tsang, Sharad Malik,

Shuyuan Zhang, Vahid Rajabian-Schwart, and Walt Tirenin. A science of network con guration. *Journal of Cyber Security and Information Systems*, 4:18–31, 2016. URL https://www.csiac.org/wp-content/uploads/2016/03/AFRL_Special_Final.pdf.

[30] Sanjai Narain, Dana Chee, and Sharad Malik. Demonstrating Assured and Dynamic Configuration Over a Live, Emulated Network. http://www.argreenhouse.com/papers/narain/ADC-Live-Demo.pdf, 2011.

[31] Ehab Al-Shaer. *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, chapter Toward Network Configuration Randomization for Moving Target Defense, pages 153–159. Springer New York, New York, NY, 2011. ISBN 978-1-4614-0977-9. doi: 10.1007/978-1-4614-0977-9_9. URL http://dx.doi.org/10.1007/978-1-4614-0977-9_9.

[32] Hamed Okhravi, James Riordan, and Kevin Carter. *Research in Attacks, Intrusions and Defenses: 17th International Symposium, RAID 2014, Gothenburg, Sweden, September 17-19, 2014. Proceedings*, chapter Quantitative Evaluation of Dynamic Platform Techniques as a Defensive Mechanism, pages 405–425. Springer International Publishing, Cham, 2014. ISBN 978-3-319-11379-1. doi: 10.1007/978-3-319-11379-1_20. URL http://dx.doi.org/10.1007/978-3-319-11379-1_20.

[33] Rui Zhuang, Scott A. DeLoach, and Xinming Ou. A model for analyzing the effect of moving target defenses on enterprise networks. In *Proceedings of the 9th Annual Cyber and Information Security Research Conference*, CISR '14, pages 73–76, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2812-8. doi: 10.1145/2602087.2602088. URL http://doi.acm.org/10.1145/2602087.2602088.

[34] Benjamin D. Rodes, John C. Knight, and Kimberly S. Wasson. A security metric based on security arguments. In *Proceedings of the 5th International Workshop on Emerging Trends in Software Metrics*, WETSoM 2014, pages 66–72, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2854-8. doi: 10.1145/2593868.2593880. URL http://doi.acm.org/10.1145/2593868.2593880.

[35] George Cybenko and Jeff Hughes. No free lunch in cyber security. In *Proceedings of the First ACM Workshop on Moving Target Defense*, MTD '14, pages 1–12, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3150-0. doi: 10.1145/2663474.2663475. URL http://doi.acm.org/10.1145/2663474.2663475.

[36] ServiceNow white paper – Managing the Cloud as an Incremental Step Forward. http://www.techrepublic.com/resource-library/whitepapers/managing-the-cloud-as-an-incremental-step-forward/, August 2013.

[37] Marco Balduzzi, Jonas Zaddach, Davide Balzarotti, Engin Kirda, and Sergio Loureiro. A security analysis of amazon's elastic compute cloud service. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 1427–1434, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-0857-1. doi: 10.1145/2245276.2232005. URL http://doi.acm.org/10.1145/2245276.2232005.

[38] Sven Bugiel, Stefan Nürnberger, Thomas Pöppelmann, Ahmad-Reza Sadeghi, and Thomas Schneider. Amazonia: When elasticity snaps back. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 389–400, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0948-6. doi: 10.1145/2046707.2046753. URL http://doi.acm.org/10.1145/2046707.2046753.

[39] Puppet Labs - What is Puppet? https://puppetlabs.com/puppet/what-is-puppet, January 2014.

[40] VMware – Virtualization Solutions. https://www.vmware.com/, February 2016.

[41] Metal as a Service – MAAS. http://maas.io/, February 2015.

[42] Ansible – Automation for Everyone. https://www.ansible.com/, December 2015.

[43] Chef – Devops Delivered. https://www.chef.io/, May 2015.

[44] Ian Unruh, Alexandru G. Bardas, Rui Zhuang, Xinming Ou, and Scott A. DeLoach. Compiling abstract specifications into concrete systems: Bringing order to the cloud.

In *Proceedings of the 28th USENIX Conference on Large Installation System Administration*, LISA'14, pages 17–33, Berkeley, CA, USA, 2014. USENIX Association. ISBN 978-1-931971-17-1. URL http://dl.acm.org/citation.cfm?id=2717491.2717493.

[45] RightScale white paper – Quantifying the Benefits of the RightScale Cloud Management Platform. http://www.rightscale.com/info_center/white-papers/RightScale-Quantifying-The-Benefits.pdf, August 2013.

[46] Amazon Web Services – OpsWorks . http://aws.amazon.com/opsworks/, April 2014.

[47] Ubuntu Juju – Juju. https://juju.ubuntu.com/, April 2015.

[48] Microsoft Azure – Azure Service Definition Schema (.csdef). http://msdn.microsoft.com/en-us/library/windowsazure/ee758711.aspx, March 2014.

[49] Google Developers – Configuring with app.yaml. https://developers.google.com/appengine/docs/python/config/appconfig, December 2013.

[50] Github – Maestro. https://github.com/toscanini/maestro, January 2014.

[51] Github – MaestroNG. https://github.com/signalfuse/maestro-ng, April 2014.

[52] Deis – What is Deis? http://deis.io/overview/, March 2014.

[53] Flynn – Say hello to Flynn. https://flynn.io/, April 2014.

[54] Docker – What is Docker? https://www.docker.com/what-docker, January 2014.

[55] Puppet Labs – External Node Classifiers. http://docs.puppetlabs.com/guides/external_nodes.html, January 2014.

[56] PuppetLabs docs – Hiera 1: Overview. http://docs.puppetlabs.com/hiera/1/, February 2014.

[57] Redis – Introduction to Redis. http://redis.io/topics/introduction, April 2015.

[58] SaltStack docs – SaltStack. https://docs.saltstack.com/en/latest/, April 2014.

[59] BCFG2 Website - What is Bcfg2? http://docs.bcfg2.org/, April 2014.

[60] CFEngine – What is CFEngine? https://cfengine.com/learn/what-is-cfengine/, March 2015.

[61] Varnish Cache – About. https://www.varnish-cache.org/about, April 2015.

[62] Ubuntu. http://www.ubuntu.com/, March 2016.

[63] Nginx – nginx news. http://nginx.org/, April 2015.

[64] Fedora. https://getfedora.org/, March 2016.

[65] Clark C. Evans – YAML. http://yaml.org/, September 2013.

[66] Ruby – Ruby is. https://www.ruby-lang.org/en/, April 2014.

[67] RubyGems – Unicorn. https://rubygems.org/gems/unicorn, April 2015.

[68] MySQL – The world's most popular open source database. https://www.mysql.com/, September 2014.

[69] Github – Sidekiq. https://github.com/mperham/sidekiq, March 2015.

[70] Craig Dunn – Designing Puppet: Roles and Profiles. http://www.craigdunn.org/2012/05/239/, October 2013.

[71] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2012. ISBN 0262017156, 9780262017152.

[72] Alloy – a language & tool for relational models. http://alloy.mit.edu/, October 2015.

[73] Ruby docs – Hash. http://ruby-doc.org/core-2.2.0/Hash.html, April 2016.

[74] Python docs – Dictionaries. https://docs.python.org/2/tutorial/datastructures.html#dictionaries, April 2016.

[75] Ruby docs – YAML. http://ruby-doc.org/stdlib-2.1.0/libdoc/yaml/rdoc/YAML.html, February 2016.

[76] Roland Koebler – Template Engine. http://www.simple-is-better.org/template/, March 2016.

[77] Kuwata-lab.com – Erubis. http://www.kuwata-lab.com/erubis/, February 2016.

[78] Puppet Labs – What is MCollective and what does it allow you to do? http://puppetlabs.com/mcollective, January 2014.

[79] Sensu – What is Sensu? http://sensuapp.org/, April 2014.

[80] OpsView – Let's make complex, simple. http://www.opsview.com/, April 2014.

[81] Fog – The Ruby cloud services library. http://fog.io/, January 2014.

[82] Roy T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[83] CloudVelocity Software. One hybrid cloud software. Technical report, 2013.

[84] William Fellows – 'Run any app on any cloud' is CliQr's bold claim. http://cdn1.hubspot.com/hub/194983/Run_any_app_on_any_cloud.pdf, January 2013.

[85] Amazon Web Services – AWS CloudFormation. https://aws.amazon.com/cloudformation/, April 2015.

[86] OpenStack Wiki – Heat. https://wiki.openstack.org/wiki/Heat, April 2015.

[87] Terraform – Introduction to Terraform. http://www.terraform.io/intro/index.html, September 2014.

[88] Github – Heat Wordpress Template. https://github.com/openstack/heat-templates/blob/master/cfn/F17/WordPress_With_LB.template, September 2015.

[89] Github – Heat MySQL Template. https://github.com/openstack/heat-templates/blob/master/cfn/F17/MySQL_Single_Instance.template, September 2015.

[90] Terraform – Terraform vs. CloudFormation, Heat, *etc.* http://www.terraform.io/intro/vs/cloudformation.html, September 2014.

[91] Steven J. Vaughan-Nichols – Canonical Juju DevOps Tool Coming to CentOS and Windows. http://www.zdnet.com/canonical-juju-devops-tool-coming-to-centos-and-windows-7000029418/, October 2014.

[92] Constantine Sapuntzakis, David Brumley, Ramesh Chandra, Nickolai Zeldovich, Jim Chow, Monica S. Lam, and Mendel Rosenblum. Virtual appliances for deploying and maintaining software. In *Proceedings of the 17th USENIX Conference on System Administration*, LISA '03, pages 181–194, Berkeley, CA, USA, 2003. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1051937.1051965.

[93] Kyrre Begnum. Managing large networks of virtual machines. In *Proceedings of the 20th Conference on Large Installation System Administration*, LISA '06, pages 16–16, Berkeley, CA, USA, 2006. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1267793.1267809.

[94] Jeannie Albrecht, Christopher Tuttle, Ryan Braud, Darren Dao, Nikolay Topilski, Alex C. Snoeren, and Amin Vahdat. Distributed application configuration, management, and visualization with plush. *ACM Trans. Internet Technol.*, 11(2):6:1–6:41, December 2011. ISSN 1533-5399. doi: 10.1145/2049656.2049658. URL http://doi.acm.org/10.1145/2049656.2049658.

[95] Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, and David Walker. Composing software-defined networks. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, pages 1–14,

Berkeley, CA, USA, 2013. USENIX Association. URL http://dl.acm.org/citation.cfm?id=2482626.2482629.

[96] Hao Qian and Daniel Andresen. Reducing mobile device energy consumption with computation offloading. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2015 16th IEEE/ACIS International Conference on*, pages 1–8, June 2015. doi: 10.1109/SNPD.2015.7176219.

[97] Security Week – Prioritizing Patch Management Critical to Security. http://www.securityweek.com/prioritizing-patch-management-critical-security, March 2016.

[98] Netflix – Chaos Monkey Released Into The Wild. http://techblog.netflix.com/2012/07/chaos-monkey-released-into-wild.html, January 2014.

[99] OpenStack Docs – Security Groups. http://docs.openstack.org/openstack-ops/content/security_groups.html, February 2014.

[100] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 199–212, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-894-0. doi: 10.1145/1653662.1653687. URL http://doi.acm.org/10.1145/1653662.1653687.

[101] David Gullasch, Endre Bangerter, and Stephan Krenn. Cache games – bringing access-based cache attacks on aes to practice. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, SP '11, pages 490–505, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-0-7695-4402-1. doi: 10.1109/SP.2011.22. URL http://dx.doi.org/10.1109/SP.2011.22.

[102] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Cross-vm side channels and their use to extract private keys. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 305–316, New York,

NY, USA, 2012. ACM. ISBN 978-1-4503-1651-4. doi: 10.1145/2382196.2382230. URL http://doi.acm.org/10.1145/2382196.2382230.

[103] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Cross-tenant side-channel attacks in paas clouds. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 990–1003, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2957-6. doi: 10.1145/2660267.2660356. URL http://doi.acm.org/10.1145/2660267.2660356.

[104] Aspera white paper – Aspera Direct-to-Cloud Storage. http://asperasoft.com/fileadmin/media/Asperasoft.com/Resources/White_Papers/AsperaWP_Direct_to_cloud.pdf, March 2015.

[105] OpenStack Marketplace – OpenStack Drivers. https://www.openstack.org/marketplace/drivers/, July 2015.

[106] OpenStack Wiki – Cinder. https://wiki.openstack.org/wiki/Cinder, May 2015.

[107] Mirantis – Mirantis OpenStack/Fuel. http://software.mirantis.com/, April 2015.

[108] http-perf – Node HTTP Server Performance Tool accessed. https://www.npmjs.com/package/http-perf, April 2015.

[109] Wikibench – The realistic Web hosting benchmark. http://www.wikibench.eu/, November 2015.

[110] Github – HiBench (version 2.2). https://github.com/intel-hadoop/HiBench/tree/MRv2, April 2015.

[111] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pages 41–51, March 2010. doi: 10.1109/ICDEW.2010.5452747.

[112] Drupal – About. https://www.drupal.org/, April 2015.

[113] Magento – Magento Community Edition. https://magento.com/products/community-edition, November 2015.

[114] Magento – Installing Sample Data for Magento Community Edition (CE). http://devdocs.magento.com/guides/m1x/ce18-ee113/ht_magento-ce-sample.data.html, November 2015.

[115] Alexa – How popular is amazon.com? http://www.alexa.com/siteinfo/amazon.com, February 2016.

[116] Mediawiki – Welcome to MediaWiki.org. https://www.mediawiki.org/wiki/MediaWiki, November 2015.

[117] Soo-Jin Moon, Vyas Sekar, and Michael K. Reiter. Nomad: Mitigating arbitrary cloud side channels via provider-assisted migration. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1595–1606, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3832-5. doi: 10.1145/2810103.2813706. URL http://doi.acm.org/10.1145/2810103.2813706.

[118] Archive.org – Wikimedia database dump of the English Wikipedia on January 03, 2008. https://archive.org/details/enwiki-20080103, November 2015.

[119] Cloudera – Local, On Premise, or Cloud-based Apache Hadoop Management. https://www.cloudera.com/downloads.html, April 2015.

[120] Hadoop – HDFS Architecture Guide. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html, April 2015.

[121] Hadoop – Apache Hadoop YARN. https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html, April 2015.

[122] Jeff Hughes and George Cybenko. Quantitative metrics and risk assessment: The three tenets model of cybersecurity. In *Technology Innovation Management Review*, 2013. URL http://timreview.ca/article/712.

[123] Stanford Crypto – The Chinese Remainder Theory. http://stanford.io/20kH3JW, January 2016.

[124] Mehmet Sinan İnci, Berk Gülmezoğlu, Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. Seriously, get off my cloud! Cross-VM RSA Key Recovery in a Public Cloud. In *IACR Cryptology ePrint Archive*, Sept. 2015.

[125] Yunjing Xu, Michael Bailey, Farnam Jahanian, Kaustubh Joshi, Matti Hiltunen, and Richard Schlichting. An exploration of l2 cache covert channels in virtualized environments. In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, pages 29–40, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1004-8. doi: 10.1145/2046660.2046670. URL http://doi.acm.org/10.1145/2046660.2046670.

[126] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-channel Attack. In *USENIX Security Symposium*, SEC'14, pages 719–732, Berkeley, CA, USA, 2014. USENIX Association. ISBN 978-1-931971-15-7. URL http://dl.acm.org/citation.cfm?id=2671225.2671271.

[127] Adam Bates, Benjamin Mood, Joe Pletcher, Hannah Pruse, Masoud Valafar, and Kevin Butler. Detecting co-residency with active traffic analysis techniques. In *Proceedings of the 2012 ACM Workshop on Cloud Computing Security Workshop*, CCSW '12, pages 1–12, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1665-1. doi: 10.1145/2381913.2381915. URL http://doi.acm.org/10.1145/2381913.2381915.

[128] Venkatanathan Varadarajan, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. A placement vulnerability study in multi-tenant public clouds. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, pages 913–928, Berkeley, CA, USA, 2015. USENIX Association. ISBN 978-1-931971-232. URL http://dl.acm.org/citation.cfm?id=2831143.2831201.

[129] Zhang Xu, Haining Wang, and Zhenyu Wu. A measurement study on co-residence

threat inside the cloud. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, pages 929–944, Berkeley, CA, USA, 2015. USENIX Association. ISBN 978-1-931971-232. URL http://dl.acm.org/citation.cfm?id=2831143.2831202.

[130] Yinqian Zhang, Ari Juels, Alina Oprea, and Michael K. Reiter. Homealone: Co-residency detection in the cloud via side-channel analysis. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, SP '11, pages 313–328, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-0-7695-4402-1. doi: 10.1109/SP.2011.31. URL http://dx.doi.org/10.1109/SP.2011.31.

[131] Zhenyu Wu, Zhang Xu, and Haining Wang. Whispers in the hyper-space: High-speed covert channel attacks in the cloud. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 9–9, Berkeley, CA, USA, 2012. USENIX Association. URL http://dl.acm.org/citation.cfm?id=2362793.2362802.

[132] Amazon Web Services – Amazon Virtual Private Cloud (VPC). https://aws.amazon.com/vpc/, October 2015.

[133] Amazon Web Services – Copying an Amazon EBS Snapshot. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-copy-snapshot.html, July 2016.

Appendix A

Additional Proofs

X is the target node while $\{Y_1, ..., Y_{l-1}\}$ are the stepping-stone nodes on the way to X. Assuming $A, B \in \{X, Y_1, ..., Y_{l-1}\}$ AND $A \neq B$ then

$t_{min} = \texttt{min}(start\_time_{T_r(A)}, start\_time_{T_r(B)})$

$t_A = start\_time_{T_r(A)} - t_{min}$

$t_B = start\_time_{T_r(B)} - t_{min}$

$$\begin{cases} T_r(A) \text{ and } T_r(B) \text{ are NOT pairwise coprime (1)} \\ t_A \equiv t_B \mod \gcd(T_r(A), T(B)) \text{ is FALSE} \quad \text{(2)} \end{cases}$$

(1)$\Rightarrow \gcd(T_r(A), T(B)) = z,\ z \neq 1$

In other words

$T_r(A) = z \times s_1,\ s_1$ is an Integer

$T_r(B) = z \times s_2,\ s_1$ is an Integer

(2)$\Rightarrow t_A \neq t_B$

If m would exist then

Assuming $T_r(A)$ starts before $T_r(B)$, then

$t_{min} = start\_time_{T_r(A)}$

$t_A = 0$

$t_B = start\_time_{T_r(B)} - start\_time_{T_r(A)}$

Furthermore

$$\begin{cases} m = t_B + q \times T_r(B) \ \text{, } q \texttt{ is an Integer} \\ m = p \times T_r(A) \ \text{, } p \texttt{ is an Integer} \end{cases}$$

$\Rightarrow p \times T_r(A) = t_B + q \times T_r(B)$

$\Leftrightarrow t_B = q \times T_r(B) - p \times T_r(A)$

$\Leftrightarrow t_B = q \times z \times s_2 - p \times z \times s_1$

$\Leftrightarrow t_B = z \times (q \times s_2 - p \times s_1)$

$\Rightarrow t_B \texttt{ is divisible by } z$

$\Rightarrow t_B \texttt{ is divisible by } \gcd(T_r(A), T_r(B))$

$\Rightarrow 0 \equiv t_B \mod \gcd(T_r(A), T_r(B)) \texttt{ is TRUE}$

$\Rightarrow t_A \equiv t_B \mod \gcd(T_r(A), T_r(B)) \texttt{ is TRUE}$

which CONDRADICTS the initial assumption `(2)`

If $T_r(B)$ starts before $T_r(A)$ then the proof is symmetric to the one presented above.

Assuming $A, B, C, D \in \{X, Y_1, ..., Y_{l-1}\}$

$$\begin{cases} T_r(A), T_r(B), T_r(C), T_r(D) \texttt{- NOT pairwise coprime (3)} \\ t_A \equiv t_B \mod \gcd(T_r(A), T(B)) \texttt{ is FALSE} \quad \texttt{(4)} \\ t_C \equiv t_D \mod \gcd(T_r(C), T(D)) \texttt{ is TRUE} \quad \texttt{(5)} \end{cases}$$

If m **would exist then** based on the proof above:

$$\begin{cases} t_A \equiv t_B \mod \gcd(T_r(A), T(B)) \texttt{ is TRUE} \\ t_C \equiv t_D \mod \gcd(T_r(C), T(D)) \texttt{ is TRUE} \end{cases}$$

Contradicts the initial assumption `(4)`, therefore m does **not** exist.

Appendix B

OpenStack Filter Scheduler

The latest versions of Openstack use the Filter Scheduler,[1] which supports filtering and weighting to enable informed decisions on where a new instance should be created. Depending on the employed filters and the weighting approach, the schedulers' complexity ranges from simple (basic) to very complex. Figure B.1 illustrates a sample decision process when using the Filter Scheduler.

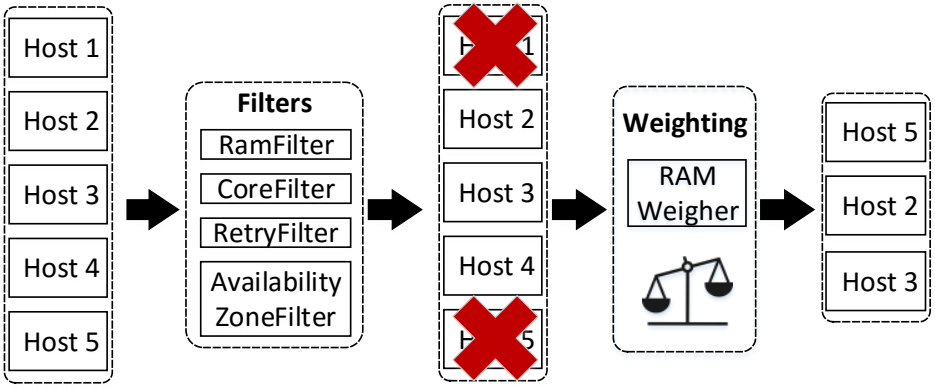


Figure B.1: OpenStack sample Filter Scheduler[1]

The scheduler uses various filters to find the potential destination hosts and then orders these hosts based on a weighting scheme. For example, the RamFilter only passes hosts that have sufficient RAM for the new instance. On the other hand, the RAMWeigher is used to

sort the valid hosts based on their available RAM.

In our deployment, we used Openstack (Icehouse) with the default scheduler options.

`/etc/nova/nova.conf`:

```
scheduler_default_filters=RetryFilter,
AvailabilityZoneFilter, RamFilter,
ComputeFilter,ComputeCapabilitiesFilter,
ImagePropertiesFilter,ServerGroupAnti  AffinityFilter,ServerGroupAffinityFilter
ram_weight_multiplier=1.0
```

Appendix C

Ruby ERB/Erubis Template for Generating an Alloy Model

The Ruby ERB/Erubis template pictured below is used to generate the Alloy model (*als* file) that is passed to the Alloy Analyzer. The scenario specific information from the three inputs is passed to the template through hash table-like Ruby structures (e.g., `roles`, `interfaces` `role_implementations`).

```
/*
* Constraint Model - Compiling Abstract Specifications into Concrete Systems
*
* Inputs: ARML specification, list of available role implementations
* Objective: Trying to find a compatible combination of role implementations
*            that will fulfill the roles defined in the ARML specification.
*
* @author Alex Bardas
*
*/

module constraintModel/dependent_dependee_check

//Roles from the ARML specification
abstract sig Role {
  interface: one Interface,
  imported_roles: set Role
}
```

```
<% for role in roles -%>
one sig <%= role.keys[0].capitalize %> extends Role {}
<% end -%>

//Interfaces
abstract sig Interface {}
<% for interface in interfaces -%>
one sig <%= interface.capitalize %> extends Interface {}
<% end -%>

//Role Implementations from the list of available implementations
abstract sig RoleImpl {
  interface: some Interface,
  imported_role_impls: set ImportedRi
}

sig ImportedRi {
  interface: one Interface,
  ri: some RoleImpl,
  imported_ris: interface one -> some ri
}

<% for ri in role_implementations -%>
one sig <%= ri.keys[0].capitalize %> extends RoleImpl {}
<% end -%>



//Populating the roles and the role_implementations sets with the information
//from the ARML spec and the available role implementations list
fact {

  //parameters' order: current_role, interface
  <% role_names = Array.new -%>
  <% for role in roles -%>
  populateRoleInterface [<%= role.keys[0].capitalize %>,
                         <%= (role[role.keys[0]]["interface"]).capitalize%>]
  <% role_names.push(role.keys[0]) -%>
  <% end -%>

  //Populating role imports
  //parameters' order: current_role, role_1, role_2, ..., role_n
  <% for role in roles -%>
    <% output = "" -%>
    <% current_imports = role[role.keys[0]]["imported_roles"] -%>
```

```
  <% for role_name in role_names -%>
    <% if current_imports.include?(role_name) -%>
      <% output = output + ", #{role_name.capitalize}" -%>
    <% else -%>
      <% output = output + ", none" -%>
    <% end -%>
  <% end %>
populateRoleImports [<%= role.keys[0].capitalize %><%= output %>]
<% end -%>

//Building "lattices" using the available role impls (interface name
//is the "supremum")
//parameters' order: current_role_impl, interface
<% impl_names = Array.new -%>
<% for ri in role_implementations -%>
populateRiInterface [<%= ri.keys[0].capitalize %>,
                     <%= (ri[ri.keys[0]]["interface"]).capitalize -%>]
<% impl_names.push(ri.keys[0]) -%>
<% end -%>

//Populating Role impl imports
//parameters' order: current_ri, interface,
//  role_impl_1, role_impl_2, ..., role_impl_n
<% for ri in role_implementations -%>
  <% current_role_impls = ri[ri.keys[0]]["imported_role_impls"] -%>
  <% if current_role_impls.length != 0 -%>
    <% for current_role_impl in current_role_impls -%>
      <% output = "" -%>
      <% current_ri = current_role_impl[current_role_impl.keys[0]] -%>
      <% for impl_name in impl_names -%>
        <% if current_ri.include?(impl_name) -%>
          <% output = output + ", #{impl_name.capitalize}" -%>
        <% else -%>
          <% output = output + ", none" -%>
        <% end -%>
      <% end -%>
populateRiImports [<%= ri.keys[0].capitalize %>,
                   <%= (current_role_impl.keys[0]).capitalize%><%= output %>]
    <% end -%>
finishRiImports [<%= ri.keys[0].capitalize %>,
                 <%= (ri[ri.keys[0]]["imported_role_impls"]).length %>]
  <% else %>
<%= ri.keys[0].capitalize %>.imported_role_impls = none
  <% end -%>
<% end -%> }
```

```
pred finishRiImports (ri: RoleImpl, no_of_imported_ri: Int) {
  #ri.imported_role_impls = no_of_imported_ri
}

pred populateRoleInterface (current_role: Role, interface1: Interface){
  current_role.interface = interface1
}

pred populateRiInterface (ri_current: RoleImpl, interface1: Interface){
  ri_current.interface =interface1
}

<% impl_param = "" -%>
<% impl_add = "" -%>
<% impl_substract = "" -%>
<% i = 1 -%>
<% while i <= impl_names.length -%>
  <% impl_param = impl_param + ", ri#{i}: RoleImpl" -%>
  <% if impl_add != "" -%>
    <% impl_add = impl_add + " + ri#{i}" -%>
  <% else -%>
    <% impl_add = impl_add + "ri#{i}" -%>
  <% end -%>
  <% impl_substract = impl_substract + " - ri#{i}" -%>
  <% i = i + 1 -%>
<% end -%>

pred populateRiImports (ri_current: RoleImpl,
                        imp_interface: Interface<%= impl_param %>){
  let ris = <%= impl_add %> |
    ri_current.imported_role_impls.imported_ris =
      ri_current.imported_role_impls.imported_ris +
      imp_interface -> ris

  let not_imported = RoleImpl<%= impl_substract %> |
    ri_current.imported_role_impls.imported_ris[imp_interface] &
    not_imported =
    none
}

<% role_param = "" -%>
<% role_add = "" -%>
<% role_substract = "" -%>
<% i = 1 -%>
```

```
<% while i <= role_names.length -%>
  <% role_param = role_param + ", role#{i}: Role" -%>
  <% if role_add != "" -%>
    <% role_add = role_add + " + role#{i}" -%>
  <% else -%>
    <% role_add = role_add + "role#{i}" -%>
  <% end -%>
  <% role_substract = role_substract + " - role#{i}" -%>
  <% i = i + 1 -%>
<% end -%>

pred populateRoleImports (current_role: Role, <%= role_param %>) {
  let all_imported_roles = <%= role_add %> |
    current_role.imported_roles = current_role.imported_roles +
                                  all_imported_roles

  let not_imported_roles = Role<%= role_substract %> |
    current_role.imported_roles & not_imported_roles = none
}

//SoftwareStack stores the solution
sig SoftwareStack {
  imp: Role -> RoleImpl
}

<% role_param = "" -%>
<% impl_param = "" -%>
<% i = 1 -%>
<% for role_name in role_names -%>
  <% role_param = role_param + ", role#{i}: #{role_name.capitalize}" -%>
  <% impl_param = impl_param + ", ri#{i}: RoleImpl" -%>
  <% i = i + 1 -%>
<% end -%>

pred findImpl (s: SoftwareStack<%= role_param %><%= impl_param %>) {
<% i = 1 -%>
<% while i <= role_names.length -%>

  <%= "role#{i}" %>.interface = <%= "ri#{i}" %>.interface
  #<%= "role#{i}" %>.imported_roles = #<%= "ri#{i}" %>.imported_role_impls
  s.imp[<%= "role#{i}" %>.imported_roles] in
                                    <%= "ri#{i}" %>.imported_role_impls.ri
  s.imp[<%= "role#{i}" %>] = <%= "ri#{i}" %>
  <% i = i + 1 -%>
<% end -%> }
```

```
<% imp_ris = Array.new -%>
<% for ris in role_implementations -%>
  <% for ri in ris[ris.keys[0]]["imported_role_impls"] -%>
  <% imp_ris << ri -%>
  <% end -%>
<% end -%>

run findImpl for 1 SoftwareStack, <%= imp_ris.uniq.length %> ImportedRi
```