Automating the Creation of Virtual Educational Environments to Enhance Cybersecurity and Information Technology Education

A Thesis

Presented in Partial Fulfillment of the Requirements for the

Degree of Master of Science

with a

Major in Computer Science

in the

College of Graduate Studies

University of Idaho

by

Christopher E. Goes

May 2017

Authorization to Submit Thesis

This thesis of Christopher E. Goes, submitted for the degree of Master of Science with a Major in Computer Science and titled "Automating the Creation of Virtual Educational Environments to Enhance Cybersecurity and Information Technology Education," has been reviewed in final form. Permission, as indicated by the signatures and dates below is now granted to submit final copies for the College of Graduate Studies for approval.

Advisor:	______________________	__________
	Daniel Conte de Leon, Ph.D.	Date
Committee Members:	______________________	__________
	Michael Haney, Ph.D.	Date
	______________________	__________
	Axel Krings, Ph.D.	Date
Department Chair:	______________________	__________
	Frederick Sheldon, Ph.D.	Date

Abstract

Hands-on exercises utilizing virtual environments have been demonstrated as an effective means of teaching cybersecurity. These exercises are currently built manually by educators, and require significant time and Information Technology (IT) knowledge. Manual setup also leads to inconsistencies, especially when sharing or reuse is desired.

ADLES seeks to solve these problems by enabling educators to easily design, specify, and build portable virtual environments for their courses. This automation, and its associated specification language, can save significant time and effort and ensures the creation and deployment of deterministic, repeatable, and shareable instructional environments. In this thesis, the design and implementation of ADLES is described, and its capabilities are demonstrated using real-world scenarios. In the future, ADLES can serve as a framework for sharing these environments, possibly through an open-source repository of pre-built exercises, competitions, and classes, helping schools that lack the resources for high-quality cybersecurity education.

Acknowledgements

First and foremost, I would like to thank my advisor, Dr. Daniel Conte de Leon, for his constant support and guidance during my graduate studies. His passion for security and teaching, and genuine care for his students, continues to inspire me.

I would like to thank Dr. Alton Campbell, for his excellent mentorship and guidance during my undergraduate studies at this University. Without his support, I would not be where I am today.

I graciously acknowledge the support of the National Science Foundation Scholarship for Service CyberCorp program for their financial support of my graduate studies and a portion of my undergraduate studies.

Much of this work would not have been possible without the support of the Spring 2017 CS 439 Applied Security Concepts course. I would like to thank them for providing their tutorials and for their honest feedback during this research.

I would like to thank Victor House for his assistance with running RADICL, helping me hash out ideas, and setting up the testbeds. I would like to thank Keith Drew, for his valuable feedback and for keeping me inspired. I am grateful to Antonius Stalick, Zachery Lontz, and the other SFS students for their invaluable input and feedback while working on this, and to Matthew Brown, for graciously providing his thesis LaTeXtemplate. Finally, I would like to thank Christopher Waltrip, for being one of the initial inspirations of this work.

Dedication

Thank you to my family, for your unconditional and continuous love and support through my college career. This would not have been remotely possible without you all.

Table of Contents

List of Figures

List of Code Listings

List of Acronyms

ADLES	Automated Deployment of Lab Environments System
AD	Microsoft Active Directory
API	Application Programming Interface
CCDC	Collegiate Cyber Defense Competition
CDC	Cyber Defense Competition
CTF	Capture the Flag
CS	Computer Science
FOSS	Free and Open-Source Software
ICS	Industrial Control System
IP	Internet Protocol
IT	Information Technology
JSON	JavaScript Object Notation
NSA	National Security Agency
NIST	National Institute of Standards and Technology
NICE	National Initiative for Cybersecurity Education
NIC	Network Interface Card
NSF	National Science Foundation
OS	Operating System
OCCP	Open Cyber Challenge Platform
OVF	Open Virtualization Format
PRCCDC	Pacific Rim Collegiate Cyber Defense Competition
PyPI	Python Package Index
RADICL	Reconfigurable Attack-Defend Instructional Computing Laboratory

SDK	Software Development Kit
VM	Virtual Machine
VMI	Virtual Machine Introspection
VLAN	Virtual Local Area Network
VMOMI	Virtualization Management Object Management Infrastructure
YAML	YAML Ain't Markup Language

Chapter 1: Introduction

If we adopt a picture that ignores practice, our field (computing) will end up like the failed "new math" of the 1960s - all concepts, no practice, lifeless; dead. - Peter Denning

Cybersecurity is a broad and rapidly changing field. That nature has historically made it difficult to teach effectively using traditional educational methods. The explosion of innovation and communication ability in the past decade, and the resultant threat landscape, has posed many new challenges that these methods were ill-equipped to deal with. The continued prevalence of vulnerabilities in software and systems[27, 60, 54], combined with a shortage of skilled security workers[40, 61], underscores the need for security education to be more tightly integrated into Computer Science and Information Systems curricula[49, 16]. Hands-on methods of education are well known as an effective means of teaching and training.[32, 39, 57] Their application to the teaching of cybersecurity in recent years has shown great promise[14] in tackling this lack of integration.

1.1 Problem

Teaching cybersecurity effectively using hands-on methods requires secure, dynamic, and reproducible virtual environments[29, 45]. At present, creating these environments is difficult, requiring specialized Information Technology (IT) knowledge and significant amounts of time and effort[58]. They are often limited to the platform they were created on, making it difficult or impossible to share with other institutions, leading to duplication of work. Lessons can be difficult to reproduce in the same way semester after semester, let alone shared with other educators or students. Finally, the mapping of curricula learning outcomes or objectives to the environments created is often complicated or unclear.

These issues are due to the setup process being non-deterministic. This makes it difficult to re-create the exercise in subsequent semesters if the original environment is lost, and for any research done using the exercise to be replicated. Moreover, last-minute changes to deployed environments are effectively impossible, as any change requires applying the change to *all* of the systems deployed.

The difficulties in the exercise setup process mean security and safety often become secondary considerations. Educators, system administrators, and students take shortcuts to get things to "just work," possibly jeopardizing security (Psychological acceptability). Additionally, due to knowledge gaps and lack of experience, implementations are often imperfect, raising further safety and security concerns for educational networks and IT resources.

For classes that require students to make their own tutorials, excessive amounts of time (dozens of hours) are spent on environment setup, detracting from learning. For competitive environments, that usually have large and complicated networks, setup time can be in the thousands of hours. Finally, grading and scoring these environments is very difficult, and it is often simple for students to cheat.

1.2 Solution

The approach to solving this problem is twofold: the creation of formal specifications to describe virtual educational environments and the development of a system to create the environments described by the specifications.

The objectives of this approach are as follows:

- Reduce the need for educators to be specialized in IT to teach using hands-on methods.
- Reduce the time and effort of environment setup by automating workloads and providing a user-friendly interface.
- Introduce determinism to improve the repeatability and research quality of hands-on educational methods and improve mapping of curricula to the created environments.
- Improve portability of environments, in order to facilitate the building of a collaborative security education community.
- Maintain flexibility and extensibility to accommodate future requirements.
- Maintain platform independence where possible, and ensure supporting additional platforms is relatively painless.
- Improve the data collection capabilities of educators. This aids educators ensure that educational objectives are being met, mitigates issues such as cheating and academic dishonesty, and provides definitive and reliable results when conducting research.

Since a deterministic model is one in which there is no randomness involved in the development of future states of the model, a created virtual learning exercise should always produce the same environment (satisfying the desired learning objectives) from a given starting condition or initial state. It then follows that this initial starting state should be a formal specification of the environment that will be produced. To accomplish this for virtual educational exercises, three formal specifications were developed: the package specification, the exercise specification, and the infrastructure specification.

Complementing and implementing the specifications is the Automated Deployment of Lab Environments System (ADLES), a platform-independent system using self-contained exercise "packages" to automatically and deterministically create virtual environments. Using these specifications and ADLES, an educator can easily build the environments they require for a tutorial or competition, and be able to share what they have created with other educators at their school or in the academic community.

1.3 Thesis Overview

Chapter 2 describes the two types of hands-on cybersecurity educational exercises ADLES is being designed to support, tutorials and competitions. It also provides some background on virtualization technologies, including provisioning systems, and describes the target environment, the Reconfigurable Attack-Defend Instructional Computing Laboratory (RADICL) lab.
Chapter 3 discusses requirements for the system and describes the design of ADLES and its components, including the deployment phases, platform-independent interface, and specification parser.
Chapter 4 describes the three formal specifications in detail. These include the package specification, the exercise specification, and the infrastructure specification.
Chapter 5 discusses the implementation of ADLES. This includes implementation choices such as programming language, the vSphere interface implementation, and the development process.
Chapter 6 describes in detail two primary example scenarios used to test and demonstrate the system's capabilities: a hands-on penetration testing tutorial and a cyber defense competition. Additionally, it briefly discusses three additional examples: two tutorials, and a research experiment.
Chapter 7 briefly describes the testing environment, then presents and discusses the results of using ADLES to generate the examples on those testing environments.
Chapter 8 discusses related works that are solving similar problems, some of their limitations, and compares several to ADLES.
Chapter 9 discusses future work for ADLES, including the project's ”grand vision”, expansions of the specifications, new platforms the system could support, and next steps for improving and expanding upon the work done in this thesis.
Chapter 10 summarizes and concludes the thesis.

Chapter 2: Background

I hear and I forget. I see and I remember. I do and I understand - Confucius

Hands-on cybersecurity educational exercises teach students important security concepts by allowing them to experience how these concepts are applied in practice. These exercises are usually performed on Virtual Machines (VMs) created by an instructor, running on a workstation or a shared virtual environment. During an exercise, students access the VMs and perform tasks as they would on normal systems.

2.1 Tutorials

Tutorials are a structured approach to hands-on cybersecurity education. They generally begin with a short background on the topic of the tutorial, followed by a series of tasks students perform. The level of guidance and instruction during the task phase varies by tutorial and instructor. Grading is typically done using lab reports on or direct verification of the results of actions performed by students during the tutorial.

Typical examples of tutorials include the following:

- Write rules for the Snort intrusion detection system to detect an Nmap port scan and a directory traversal in a URL[65].
- Perform a Cross-Site Request Forgery attack on a vulnerable application, then implement mitigations for the attack[23].
- Learn how the *Set-UID* security mechanism works in Linux, exploit an improper use of it, and implement proper use of it[24].
- How to use Microsoft Active Directory (AD) and Group Policy management tools to manage the security of Windows workstations on network. Students implement safe password policies, configure workstations to be secure, and lock down applications that run on the workstations[41].

2.2 Competitions

Competitions are a less structured, ”real-world” approach to cybersecurity education. Common examples are Capture the Flag (CTF) and Cyber Defense competitions. These are usually run on a shared server or a virtualized environment that students log in to. Scoring is performed by awarding points for solving problems, maintaining availability of services, or preventing attacks.

The following are some real-world examples of these competitions:

- PicoCTF: Traditional CTF geared toward high school students. Consists of a series of security problems to solve that are assigned point values[12].
- OverTheWire: Publicly accessible website and Linux server that has sequences of challenges to overcome. Problem difficulties range from solving Caesar Ciphers to exploiting the Linux kernel[15].
- CCDC: The Collegiate Cyber Defense Competition (CCDC) is an attack/defend scenario-based exercise that emulates a real business IT network with a wide variety of systems[11]. Students (the Blue team) have to defend this network against experienced attackers (the Red team), while preserving up-time for their services and performing business tasks (”injects”) throughout.
- TracerFIRE: Forensic and incident response exercise that simulates an industrial complex network, including typical business IT systems and Industrial Control Systems (ICSs). Malware from real-world cyber attack or espionage campaigns and physical models of the complex are utilized to add realism to the exercise[1].
- ThunderBird Cup: Cybersecurity awareness exercise for 6-12th grade students that utilizes a combination of training and competition over the span of a three-day workshop[43].
- DEFCON: World-renowned annual CTF competition hosted by the infamous hacker conference. Consists of teams attacking services and solving challenges to get flags, while also sabotaging the other teams attempting to do the same[18].

2.3 Virtualization Platforms

Historically, networks were emulated using large numbers of physical machines connected in a tangle of cables to numerous switches with complicated configurations[44]. This was unwieldy and expensive, to say the least, and changing configurations was notoriously difficult. Today, Virtualization technology enables these emulations of networks to be performed on a single physical server.[1] There are a number of platforms that provide this technology, including the open-source Xen and KVM, and proprietary Microsoft Hyper-V and VMware vSphere. The capabilities of these platforms to dynamically create realistic networks of machines with limited hardware resources makes them ideal for conducting cybersecurity and IT educational exercises. As a result, almost all hands-on cybersecurity educational exercises developed in the past decade have utilized virtualization to some extent, with many being completely virtualized.

[1] Virtualization technology is a complex subject that is beyond the scope of this thesis to discuss in depth. Refer to Barham[5] for an introduction to the modern techniques and technologies used in this thesis.

The VMware vSphere Platform

VMware vSphere is a robust virtualization platform with a broad range of capabilities. Notably, these include emulation support for most x86-based Operating Systems (OSs), saving and recovery of a VM's state at any point in time using snapshots, virtual networking, and abstraction of hardware resources using host clustering and storage networking technologies. VMs on the platform are essentially the same as their hardware counterparts, and can be configured with networking, video, sound, etc. Access to the platform is provided through a web or workstation-based graphical interface, or programmatically through the Virtualization Management Object Management Infrastructure (VMOMI) Application Programming Interface (API).

The ESXi hypervisor, also known as "the host," runs on server hardware ("bare metal") and provides the virtualization capabilities of vSphere. These hosts generally do not have their own storage, instead relying on abstracted hard disk arrays known as Datastores. At the heart of the platform is the vCenter server, which enables centralized management of the platform through a unified interface and the VMOMI API. Using vCenter, hosts and Datastores can be grouped into Clusters, which enable effective use of and load balancing between hardware resources. Finally, these Clusters and individual hosts and Datastores can be grouped together into Datacenters[35].

Cloud Platforms

Cloud platforms are services that provide Internet-accessible virtualization resources for a nominal fee. Some examples include Microsoft Azure, Amazon AWS, DigitalOcean and the Google Cloud Platform. Due to intense competition between the providers the past several years, the cost to use these platforms has become very economical. This opens up new possibilities for schools without the resources, money, or technical expertise to build out their own infrastructure. These schools, using a system like ADLES, would only have to pay for time spent conducting exercises on a cloud platform, enabling educational exercises that would otherwise not be possible under constrained budgets.

Provisioning Tools

Configuration of newly created VMs is often done using provisioning and configuration management tools, known as "provisioners", that automate the configuration of a system to match a given specification. Some examples of these are Ansible[2], Puppet[52], and Chef[13]. Configuration is accomplished using either a specialized software agent that runs on the system being configured (Puppet, Chef) or common tools already included with the system (Ansible). ADLES can utilize these provisioners to perform configurations of services in lieu of a user.

Some provisioners also have the ability to manage and create virtualization infrastructure, such as VMware vSphere VMs or Docker containers. Unlike ADLES, their configurations are fairly complex and generally not platform-agnostic. Refer to Chapter 8 for further discussion of this.

2.4 The RADICL lab

The Reconfigurable Attack-Defend Instructional Computing Laboratory (RADICL) is a cybersecurity educational lab created in 2004 by the University of Idaho Computer Science (CS) department[10, 9, 42]. Since then, it has gone through a number of iterations that improved its capabilities by utilizing new technological advances, the most recent of which was in 2014. Its primary objective is to provide the ability for students to do exercises and experiments in a safe, controlled, and isolated environment. It is utilized by several cybersecurity courses at the University for hands-on exercises, and is the type of environment that ADLES is designed to assist.

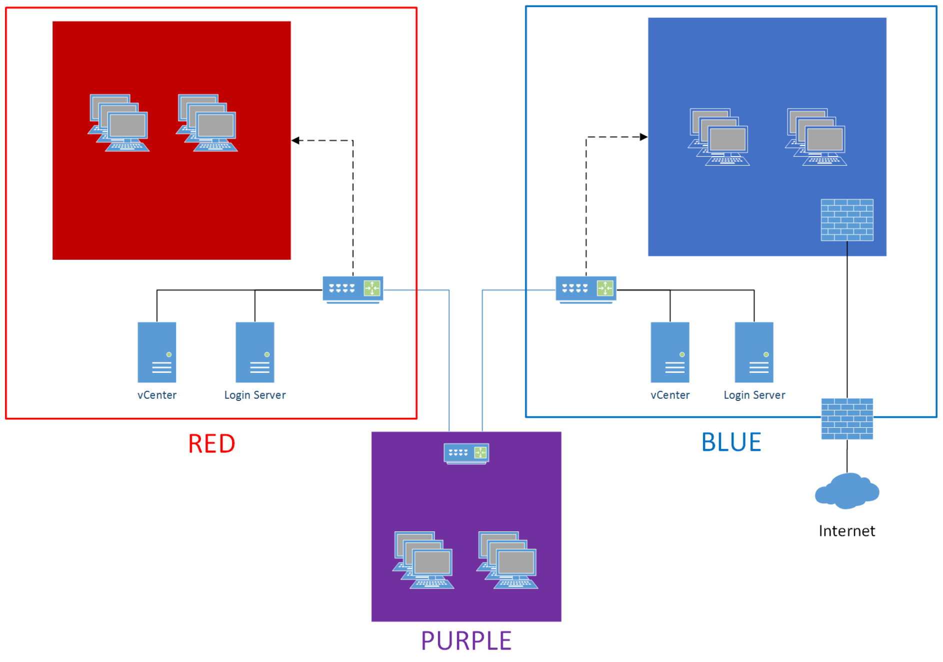


Figure 2.1: Generalized network design of RADICL

Software

RADICL fully emulates real networked computing environments through virtualization technology using the VMware vSphere platform described in the previous section. Instructors can set up and run hundreds of networked VMs that can be accessed by dozens of students simultaneously during an exercise.

These VMs run on one of two sides of the lab: the isolated *Red*, and the Internet-connected *Blue*. Red is isolated from Blue, preventing mistargeted attacks or malware from affecting the University network or the Internet at large. As part of this isolation, students access the environment from the *Purple* network using specialized "thin clients" that run only the software needed to access the environment, with no access to the Internet. Additionally, lab infrastructure, including routing, AD, DHCP, DNS, and other services, is completely virtualized, further improving isolation.

Hardware

The lab is built on exclusively Dell hardware and resides in a single half-height portable server rack (See figure 2.2). The Blue and Red sides of the lab each have their own Dell VRTX chassis with two servers and a number of storage drives, as well as a Dell R310 blade server that hosts the infrastructure VMs for that side. Blue side's VRTX has a Dell M630 and a Dell M520 server, while Red side has two M520 servers. Purple's infrastructure is housed on the R310 between Blue and Red. Each VRTX has 32 physical and 64 logical cores across the two servers and four processor sockets, giving the lab a total of 64 physical and 128 logical cores of computation power. In addition, each chassis has 192GB of RAM, for a lab total of 384GB. Each side has approximately 7TB of RAID 10 redundant shared storage that can be used for storing OS images, VM templates, and payloads for exercises. Finally, networking between devices in the lab is done using a Pica8 Layer-3 managed switch, featuring 48 Gigabit Ethernet ports, and four 10G fiber sockets. These hardware capabilities enable the lab to effectively run most cybersecurity exercises, with limitations only recently beginning to be reached.

The thin clients students use are low-spec Dell WYSE machines that connect to the Pica8 switch to access the Purple network. Each client has two 24-inch monitors, allowing students to work in both sides of the lab simultaneously. In fact, the common use case is to have exercise materials pulled up in an Internet-connected Blue-side VM on one monitor while performing exercises in Red on the other monitor. Additionally, the lab is equipped with two 60-inch wide-screen HDTVs, enabling instructors and students to effectively teach concepts or walk through tutorials. Finally, all equipment in the lab, from the servers to the thin clients, is attached to backup batteries. This provides protection from surges, brownouts, and blackouts, and ensures lab activities are not impacted by power issues.

Usage

During the current semester, RADICL is used to teach two CS courses at the University: CS 439/539: Applied Security Concepts in the Spring, and CS 499/502 Cyber Defense I/II in Fall/Spring. CS 439 consists of hands-on tutorials constructed by students covering a wide range of security concepts, while

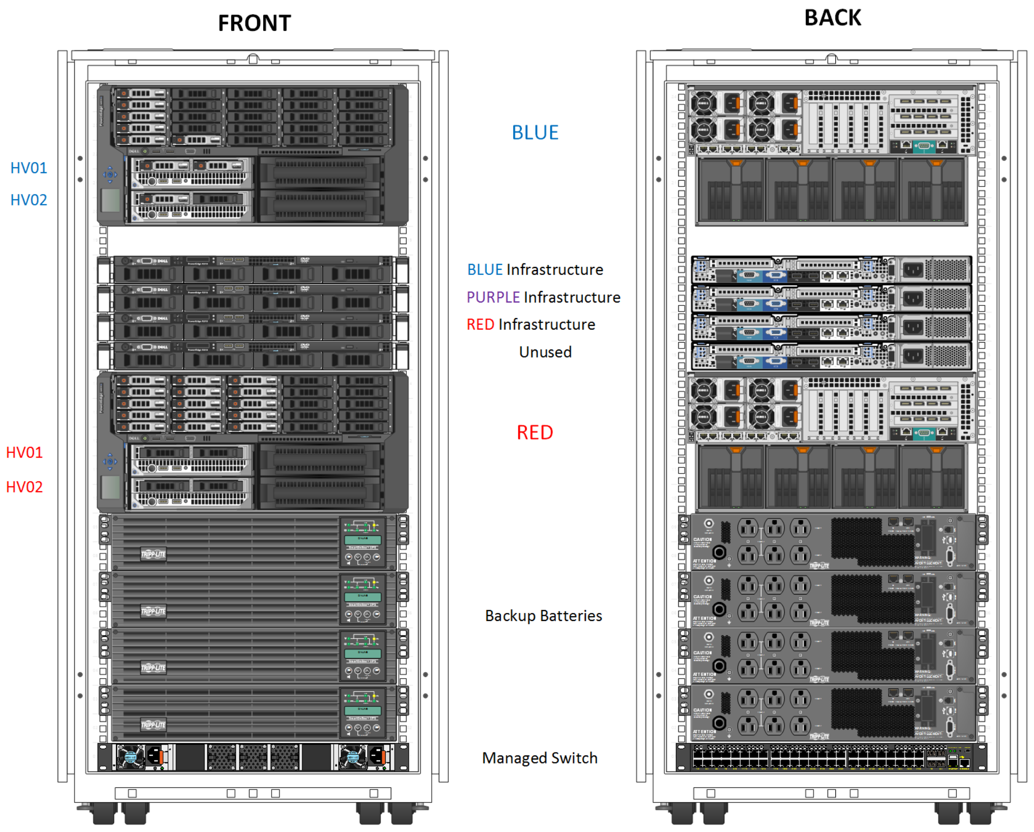


Figure 2.2: Server hardware setup of RADICL

CS 499's focus is on learning how to secure diverse systems in preparation for the Pacific Rim Collegiate Cyber Defense Competition (PRCCDC). Out of the two, CS 439 is the heaviest user, with two-hour tutorials two days a week. These tutorials often involve potentially dangerous activities, such as exploiting vulnerabilities in systems or handling malware, and therefore are run on the *Red* side of the lab.

The average tutorial takes a significant amount of time to prepare. According to students who use the lab, this preparation time varies from 10 to 40 hours for a team of two students, including research time, experimentation, and creation of the exercise. This has led to students being in the lab during other meetings and classes, late hours, over weekends, and even during academic breaks. There are a number of factors causing this, including manual creation and deployment of environments, and configuration of networking for environments. Additionally, these issues are compounded by student's (understandable) lack of knowledge about vSphere, especially at the beginning of the semester. ADLES is intended to help address these issues through the use of automation by reducing the amount of esoteric platform knowledge students must know to create these environments, and significantly reducing the time spent on deployment and configuration.

Chapter 3: Description of the ADLES system

The ADLES system requirements, design, and components will be explained and discussed in this chapter. This includes an overview of how the system works from the perspective of its users and how the specifications play into the system operation, laying the groundwork for Chapter 4's description of the specifications themselves.

3.1 System Requirements

The system requirements serve to codify in software terms what is outlined in the objectives discussed previously. They are the following:

1. Deterministic. Specifications should build the same environment every time, regardless of platforms used. Furthermore, the same specification should be able to tear down and cleanup an environment, even if the creation was performed from a different workstation with a different version of the system.

2. Platform-agnostic. The system should be able to interface with multiple different platforms in the same manner. Likewise, the system should be able to run on a wide variety of systems, and not be locked to a specific OS. Both of these improve the portability and versatility of the system.

3. Versatile. New functionality should be straightforward to add to the specifications and implement in the system. Components should be modular and loosely coupled, so they can be added or removed as needed.

4. Portable. Sharing of packages and their components with others should be relatively easy to do. This facilitates improvement and reduction of repeated work through ease of collaboration. Educators and students can generate new ideas, build on existing ones and fix bugs, enabling a cohesive community around hands-on cybersecurity education.

5. Self-contained. All the components to build and run an exercise using the system should be able to be packaged into a single compressed file or Git repository. This is closely related to the portability requirement, as requiring external dependencies (that a user may not have access to) greatly hinders the portability of the system.

6. User-centric. The system should abstract away unnecessary details for students/instructors, while retaining their ability to customize and fine-tune if desired.

7. Data-aware. The system should be aware of data being generated by an exercise, and make efforts

to ensure it is preserved intact. This is critical for grading student work, scoring competitions, and conducting accurate research.

3.2 System Design

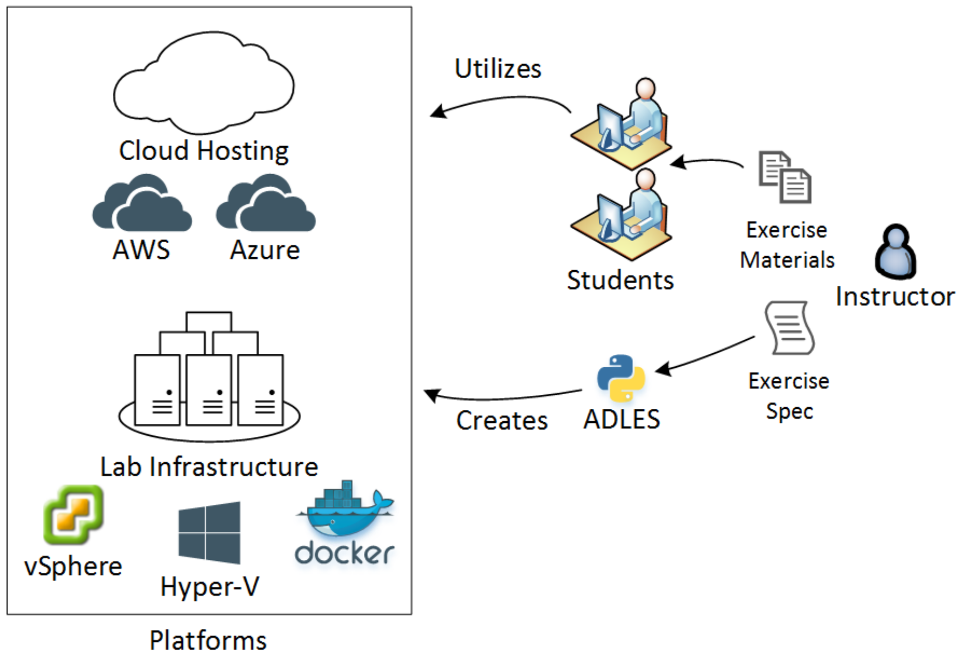


Figure 3.1: Overview of how ADLES is used

The ADLES is designed from the ground-up to be platform-agnostic, extensible, and user-friendly. The idealized version of the system workflow is the following:

1. The instructor writes and inputs an exercise specification into the system.

2. The system interacts with virtualization platform(s) to create the exercise.

3. Students access the virtual environment running on the platform(s) and conduct the exercise.

4. The system saves results from the exercise upon its conclusion.

5. The system cleans up the environment and frees platform resources.

In practice, this process looks a bit different. The creation of an environment is done in two phases: the *Mastering* phase, and the *Deployment* phase. During the Mastering phase, *Master* instances of the base services that will be used in the final environment are created and configured. This configuration is performed by the educators putting together the exercise or a provisioning system such as Ansible.

These instances are then used during the Deployment phaseto build (”deploy”) the exercise environment. Therefore, changes to a deployed environment simply require un-freezing the Master instance(s), making the change, and re-running the deployment phase.

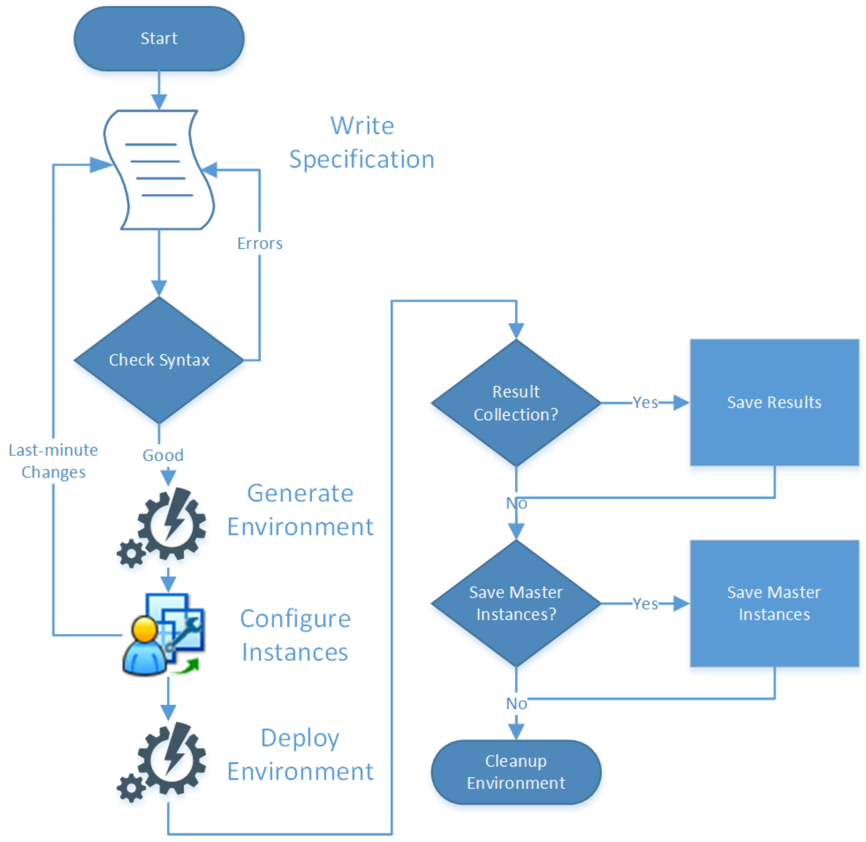


Figure 3.2: Usage flowchart for ADLES

The complete process flow of the system, then, is the following:

1. Instructor writes the exercise specifications and inputs them to the system running on a machine with network access to the platforms desired.
2. System reads the specifications and validates their syntax and semantics.
3. System instantiates the platform interfaces for the platforms defined in the specifications.
4. System creates Master instances on the platforms.
5. Instructor (or a provisioning system) configures the Master instances.
6. System verifies and freezes the Master instances.
7. System uses the Master instances to deploy the full exercise environment on the platforms.

8. Students access the environment and perform the exercise.
9. System saves results from the exercise, log files from system operation and the Master instances, if desired.
10. System cleans up the environment and frees platform resources.

3.3 System Components

ADLES is made up of a number of components that are used to perform the process described in the previous section. These include parsing and syntax validation, interacting with platforms in a generalized manner, saving the results of exercises, and cleaning up of environments post-exercise.

Parsing and Syntax Verification

The validation component of ADLES handles parsing and validation of an exercise specification's syntax, and verification of any files or other content needed for the exercise. Humans are fallible, and even cybersecurity educators make mistakes. The purpose of the syntax checker is to both ensure these mistakes are caught before an environment is deployed, and aid those new to the specification syntax by providing a way to check if what they made is canonical.

The Platform Interface

The *PlatformInterface* class is a generic API that is used to interact with infrastructure platforms in a generalized and consistent manner. This abstraction separates the core logic of ADLES from the platform implementations by providing a small set of well-defined interface functions.

These interfaces contain only the logic needed to implement the functions they provide, such as the creation and destruction of environments. In order to preserve this purity of logic, platform management and interfacing is done through specialized platform wrapper classes. These classes provide an abstraction "wrapping" the low-level API for a given platform, reducing the complexity of the interface logic and improving the ability to add new interface functionality. They manage platform state, handle errors, and provide the interfaces with methods to perform the necessary environment creation actions on the platform.

At initialization, the PlatformInterface ingests the parsed infrastructure specification, initializes of any user groups that are required, and initializes the platform-specific interfaces using the specification and groups. Once initialized, the application can call the desired functions presented by PlatformInterface, and it will dutifully make the requested call to each platform-specific interface it has instantiated.

One advantage afforded by this design is the ability to easily extend the platform's functionality. For instance, to add the ability to freeze an in-progress exercise, a function "freezeExercise()" can be added to a platform's interface. Then, this functionality can be utilized by a new entry-point, the addition of a command line argument to the main entry-point, or by a extension to the exercise specification.

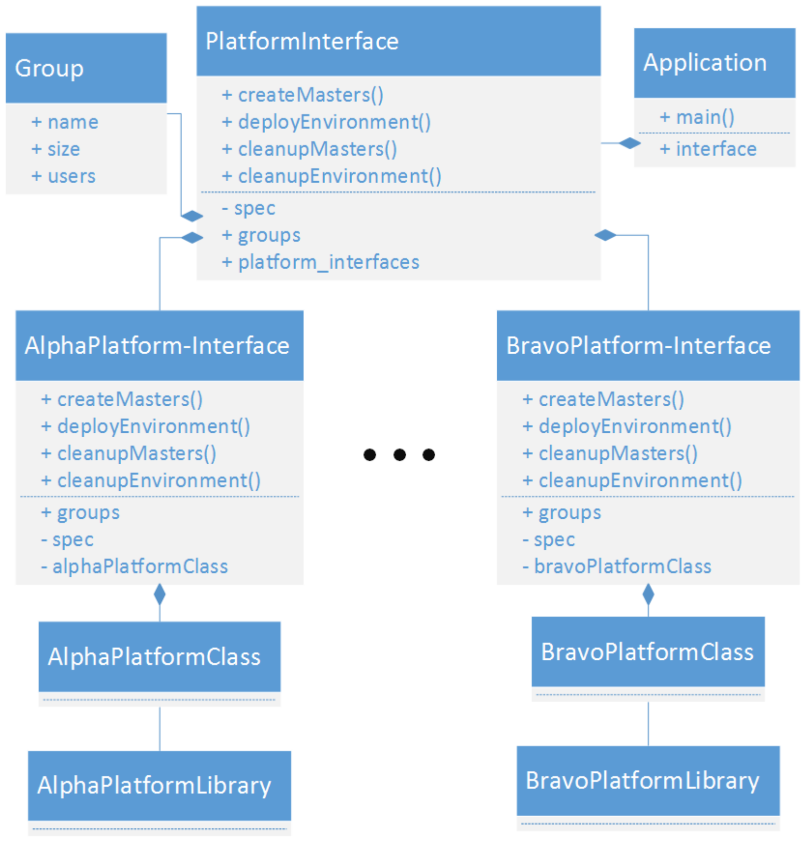


Figure 3.3: Generic platform interface design

Saving Results

The system is designed to be able to accurately save the results of exercises. These results are packaged and included with the other exercise materials in the package. When the results are later examined, all the components that were used in the generation of them are already present and accessible, making the mapping from a given result, to the object that produced it, relatively simple and straightforward.

There are a variety of methods that could be used for result collection. An additional network interface could be added to VMs and attached to a network isolated from the exercise, for the purposes of aggregating logs to a centralized logging server. Virtual switches could be set to promiscuous mode, and the resulting network capture data saved. System calls performed by the VMs could be captured and saved using Virtual Machine Introspection. While not all of these methods will be implemented, the system is designed to be able to accommodate them without requiring significant modifications.

Logs from application runs are sent to a centralized logging server, such as Splunk or ELK, or a local system logging interface. This protects the logs from tampering by students or instructors, as system utilities or logging servers are usually subject to strict access controls, and enables analysis using existing logging tools. If the system is installed properly, users should not be able to modify the its code without administrator privileges, preventing tampering with the logging system and contamination of its output. The purpose of this is two-fold: to fulfill the requirement of keeping a complete and unaltered history of a given run, and to aid users and platform administrators in debugging issues with the system.

Cleanup

Cleanup is involves shutting down and removing any instances, networks, files, or other materials created by the deployment of or during an exercise. The cleanup process can be run from any workstation with network access to the platforms and requires only the specification used to generate the environment. This makes the instructors' task of freeing up resources after the completion of an exercise far less daunting and time-consuming. Furthermore, the automation ensures only the objects used for the exercise are removed, preventing the occasional accidental deletions done by tired instructors after a long day of teaching.

Chapter 4: Formal Specifications

Virtual educational environments should be described in a formalized manner that is *repeatable*, *portable*, *extensible*, and *approachable*. These desired characteristics should be like the blueprint of a building, and ideally as rigorous as one. Therefore, the requirements for the specification design are as follows:

1. After reading the description of an environment, it should be simple to determine what will be built, with the sole purpose of ADLES being to manifest the environment defined.

2. The description should be able to be shared easily, read by anyone with a basic text editor, and annotated in a users' native language.

3. Addition of new features should be straightforward and able to be done in such a way that the extensions do not impact the base functionality of systems not running the extensions.

4. The description should be defined in such a way that anyone with basic technical background can both understand the syntax and be able to write a new description for a environment without specialized knowledge or training.

There are many excellent examples of this sort of specification, two of which are Docker Compose[22] files and Ansible Playbooks[3]. Docker Compose files describe the setup of a web-based application in terms of its services, networks, and storage volumes. Anyone that reads it can easily determine what will be created, thanks to its concise syntax and the ability to add human-readable labels and names, so the only purpose of running Docker Compose on the file is simply to manifest the environment described. Ansible Playbooks work in a similar manner, using a declarative syntax to describe the desired configuration. They differ in that Playbooks are specialized for provisioning systems, regardless of hardware or virtual platform they are running on. This difference is most notable in the concepts of user-specifiable "tasks". These enable customization and extensibility of the Playbook for specialized usages without having to modify Ansible's code, adding flexibility to the language and system.

Both of these examples are easy to share, requiring only the file and the tool to generate the environment. They are also both based on YAML Ain't Markup Language (YAML). YAML[6] was selected as the base language for the specifications for similar reasons the previous two examples were: it's human-friendly, it's ubiquitous, and it's portable. Since its syntax is similar to that of many programming and scripting languages, technical students should be able to use it without having to learn yet another language. High-quality and well-supported parser implementations exist for many programming languages, improving portability of the specifications to systems other than ADLES.[1]

This ideal description is the guidance for the development of the specifications for ADLES. There are three primary specifications that were developed:

- The Package specification, which describes the contents of a "package" and contains metadata for the package itself.
- The Exercise specification, which describes the virtual environment for an educational exercise.
- The Infrastructure specification, which describes the hardware and software infrastructure that will be hosting the environments.

The specification descriptions that follow contain selected portions of the specifications being discussed. For the full specifications, refer to the appendices or the ADLES project repository[33].

4.1 Specification Syntax and Semantics

Labels

Labels are given for each semantic component as a comment in a specification. Children do not have to be specified if their parent is not required or defined. If no label is given, then context will determine label. The labels for components are as follows:

- *REQUIRED*: Must be defined. Not doing so is an parse-time error.
- *Suggested*: Should be defined. Not doing so will result in a parse-time warning.
- *Optional*: Does not have to be specified.
- *Option X*: One of the options specified at that level of nesting must be defined, 'X' being the identifier for that option. Not doing so is a parse-time error.

Extensions to the Specifications

Extensions are relatively simple to implement for most aspects of the specifications. Simply add the desired functionality to the specification, increase the version number, and provide an example using the new functionality. Since the version number of each specification is tracked using Semantic Versioning[51], the extension writer only has to state what version of the specification had the extensions added when

[1]YAML 1.1 is used due to lack of support for 1.2 by PyYAML's underlying C library, LibYAML[59]. However, since YAML 1.2 only adds full JavaScript Object Notation (JSON) compatibility, the trade-off is acceptable[26].

instructing users about their extension. Additionally, most of the configurations can be extended without breaking other versions or implementations of ADLES. Anything that is not required or supported by a given version or implementation will simply be skipped over, providing backward compatibility. Refer to Future Work (Chapter 9) for examples demonstrating the extensibility of these specifications.

4.2 Package Specification

The package specification describes the contents and layout of an exercise package. This ”exercise package” is a self-contained bundle of the elements needed to run an educational exercise. The only required component is the environment specification, any further contents will vary by exercise. In practice, this package is a directory consisting of various files and sub-directories. To share the package, the package itself and its sub-directories can be recursively compressed into a single small file using any number of free compression tools. The package structure and elements can be seen in figure 4.1.

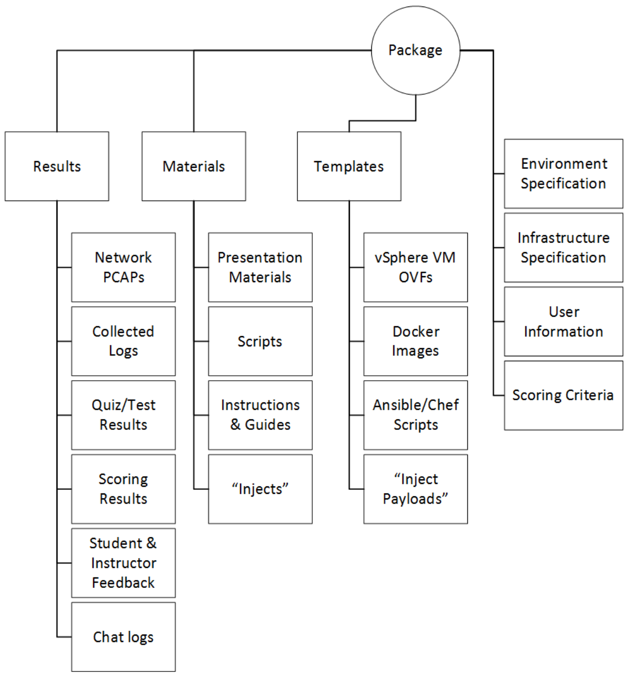


Figure 4.1: Structure of the contents of an exercise package

There are three main directories in an exercise Package: *results*, *materials* and *templates*. The *results* directory is used to contain results from exercise runs. Each time the exercise is run, results are saved and timestamped with the time and date the exercise was begun. These results can be used for research purposes, as well as to formulate improvements of future exercises. Additionally, they can be included when sharing the exercise, enabling others to view and compare the results with their own runs of the exercise. The *materials* directory holds any materials needed for the exercise, such as scripts, tutorial walk-throughs, task "injects" for a competition, or presentation materials. Finally, the *Templates* directory contains templates needed for the exercise's services. These can include vSphere VMs in the Open Virtualization Format (OVF), Docker images, Dockerfiles, and virtual disks. It can also contain scripts used for configuration provisioners (e.g Puppet or Ansible) and payloads used for some competition task "injects" (e.g a malicious or obfuscated binary).

Package Metadata

The *metadata* section describes metadata for the package as a whole. The two required fields are the *timestamp* and the *tag*. Timestamp is the date the package was created and tag is unique tag for the package, e.g "uidaho-cs439". They are required to provide a uniqueness property to differentiate packages with similar names and purposes. For example, it would be easy to confuse two tutorials published by different institutions named "Nmap Tutorial" without a uniqueness property. *version*

```
metadata:
  timestamp: "date"   # REQUIRED   Date package was created in UTC
   ↪ format: YYYY-MM-DD
  tag: "tag-name"     # REQUIRED   Unique identifier for this package
  name: "name of package"     # Suggested  Human-readable name for the
   ↪ package  [default: filename]
  description: "description"  # Suggested  Human-readable detailed
   ↪ description of package
  version: "0.0.0"            # Suggested  Semantic version of the
   ↪ package: major.minor.bugfix
```

Listing 4.1: Package Metadata

Package Contents

The *contents* section is where the contents of an exercise package are defined. The only required element is the exercise *environment* specification. Beyond that, there are a number of possibilities, depending on the desired exercise package layout as discussed earlier. Also, note that configurations made at the "package level" will override configurations made elsewhere. For instance, while the infrastructure configuration file for a exercise can be specified in the exercise file itself, it will be overridden by the the one specified in the package.

```
contents:
  environment: "enviro-spec.yaml"   # REQUIRED    Name of the file containing the environment specification
  infrastructure: "infra-spec.yaml" # Suggested   Infrastructure configuration (This can be specified in environment spec as well)
  scoring: "scoring-criteria.yaml"  # Suggested   Scoring criteria (This can be specified in environment spec as well)
  results: "path/to/dir"      # Suggested   Relative path to directory containing: Network PCAPs, Collected logfiles, Quiz/test results, Scoring results, Student & Instructor feedback, and Chat Logs
  templates: "path/to/dir"    # Suggested   Relative path to directory containing: VM OVFs, Docker Images/Dockerfiles, provisioning scripts/packages, other payloads
  materials: "path/to/dir"    # Suggested   Relative path to directory containing: presentations, exercise scripts, instructions/guides/documentation
```

Listing 4.2: Package Contents

4.3 Exercise Specification

The exercise specification is the format used to describe the virtual environment for an educational exercise. It defines and describes all aspects of an educational exercise, such as a competition or tutorial. Some examples are the instances that make up the exercise, user permissions and access to instances or resources, the number of instances, and network connections.

There are a few things to note about the specification. *Instances* is assumed to be one unless specified otherwise, and cannot be negative or zero. The top-level definitions that will be described are only required if used. For instance, if there is no networking or resources required for a given exercise, the *networks* and *resources* sections do not have to be specified.

Metadata

Name is the human-readable name of the tutorial. *Description* is a short human-readable description of what the tutorial will be doing. *Activity* is the name of the class that the tutorial is for, in this case CS 439. The *prefix* is a concatenation of the course number, the tutorial round, and the tutorial number, and will be used to uniquely identify this course from others in the same class or different classes when needed on a given platform. The *version* is the current Semantic Version of the tutorial. Examples of minor revisions are changes such as editing configurations or removing notes, and bug-fix revisions are changes to comments or wording of descriptions. The *folder-name* specifies the path to the folder where the tutorial should reside on the environment. This path will be generated if it does not yet exist in the environment. Finally, since the metadata for all of the examples is roughly the same, it will not be discussed in the next two examples.

The Metadata section is composed of metadata and configurations for the specification. The important features are the *name*, a globally unique prefix to distinguish the exercise from others on the same infrastructure, and the *infra-config*, which contains the infrastructure configuration. In the future, if the infrastructure configuration file is specified in the package, then specifying it here is redundant and not required.

```yaml
metadata:
  name: "name"                    # Suggested   Human-readable title for the
   ↪  specification [default: filename]
  description: "description"  # Suggested   Human-readable detailed
   ↪ description of the specification
  activity: "the activity"     # Optional    The activity the
   ↪ specification is being used for, e.g "CS 439" or "PRCCDC"
  prefix: "GLOBAL-PREFIX"      # REQUIRED    Globally unique prefix that
   ↪ distinguishes this exercise's environments from others on the same
   ↪  infrastructure
  date-created: "date"         # Optional    UTC format: YYYY-MM-DD (
   ↪ Example: 2016-10-12)
  version: "0.0.0"              # Suggested   Semantic version of the
   ↪ document: major.minor.bugfix (Refer to: http://semver.org/)
  infra-file: "filename.yaml"           # REQUIRED    YAML file
   ↪ specifying the infrastructure used to create the exercise
   ↪ environment (See: infrastructure-specification.yaml for syntax)
  folder-name: "/Path/To/Folder-Name"   # Suggested   Path of the folder
   ↪  that will contain the exercise, relative to root defined in the
   ↪ infrastructure configuration
```

Listing 4.3: Exercise Metadata

Groups

The Groups section consists of definitions of user groups, such as instructors, students, and competitive teams. The definitions are used to apply permissions and restrict access to the resources and machines in the generated environment. There are two types of groups: *regular* and *template*.

Users for regular groups can be specified using an AD user group, a JSON file containing the users, or a simple list of the users. Since only permissions are being applied, passwords are not required, but can be added if a given platform or an extension requires it.

Template groups are batch groups created from a common template base, and are usually used for teams in a competition or for group work in a class. The number of group instances must be specified, and users can only be specified using AD groups or file names.

```yaml
groups:
  Group Regular Example:
    # The three different methods of specifying users for a regular group
    ad-group: "Users"        # Option A  AD group must exist
    filename: "a-file.json" # Option B  File format: specifications/user-json-specification.json
    user-list: [ "user-a", "user-b" ] # Option C  List of usernames

  # For creating batch groups from a common template base, the "template" type can be used
  Group Template Example:
    instances: 10           # REQUIRED  Number of groups created from this template. This marks a group as a template.
    ad-group: "Group [X]"    # Option A  '[X]' is the instance number. AD group must exist
    filename: "users.json"  # Option B  File format: specifications/user-json-specification.json
```

Listing 4.4: Groups

Services

Services are the instances that will be created in the exercise environment, such as hosts, servers, and routers, and are defined in the *services* section of the specification. These definitions serve as the primary point of configuration for the services being instantiated in the *folders* section. Each definition's configurations are based on the platform being targeted. While some configurations are shared between services regardless of platform, most are dependent on the platform being configured. It should also be noted that multiple platforms cannot be configured for a service. The increase in complexity for the common use case that this imposes was determined to not be worth the minor gain in flexibility. The same outcome can be accomplished by defining multiple instances of a service, one for each platform, and simply changing which one is used prior to environment creation.

The following configurations are shared between platforms: *network interfaces*, *notes*, *provisioner*, and *resource-config*. Network interfaces is a list of networks and their optional configurations, with the actual interface configuration being done "under the hood" per-platform. Notes are human-readable annotations that can be added to a service, such as the default password used to access that service. Provisioner is the specification of a provisioning tool, such as Puppet or Ansible, and its configuration for that service. Resource-config enables configuration of resources allocated to the service, including number of CPU cores, amount of memory, and amount of disk storage space. If more platform-agnostic configurations are desired, they can easily be added, provided the requisite functionality has been implemented in the platform's interface.

```
services:
  all-service-types:  # Configurations that can appear in any service definition
    note: "A note"          # Optional    Human-readable note visible by  end-user, such as default username/password
    network-interfaces: []  # Optional    List of network interfaces and  their optional configurations  [default: template or container specific]
    provisioner:  # Optional
      name: "name"  # REQUIRED  Name of provisioning tool, e.g Ansible, Chef, Puppet
      file: "file"  # REQUIRED  File to use for provisioner, e.g Playbook, Cookbook, Manifest
    resource-config:  # Optional  Resource allocation configurations for  the service
      cores: 0    # Optional  Number of CPU cores
      memory: 0   # Optional  Amount of RAM in MB
      storage: 0  # Optional  Amount of persistent storage in GB
```

Listing 4.5: Exercise Services - Platform-agnostic configurations

Currently, there are three formalized service definitions: vSphere template, Docker container image[19], and Docker Compose file[20]. Each has its own specialized configurations, which are outlined in its formal definition in the specification. Adding new configurations to a service or defining a new service for a new platform simply requires adding them to the specification.

```
  template-based-service:   # Option A
    template: "name"  # REQUIRED
    template-config:  # Optional    Configuration of Template settings using key-value pairs
      key: "value"
    guest-extensions: no    # Optional    Guest extensions will be installed or enabled (e.g VMware Tools)
  container-based-service:  # Option B
    dockerfile: "file"  # Option A    Dockerfile to build a image
    image: "name/tag"   # Option B    Name and Tag of a pre-built image
  compose-based-service:    # Option C
    compose-file: "filename.yml"  # REQUIRED
```

Listing 4.6: Exercise Services - Platform-dependent configurations

Resources

The *resources* section describes cyber-physical resources and labs/testbeds that are potentially available to a virtual environment, such as SCADA systems, a wireless testbed, a USB device interconnect, a car computer or an electrical power grid testbed. In a manner similar to services, they are defined here, and then attached to folders, networks, or groups as needed. The access to the resource is managed using some form of federation, the implementation of which is platform and resource-specific.

What this looks like in practice depends heavily on the resource, the platform(s), and the item being "attached" to. Attaching to folders makes the resource available to objects in that folder, including the groups. Attaching to groups allows that group to access the resource, regardless of folder context. Attaching to networks makes the resource available to devices on that network. These attachments are loosely defined by design, which enable a wide range of devices and labs to be specified and used with the same specification. That being said, there is definitely room to expand this, the details of which will be discussed in Future Work.

```
resources:
  resource-p:
    lab: "power-lab"           # REQUIRED    Name of the lab the resource
    ↪  is associated with
    resource: "transformer"    # REQUIRED    Name of the specific
    ↪ resource
```

Listing 4.7: Exercise Resources

Networks

The *networks* section defines the networks[2]that are used to connect instances in a virtual environment. There are two types of networks that can be formally defined: *unique* and *generic*. *Unique* networks are the equivalent of Singleton classes in object-oriented programming: they can only have a single instance, and all references to a given unique-type network will point to the same instance. They are typically used to specify shared networks, such as to provide Internet access, or for a King of the Hill competition. *Generic* networks create new network instances with identical configurations for each base folder instance that is made of them. This instance counter is global across folders. Each instance of a given folder will refer to the "global" value of the base at that index. For example: a generic network for instance 5 of folder "hidden" will be the same network as instance 5 of folder "workstations".

Networks currently can have the following configurations: *description*, *subnet*, *vlan*, and *vswitch*. These configurations will vary by the type of network and the platform being used. For instance, the *vswitch* configuration only applies for networks that will be used on the vSphere platform, not other platforms. In addition, some configurations, like Virtual Local Area Network (VLAN) tags, do not apply to all network types.

The *description* is a human-readable description of the network, though how it is used is platform-dependent. The *vlan* manually specifies the VLAN tag of a network, which is normally automatically generated. This only applies to unique-type networks, and must be a value less than 2000 to prevent

[2]Networks in this context are virtual networks at layers 2 or 3 of the OSI model[17]. It is assumed the reader has basic knowledge of modern Internet Protocol (IP) networking and Virtual LANs.

clashing with the generated tags. The *vswitch* specifies the vSwitch that will be used for that network, and only applies to the vSphere platform. Finally, the subnet number for each instance of a generic-type network will be incremented if *increment* is set to true. Cyber Defense Competition (CDC)s are the primary use case of this final configuration, as each Blue Team instance requires the same network configuration, just with a subnet bump.

The *subnet* configuration is used to specify the IP subnet that will be used for that network. It is specified using the standard notation for the IP version being used, which is dotted decimal for IP version 4 and hexadecimal octets for IP version 6. The notation used will implicitly determine the IP version of the subnet. The subnet is specified using standard CIDR notation, which consists of a slash followed by the number of bits for the subnet. Currently, subnets that are not in private[56] IP range will result in a warning, to prevent accidental use of public IP ranges when they are not intended.

```
networks:
  unique-networks:    # Networks that are instantiated once and only
   ↪ once. Think of them as singletons.
    network-label:           # REQUIRED    Unique label used to identify
   ↪ the network (Replace "network-label" with the name of the network)
      description: "blah"    # Optional    Human-readable description of
   ↪ network
      subnet: "x.x.x.x/x"    # Suggested   IP network address and mask:
   ↪ SUBNET-IP/CIDR
      vlan: 0                # Optional    VLAN tag of the network. Must
   ↪ be a value < 2000. [default: globally unique value > 2000]
      vswitch: "name"        # Optional    Name of virtual switch used
   ↪ for the network [default: set in infrastructure-config or
   ↪ VsphereInterface]
  generic-networks:   # New networks are created per instance of a
   ↪ folder
    # This instance counter is global across folders. Each instance of a
   ↪  given folder will refer to the "global" value of the base at that
   ↪  index.
    # Example: a generic network for instance 5 of folder "hidden" will
   ↪ be the same network as instance 5 of folder "workstations"
    network-label:
      description: "description"
      subnet: "x.x.x.x/x"
      vswitch: "vswitch name"
      increment: no          # Optional    Increment the subnet value for
   ↪  each unique instance created   [default: no]
```

Listing 4.8: Exercise Networks

Folders

Folders are an hierarchical assemblage of objects used to construct an exercise. These objects can be instances of *services*, *resources*, or folders. A given folder can have a number of configurations applied to it, with those not being supported simply skipped over, making extending this section trivial. There are two types of folders: *parent* and *base*. Parents are folders that contain other folders. Base folders contain services and resources.

For both types of folders, there are the following configurations: *group*, *master-group*, *description*, *enabled*, and *instances*. The group configuration gives the specified group permissions to view and use the folder and its contents, including any sub-folders. The master-group configuration is the same as for group with the addition of an edit permission, and is intended to specify the group that will be setting up the environment instances. The description is a human-readable description of the folder, the use of which is platform-dependent. Enabled is a flag that allows selective disabling of specific folders without having to comment them out, which is useful for large folders or for programmatic editing of specifications.

```yaml
folders:
  parent-folder:  # Folders that contain other folders (replace "parent-
    ↪ folder" with name of the folder)
    group: group-label           # Optional    User group that will have
    ↪ permissions to the folder
    master-group: group-label    # Optional    User group for the pre-
    ↪ deployment masters [default: group specified for the folder]
    description: "description"   # Optional    Human-readable description
    ↪  of folder
    enabled: yes                 # Optional    Flag to selectivly disable
    ↪  a folder, so changes can be easily tested [default: yes]
    instances: 10                # Optional    Same configurations as for
    ↪  base-type folders
```

Listing 4.9: Exercise Parent Folders

Instances is used to specify if multiple instances of a folder should be created. This essentially creates multiple copies of the folder in its entirety, including any sub-folders. The number of copies can be specified with an integer, or the size of the group configured for that folder. Finally, a prefix prepended to the names of the copies can be specified if desired.

Base folders have one additional configuration that differentiate them from parents: *services*. This configuration is a list of service instances that the folder will contain. Each instance has a required *service* configuration that specifies which service defined in the top-level *services* section will be used for that instance. In addition, there are three other configurations that can be specified: *instances*, *networks*, and *scoring*. *Instances* is the same as was described for folders. *Networks* is used to attach networks configured in the *networks* top-level section to the service's network interfaces. Finally, *scoring* is used to configure how that service will be scored or graded by a scoring engine. Scoring consists of ports and protocols to score, as well as a criteria file that contains a specification of how the service should be scored. This is as-yet undefined, but it was inspired by, and could easily derive from the syntax used by the Open Cyber Challenge Platform (OCCP)[64].

```
  base-folder:  # Folders that contain services (replace "base-folder" with name of the folder)
    group: group-label           # REQUIRED    User group that will have permissions to the folder
    master-group: group-label    # Optional    User group for the pre-deployment masters [default: group specified for the folder]
    description: "description"  # Optional    Human-readable description of folder
    enabled: yes                 # Optional    Flag to selectivly disable a folder, so changes can be easily tested [default: yes]
    instances:  # Optional    Makes folder a template that is copied N-times (NOTE: instances can also simply be an integer representing N)
      number: 10            # OPTION A    Number of instances = integer
      size-of: group-label  # OPTION B    Number of instances = Size of named group
      prefix: "prefix"      # Optional    String to prepend to named instance numbers [default: name of folder]
    services:   # REQUIRED    Define services that the base folder will contain
      service-instance-name:
        service: service-label  # REQUIRED    Label as defined in services
        instances: 10           # Optional    Same configurations as for base-type folders
        networks: ["subnet-a", "subnet-b"]    # Optional    Networks to attach the service instance to (Case sensitive!)
        provisioner-file: "file"  # Optional  Override provisioner configuration file for a service
        scoring:  # Optional    Scoring for the service (e.g a competition, or verification of homework)
          ports: [0]            # Suggested   Ports used for scoring
          protocols: ["proto"]  # Suggested   Protocols used for scoring
          criteria: "file.yaml" # REQUIRED    Criteria used to score the service
```

Listing 4.10: Exercise Base Folders

4.4 Infrastructure Specification

The infrastructure specification describes the infrastructure platform that will be used to create an environment. It consists of the list of platforms that will be used, and their corresponding configurations. Specifying the infrastructure configuration separately from the exercise improves portability, as a given user of the system can swap out their own configuration with the platforms they wish to use configured the way they want. There are two platforms currently specified: *vmware-vsphere*, and *docker*.

vSphere Configurations

The configurations specified in the *vmware-vsphere* section apply to the VMware vSphere platform. The *datacenter* configuration specifies what Datacenter in vSphere the environment will be created on. *Datastore* specifies the Datastore to use for the environment's VMs. The *template-folder* configuration specifies the path to the folder in the vSphere inventory containing the templates used to build the exercise. The *server-root* configuration specifies the path to a folder that should be considered the top-level folder for the server, which can be useful for large or multi-organization vSphere environments. Finally, *vswitch* specifies the name of a vSwitch that should be used as default for exercise networks that do not specify one.

```
vmware-vsphere:
  hostname: "hostname"             # REQUIRED    Hostname of the vCenter
    ↪ server
  port: 0                          # Suggested   Port used to connect to
    ↪ the vCenter server [default: 443]
  login-file: "vsphere.json"       # Suggested   Login information used
    ↪ to connect to the vCenter server [default: prompt user]
  datacenter: "datacenter name"    # Suggested   Name of the Datacenter
    ↪ on which to create environment
  datastore: "datastore name"      # Suggested   Name of Datastore to use
    ↪  for environment VMs
  template-folder: "folder path"   # REQUIRED    Path from server root to
    ↪  folder that contains VM templates
  server-root: "folder name"       # Suggested   Name of folder
    ↪ considored to be "root" for the platform
  vswitch: "vswitch name"          # Suggested   Name of vSwitch to use
    ↪ as default
  host-list: ["a", "b"]            # Optional    List of names of ESXi
    ↪ hosts to use [default: first host found in the datacenter]
```

Listing 4.11: vSphere Infrastructure Configurations

Docker Configurations

The *docker* section of the specification contains configurations for the Docker platform. While this platform is not yet implemented in a fully working fashion yet, the configurations are still defined as an example of how another platform would look in the infrastructure specification. The *url* configuration specifies how to connect to the Docker Engine client that will be managing and creating the Docker containers. *TLS* specifies whether Transport-Layer Security should be used to connect to the docker server. This option is exposed to deal with certificate signing issues commonly encountered when doing testing of a new environment, or with isolated labs such as RADICL. Finally, *registry* allows a Docker Registry server to be specified and configured. This has two options, *url* and *login-file*, that are used to specify the server and its login information. This is useful if a instructor wishes to load custom or cached images from a local Registry, or as an alternative to needing access to the Internet-located Docker Hub.

```yaml
docker:
  url: "host:port"         # Suggested   URL to the Docker server [
   ↪ default: unix:///var/run/docker.sock]
  tls: true                # Optional    Use TLS to connect to the Docker
   ↪  server [default: True]
  registry: # Optional
    url: "url://"          # REQUIRED  URL of the registry
    login-file: "r.json"   # REQUIRED  JSON file containing login
   ↪ information for the Docker registry server
```
Listing 4.12: Docker Infrastructure Configurations

Other Platforms

There are a few other platforms that have been roughly defined to demonstrate the use and flexibility of the specification. Amazon AWS as *amazon-aws* and Digital Ocean as *digital-ocean* are cloud platforms, while Microsoft Hyper-V as *hyper-v* is hardware-based virtualization (much like vSphere).

```
# Amazon Web Services
amazon-aws:
  cred-file: "path"     # REQUIRED  Path to file with AWS access
    ↪ credentials
  config-file: "path"   # Suggested   Path to file with service
    ↪ configurations

# Digital Ocean
digital-ocean:
  token-file: "path"    # REQUIRED  Path to file containing access token

# Microsoft Hyper-V Server
hyper-v:
  hostname: "hostname"    # REQUIRED    Hostname of the Hyper-V server
  port: 0                 # Suggested   Port used to connect to Hyper-V
    ↪ server
  login-file: "path"      # REQUIRED    Path to file containing login
    ↪ information for the Hyper-V server
  version: "v2"           # Suggested   Version of the Hyper-V API to
    ↪ use
  vswitch: "name"         # Suggested   Name of VirtualEthernetSwitch to
    ↪  use as default
```

Listing 4.13: Other Infrastructure Configurations

Chapter 5: Implementation

This chapter discusses the implementation of ADLES. It includes technology and design choices made, the design of the platform interface for vSphere, the execution process for the system when using the vSphere interface, and scripts developed to use the vSphere wrapper class. Finally, the discussion of the distribution and development of the ADLES project closes out the chapter.

5.1 Implementation Choices

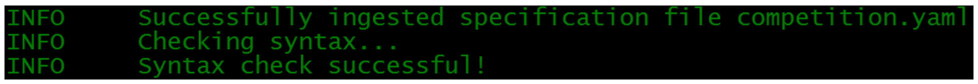
Language selection: Python

The Python language[62] is used to implement the ADLES system. It was selected for the following reasons:

- Platform-independence, which maps to the portability requirement.
- Extensive library ecosystem, which prevents duplication of work for tasks like parsing.
- Excellent documentation and strong community support.
- Strong Unicode support, mapping to the sharing requirement.
- Simple and easy to learn syntax, improving the ability to implement extensions.

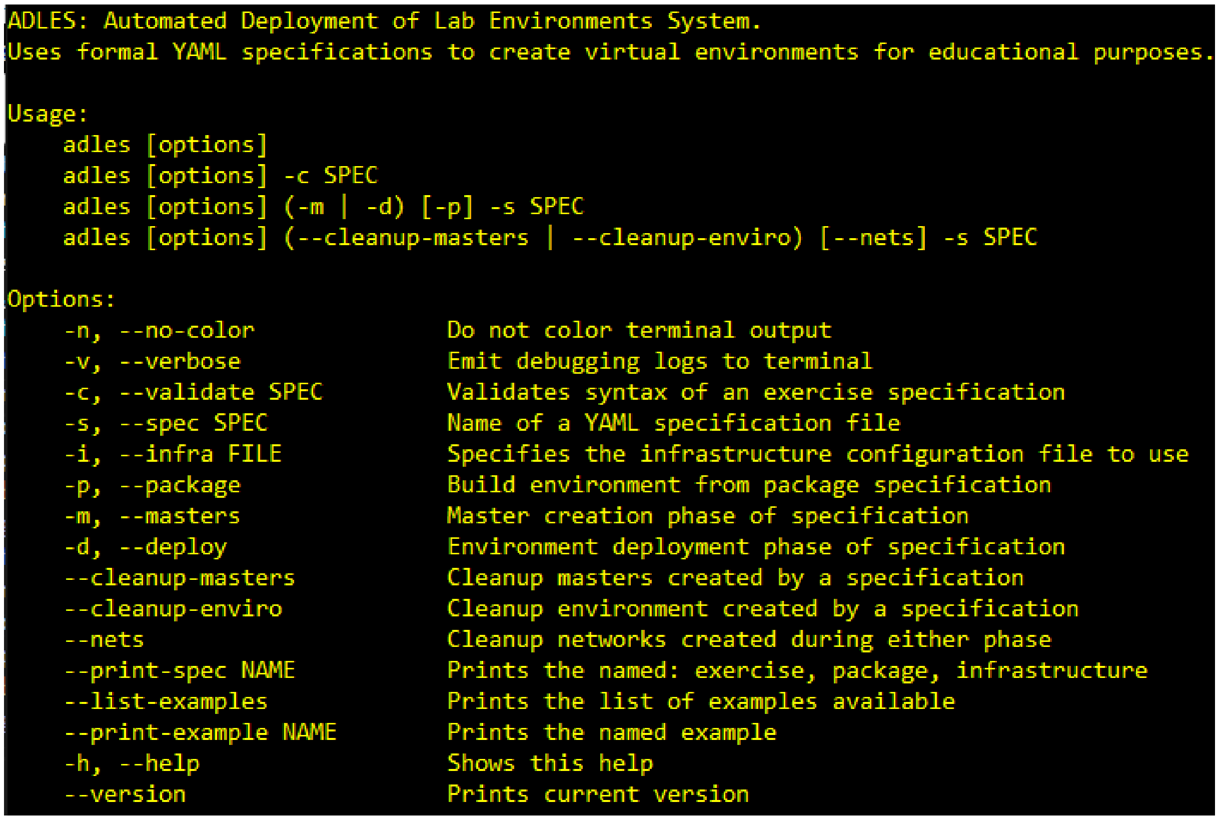
User Interface

The user interface is exclusively command-line. Usage for ADLES and the scripts is easily accessible by users with the "–help" command line argument. (fig. 5.2) Examples and specifications are bundled with the application, and can be output using the appropriate flags. Output is colored to aid usability by making errors or potential mistakes obvious, separate important output from debugging messages, and improve general look and feel. (fig. 5.1)



```
INFO     Successfully ingested specification file competition.yaml
INFO     Checking syntax...
INFO     Syntax check successful!
```

Figure 5.1: Colored output



```
ADLES: Automated Deployment of Lab Environments System.
Uses formal YAML specifications to create virtual environments for educational purposes.

Usage:
    adles [options]
    adles [options] -c SPEC
    adles [options] (-m | -d) [-p] -s SPEC
    adles [options] (--cleanup-masters | --cleanup-enviro) [--nets] -s SPEC

Options:
    -n, --no-color              Do not color terminal output
    -v, --verbose               Emit debugging logs to terminal
    -c, --validate SPEC         Validates syntax of an exercise specification
    -s, --spec SPEC             Name of a YAML specification file
    -i, --infra FILE            Specifies the infrastructure configuration file to use
    -p, --package               Build environment from package specification
    -m, --masters               Master creation phase of specification
    -d, --deploy                Environment deployment phase of specification
    --cleanup-masters           Cleanup masters created by a specification
    --cleanup-enviro            Cleanup environment created by a specification
    --nets                      Cleanup networks created during either phase
    --print-spec NAME           Prints the named: exercise, package, infrastructure
    --list-examples             Prints the list of examples available
    --print-example NAME        Prints the named example
    -h, --help                  Shows this help
    --version                   Prints current version
```

Figure 5.2: Commandline Help

5.2 The vSphere Platform Interface

The design of the vSphere platform interface follows the generic platform interface design outlined in section 3.3. Upon initialization, the VsphereInterface instantiates a Vsphere object, which makes the necessary calls to pyVmomi to connect to a vCenter server instance that manages the environment. Once initialized, calls to the Interface's functions will result in the desired operations being performed on the vSphere environment. These will be described in detail in the following section on system execution.

The vSphere Python Library

The pyVmomi Software Development Kit (SDK) is a library for vSphere that allows management of a vSphere environment using native Python bindings[63]. The bindings it provides are a thin wrapper around vSphere's VMOMI API, essentially performing the task of converting Python objects and function calls into HTTP requests to vSphere, and vice-versa. It is fast, reliable, and well-supported by both the community and VMware itself, making it the library of choice for ADLES interface with VMOMI.

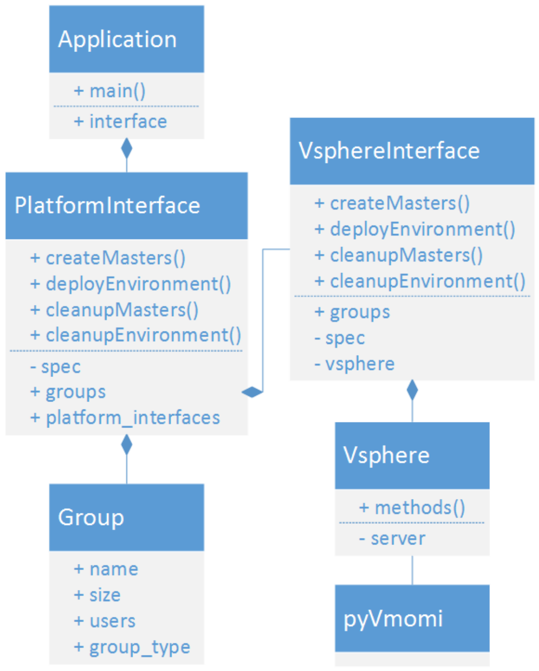


Figure 5.3: vSphere platform interface design

The vSphere class

The implementation of ADLES requires a level of abstraction that pyVmomi does not provide. Therefore, a wrapper API for pyVmomi had to be developed to provide this abstraction and simplify ADLES development. This was accomplished by means of a class representing the vSphere environment. Class instances maintain the vSphere environment state and status, and provide a robust set of methods to perform actions in the environment.

5.3 System Execution using the vSphere Platform Interface

Mastering Phase

1. Virtual networks are checked, and created if necessary. *Base* and *generic* networks are treated the same as *unique* networks during this phase.

2. Folder structure for the Master instances is created.

3. Permissions for specified group and master-group are applied to folder structure.

4. Instances are created from the specified templates.

5. A Snapshot is taken of the instance immediately post-clone, to allow those configuring the Master instances to roll back to a "clean" start if mistakes are made.
6. Network interfaces on the newly cloned VMs are configured per the specification.
7. Other configurations are performed, if specified in the services' definition.

Deployment Phase

1. Master instances are verified and prepared for deployment. For each instance:
   (a) Existence of the instance is verified.
   (b) If the instance is powered on, it is cleanly powered off.
   (c) A Snapshot of the instance is taken. This enables users to reset machines to the start of the exercise, in case of catastrophic failure or compromise by a Red Team.
   (d) The instance is converted to an immutable *template*. This prevents accidental changes to the master instances during or after the deployment stage. It also ensures they are not modified for the purposes of saving the exercise for future reuse, and for storage with the exercise results.
2. Folder structure for the exercise is created.
3. Permissions are applied.
4. Instances are cloned from their respective Master instances, multiple times if the *instances* field is configured as such.
5. If *generic* networks are set, they are checked and created if necessary, and the corresponding virtual Network Interface Card (NIC)s on the instance are updated.

5.4 vSphere scripts

As part of the development process, a number of basic scripts were developed using the vSphere wrapper API. The original purpose of these scripts was to provide basic automation for a number of tasks in RADICL. This was to get real-world testing of many of the core facilities ADLES relies on, garner user feedback, and make improvements before the core of ADLES was fully implemented. This follows the UNIX philosophy[55] of many small tools that each specialize in a single or small set of tasks in lieu of one monolithic tool that does many tasks. This ensures modularity of the system and that loose coupling of components is maintained throughout the development process. The following scripts were developed:

- cleanup-vms: Removes and destroys the files of VMs.
- clone-vms: Clones of VMs.
- vm-power: Performs power operations on VMs.
- vsphere-info: Provides information about objects in a vSphere environment, including VMs, folders, datastores, and vsphere itself.

The script implementations are designed to be simple and straightforward, and utilize library functionality wherever possible. Several examples can be found in the appendices. Here is an example of what the start of a typical script looks like:

```python
def main():
    args = docopt(__doc__, version=__version__, help=True)
    server = script_setup('vsphere_info.log', args, (__file__,
     ↪ __version__))
```

Listing 5.1: Script Setup

5.5 ADLES Project

The system is designed to be used, modified, and extended by anyone in the educational community. Simply dumping the source code for ADLES on a public FTP server with no documentation will not effectively enable that. The goal is to setup the project such that future contributors to the project can focus on extending the system, and not setting up tests, development environments, user input/output, and the like.

Project License: Apache 2.0

The goal of ADLES is to allow the widest possible use, to establish a large user base and drive creation of exercises. With this in mind, the Apache License[4], version 2.0, was chosen to license ADLES code, specifications, documentation, and examples. This well-known open-source license enables educators to freely use the project, while still requiring attribution, protecting the University's patent rights, and encouraging contribution of any improvements or extensions back to the community. Furthermore, the license's commercial use provisions provide flexibility for institutions with proprietary platforms or strict institutional requirements that still wish to use the system.

Source code repository

Since this project has the possibility of future development, version control and cloud-based hosting were implemented from the beginning. Version control is managed using a *git* repository, and this repository is publicly available on the open-source hosting service, GitHub[33]. This version control and hosting method was selected for the following reasons:

1. Cross-platform and cloud-hosted.
2. Well-known and respected in academia and industry.
3. Useful third-party integrations. (Travis CI automated testing, CodeClimate static code analysis, VersionEye dependency version checking)
4. Author's personal experience with the tool and platform.

Setup and Installation

Setup and installation of ADLES is accomplished using the "pip" tool included in most Python installations. Pip utilizes a public repository of Python packages that can be installed using the tool called Python Package Index (PyPI)[53]. Using pip to install a package will download the package and its dependencies from PyPI, unpack it into the Python module index, and compile and provide system links to any scripts specified, regardless of platform. This makes installation of complicated libraries and scripts simple and painless for users and developers. Publishing ADLES to PyPI as a package[34], therefore, matches the portability and ease-of-use requirements with aplomb.

Chapter 6: Example Scenarios

This chapter describes several real-world examples of hands-on educational scenarios that illustrate how the specifications are used. Two examples, a tutorial and a competition, will be discussed in detail. Two more tutorials will be touched on briefly. Note that for all scenarios, the examples given are only snippets taken from the full specification, and are therefore missing some details. Refer to the appendices for complete versions of the specifications.

6.1 Example 1: Penetration Testing Tutorial

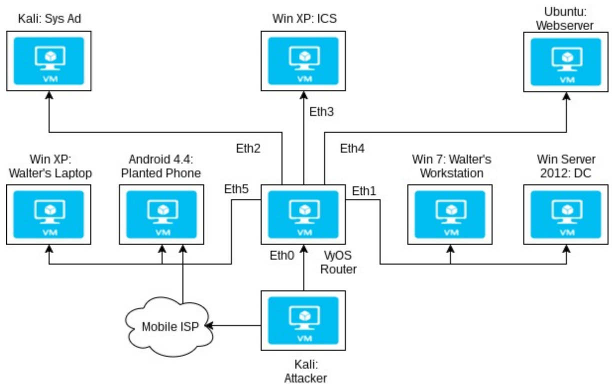


Figure 6.1: Network diagram of the penetration testing tutorial

This tutorial teaches students how to properly perform a legal penetration test[8]. The tutorial begins with background on history, legality, and pre-engagement customer contact. Then, students login to the environment and begin the exercise. The goal is a heavily-firewalled ICS network. The first stage consists of information gathering using tools such as ls, Nmap, cat, whois, nslookup, and Google, then threat modeling and vulnerability analysis to determine weaknesses that can be exploited. The second stage has students exploit the identified weaknesses, establish persistence, extract useful information, and continue pivoting around the network until the goal is reached.

Setup of this exercise was extremely time-consuming, taking over 120 hours total. Furthermore, when a bug was discovered in the deployed system, it took one of the authors several hours of intense effort to get it fixed on all the instances before the tutorial was scheduled to begin.[1]Deploying this tutorial using ADLES would have saved at least 10-20 hours of the student's time, if not more. Using ADLES, the last-minute bug-fix would have simply involved un-templating a master instance, fixing the bug, then re-running the deployment, taking only minutes instead of hours.

[1]This information comes from my personal discussions with the authors of the exercise. It is not in the document cited for this tutorial.

Specification

Metadata for the examples are simple and straightforward. Since the metadata for all of the examples is roughly the same, it will not be shown or discussed in the next two examples.

```yaml
metadata:
  name: "Penetration Testing"
  description: "Tutorial on properly conducting a penetration test"
  activity: "CS 439 - Applied Security Concepts"
  prefix: "CS439-TR03-TU16"
  date-created: "2017-03-29"
  version: "0.3.4"
  infra-file: "infra.yaml"
  folder-name: "CS-439/TR-03/TU-16"  # CS 439, Tutorial Round 3,
   ↪ Tutorial 16
```

Listing 6.1: Pentest Tutorial - metadata

There are two groups defined: an AD group of students in CS439, and a list of the user names of the instructors (and TAs) creating the tutorial. These will be used to apply permissions in the folders section later on.

```yaml
groups:
  Students:
    ad-group: "CS439 Students"
  Instructors:
    user-list: ["brau", "ocke"]
```

Listing 6.2: Pentest Tutorial - groups

For the services, it should be noted that since the default note is preserved from the base template, notes do not have to be defined here with default credentials. Additionally, since the students directly interact with only one service during the tutorial, they do not have the ability to view the other services, and thus the notes for those services are of no use to them.

```yaml
services:
  kali:   # What the students will be using
    template: "Kali Linux 2016.2 (64-bit)"
  ics:    # Industrial Control System
    template: "Windows XP SP0"
  web:    # Apache webserver
    template: "Ubuntu Server 14.04 (64-bit)"
  laptop: # Walter's Laptop
    template: "Windows XP SP2"
  phone:  # Planted phone
    template: "Android 4.4"
```

Listing 6.3: Pentest Tutorial - services

There are seven networks defined for this example, all of them generic-type. The subnets for the networks associated with the fictitious company are all in the 192.168/16 range, while the subnet for the ”cellular network” is in the 172.16/16 range. The cellular network is changed from its value in the original tutorial, which was a publicly routable IP address. This was changed for simplicity and security.

```yaml
networks:
  generic-networks:
    Attacker-net:
      description: "Used by the attacker"
      subnet: "192.168.0.0/24"
      increment: yes
    Mobile-ISP:
      description: "Attacker <-> Phone"
      subnet: "172.16.0.0/19"  # Unusual subnet mask is intended to make
  ↪  subnets more "random"
      increment: yes
    Workstations:
      description: "This is the internal corporate network"
      subnet: "192.168.1.0/24"
```

Listing 6.4: Pentest Tutorial - networks

The students each have a Kali Linux workstation they will use for the entirety of the tutorial. These are located in the student-workstations folder.

```yaml
folders:
  student-workstations:
    description: "Workstations visible to and used by students during
  ↪ the tutorial"
    group: Students
    master-group: Instructors
    instances:
      number: 15
      prefix: "WS-"
    services:
      attacker:
        service: kali
        networks: ["Attacker-net", "Mobile-ISP"]
```

Listing 6.5: Pentest Tutorial - student-workstations folder

Since the students are not supposed to know the layout of the network at the start of the tutorial, these folders are kept separate from the workstation folders. By limiting the permissions of the folder to only the Instructors, the folder and its services are essentially hidden from the students, preventing them from ”peeking” at the VMs they are supposed to find through network reconnaissance.

```
exercise-environemnt:
  description: "Systems the students will explore during the exercise,
  ↪  but should not see"
  group: Instructors
  systems:
    group: Instructors
    instances:
      number: 15
      prefix: "HIDDEN-ENVIRONMENT-"
    services:
      exercise-router:
        service: router
        networks: ["Attacker-net", "Workstations", "IT", "ICS", "Web-
  ↪ Services", "WiFi"]
      sysad:
        service: kali
        networks: ["IT"]
```

Listing 6.6: Pentest Tutorial exercise-environment folder

6.2 Example 2: Cyber Defense Competition

This example replicates a real-world cyber defense competition. Specifically, it attempts to replicate the 2016 PRCCDC[50]. The setup roughly follows the network setup of competition, based on the author's personal experience attending the competition. Blue Teams receive points for service up-time, which is scored approximately every minute by a scoring engine known as the ”score-bot”. Successful attacks by Red Team will impact this up-time, and instill penalty points on the Blue Team attacked.

Specification

There are three groups defined for the competition: Blue Teams, which are student competitors, Red Team, which are the attacking experienced industry professionals, and Black Team, which is in charge of managing the competition infrastructure. The Blue Teams are Template groups, and will be used later when the folders for the teams are described.

```yaml
groups:
  Blue Team:  # Defenders
    instances: 14
    filename: "examples/blue-teams.json"
  Red Team:   # Attackers
    ad-group: "Red Team"
  Black Team: # Infrastructure team
    ad-group: "Black Team"
```

Listing 6.7: Competition - groups

The services for the competition are defined in the same manner as the tutorial example. There are several types of ”host” workstations the competitors will use to access and interact with the competition environment. There are also a number of different services, such as the scoring engine, router, and various services the Blue Teams will be defending.

```yaml
services:
  score-bot:
    template: "PRCCDC Score Bot"
  evil-host:
    template: "Kali 2.0 (64-bit)"
  host-type-a:
    template: "Windows 7 (64-bit)"
```

Listing 6.8: Competition - services

There are a number of networks defined for the competition, three of which are unique, and two of which are generic. The competition unique network is the backbone that connects the Red Teams, Blue Teams, score-bots, competition services, and the Internet.

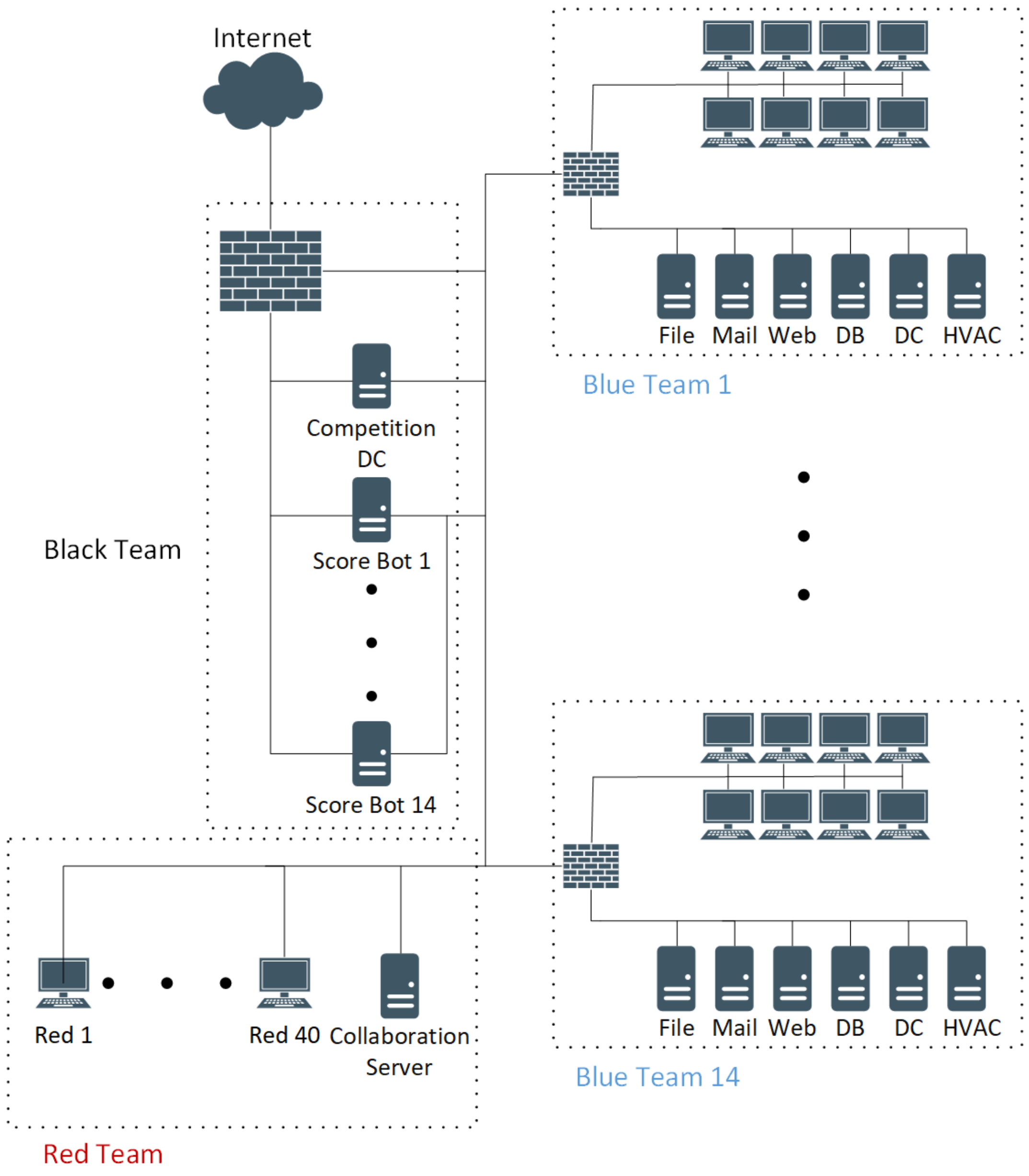


Figure 6.2: Network diagram of the cyber defense competition scenario

```
networks:
  unique-networks:
    competition:
      subnet: "172.30.0.0/16"
      vswitch: "competition_vswitch"
```

Listing 6.9: Competition - networks

The external folder contains the infrastructure needed for the competition, and is managed by the Black Team. Of note is the score-bot service. It has fourteen instances, equal to the number of Blue Teams competing, and has two network interfaces: competition and management. The management network enables reliable and secure collection of scoring information, as well as a back-channel method for the score-bots to communicate. To prevent Blue Teams from white-listing, the score-bots will randomly change IP addresses and the team they are scoring every minute, and that coordination is done over the management network.

```
folders:
  external:
    group: "Black Team"
    description: "The open competition network with red teams and
    ↪ scorebots"
    services:
      edge-router:
        description: "Connection to Internet"
        service: firewall
        networks: ["outside-world", "competition", "management"]
      score-bot:
        description: "Used to score teams"
        service: score-bot
        networks: ["competition", "management"]
        instances:
          number: 14
          prefix: "Scorebot "
```

Listing 6.10: Competition - external folder

Each student Blue Team has an instance of the blue-team folder that matches their team number. This folder contains the services they must keep running and protect from the Red Team, and the workstations they use to access the environment and manage the services. There are eight workstations, six Windows and two Linux, along with a gateway router running VyOS, and corucorpia of OSes that make up their services. The gateway has three network connections, linking the Blue Team's hosts with their services, and both to the wider competition network and the Internet.

```yaml
blue-team:
  instances:
    number: 14
    prefix: "Blue Team "
  group: "Blue Team"
  services:
    gateway:
      description: "Gateway connecting Blue teams to main competition
↪ network"
      service: firewall
      networks: ["competition", "blue-hosts", "blue-services"]
    windows-hosts:
      description: "Windows hosts that the blue team members use to
↪ access their services"
      instances:
        prefix: "Host "
        number: 6
      service: host-type-a
      networks: ["blue-hosts"]
```

Listing 6.11: Competition - blue-team folder - hosts

The website and database services (fig. 6.12) are prime examples of the Blue Team's services. The website is directly scored by the score-bot, hence the scoring criteria. The database is not scored directly, but must remain up for the website to access when providing the required content for scoring.

```yaml
    web:
      description: "Web server hosting the team's page"
      service: web-server
      networks: ["blue-services"]
      scoring:
        ports: [80, 443]
        protocols: ["http", "https"]
        criteria: "criteria-file.yaml"
    db:
      description: "Database"
      service: database-server
      networks: ["blue-services"]
```

Listing 6.12: Competition - blue-team folder - web services

The attacking Red Team has two members dedicated to a Blue Team, ten ”meta” team members that apply the same attacks to all Blue Teams, and two members, a captain and co-captain, that lead the team. Each member has a Kali Linux workstation from which to launch attacks, and access to a collaboration server. The collaboration server has a text communication server the team can use to communicate, along with a special server the ”meta” Red Team can use to share persistence gained on Blue Teams.

```
red-team:
  group: "Red Team"
  services:
    attacker-hosts:
      instances:
        number: 40
        prefix: "Attacker "
      description: "Host used by Red Team to attack Blue teams"
      service: evil-host
      networks: ["competition"]
    collaboration-server:
      description: "Server for Red Team collaboration, including
↪ Armitage, SFTP, and IRC"
      service: web-server
      networks: ["competition"]
```

Listing 6.13: Competition - red-team folder

6.3 Additional examples

There are three additional examples that further demonstrate the capabilities of ADLES: Network Firewalls tutorial, Spoofing tutorial, and Network Security Visualization experiment. The tutorials are derived from real tutorials created and performed by students in the CS 439: Applied Security Concepts course at the University of Idaho, while the experiment is a made-up example of data collection for security visualization research. Since their specifications are similar to the tutorial described earlier, only the content of the examples will be discussed in this section. Refer to the appendices for the full specification for each tutorial.

Network Firewalls

The learning objectives of the Network Firewalls tutorial are to teach students how to manage firewalls through both graphical and command-line interfaces, and how to write good firewall rules[30]. The firewalls used are PFSense, which is configured through a graphical user interface, and VyOS, which is configured using a command-line interface. There are Windows and Linux servers and hosts that are used to test configurations and rules applied to the firewalls.

Interestingly, the scenario was connected to the same layer 2 network, with two subnets configured to share that network. (With each student having their own network, of course) The authors were asked why they used this setup for firewalls, which typically have at least two separate layer 2 networks. The reason given was difficulties configuring port-groups on the vSphere infrastructure. Adding and configuring a second port-group for each student and on each VM would have taken several hours, given the students' limited experience with the platform and sluggishness of the lab's clients. Having two interfaces on the same port-group, just with different subnets, significantly reduced the time it took to setup the tutorial. This is an example of where corners are understandably being cut, sacrificing fidelity and realism. Using ADLES, having two separate networks for this tutorial simply involves adding an extra generic-type network, which takes about a minute to change in the specification. This is an example of where use of ADLES can save significant educator burden and improve educational value.

Spoofing

The learning objectives of the Spoofing tutorial are to teach students how to perform ARP and DNS spoofing Man-in-the-Middle attacks to steal credentials and exploit machines, and how to implement effective mitigations preventing them[67]. It consists of a Windows host, a Kali Linux workstation, a Ubuntu server, and a VyOS router. Students use the Kali workstation to attack the Windows host using

ARP spoofing, and spoof the DNS provided by the Ubuntu server.

Experiment

This example's purpose is to generate a sizable corpus of realistic network traffic for network security visualization research. There are three main groups of services: clients, servers, and Internet. The services group consists of Windows IIS and Linux Apache web servers that are load balanced by an NGINX instance. The clients are Windows and Linux clients that access the web services hosted by the servers. Finally, the Internet is a simulation of services found on the Internet, such as network time and domain name resolution.

Chapter 7: Application and Results

In this chapter, the results of the application of ADLES to the example scenarios described in the previous chapter will be presented and discussed. It begins with a description of the testing environment used, presents the results of using ADLES to create and deploy the competition and three tutorial scenarios, and closes with a discussion of the results. For the two primary tutorials, the truncated commandline output will be shown, followed by the resulting environment in vSphere.

7.1 Description of Testing Environment

The tests runs of ADLES using the example scenarios described earlier were run on a Dell R620 server hosted by the University of Idaho CS department. This server is equipped with a six-core Intel Xeon processor, sixteen gigabytes of RAM, and one terabyte of disk storage. It is running VMware ESXi 6.5 as the hypervisor for vSphere, and vCenter Server 6.5 in a VM on the hypervisor. Screen shots are of the resulting views displayed using the HTML5 Web Client included with vCenter 6.5.

The tests were executed from a desktop computer running Windows 10 x64 build 14393, and connected to the testing server over a high-speed Ethernet link. There were two environments and versions of Python used for testing:

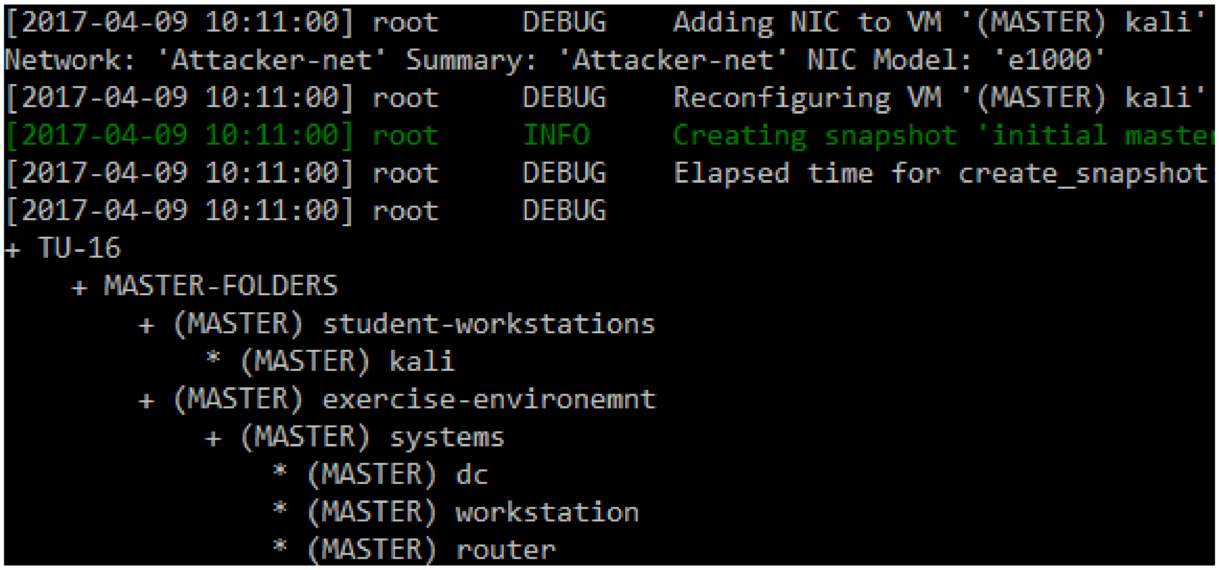
- Bash on Ubuntu on Windows environment, which will be referred to as the ”Bash environment”, running Python 3.4.3. This essentially a Ubuntu 14.04 sandbox, and acted as a fast equivalent of a Linux machine for testing purposes. The environment is the prompt with orange text and a black background.

- The standard Windows 10 environment, which will be referred to as the ”Windows environment”, running Python 3.6.0. The Windows CMD prompt (green text, transparent-black background) and Windows PowerShell (blue background) terminal interfaces were both used to test this environment, to ensure the system worked as expected under a variety of usage scenarios.

Finally, it should be noted that some testing not shown here was done using Python 2.7.6, to ensure the system has reasonable backward compatibility with the older version of the language. Efforts have been made throughout development to maintain compatibility, but it is not a priority, and will be deprecated in the future when multi-lingual specifications are supported and performance enhancements are made.

7.2 Results for Penetration Testing Tutorial

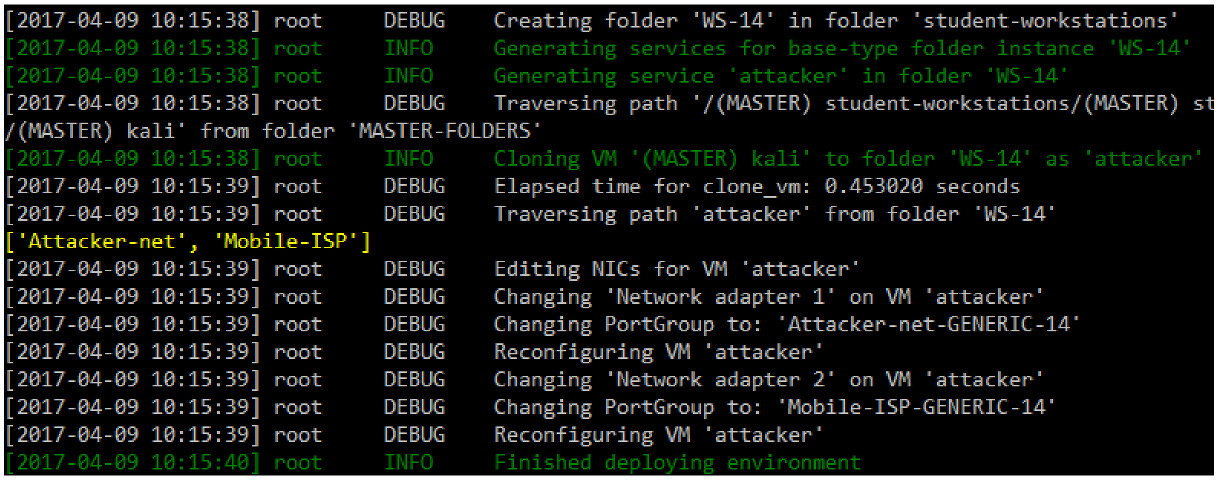
The test runs for the penetration testing tutorial were run using Bash environment, with ”verbose” debugging output enabled. Figure 7.1 is the command line output from ADLES while creating the Master instances during the Mastering phase. The addition of virtual NICs to the final VM can be seen, followed by a snapshot once configuration is complete. The resulting file structure is output to debugging, so the user can see what was created if they desire. Even if verbose output is disabled, the user can simply open the log file and examine the results, without having to connect to a vSphere console.



```
[2017-04-09 10:11:00] root     DEBUG    Adding NIC to VM '(MASTER) kali'
Network: 'Attacker-net' Summary: 'Attacker-net' NIC Model: 'e1000'
[2017-04-09 10:11:00] root     DEBUG    Reconfiguring VM '(MASTER) kali'
[2017-04-09 10:11:00] root     INFO     Creating snapshot 'initial maste
[2017-04-09 10:11:00] root     DEBUG    Elapsed time for create_snapshot
[2017-04-09 10:11:00] root     DEBUG
+ TU-16
    + MASTER-FOLDERS
        + (MASTER) student-workstations
            * (MASTER) kali
        + (MASTER) exercise-environemnt
            + (MASTER) systems
                * (MASTER) dc
                * (MASTER) workstation
                * (MASTER) router
```

Figure 7.1: Commandline output from pentest tutorial mastering phase

Figure 7.2 is the output from the Deployment phase. Since this example uses generic networks, these are created as portgroups on vSphere. Once the Master VMs are cloned, their virtual NICs are reconfigured to use the configured generic networks instead of the dummy networks used during the Mastering phase. The ”traversing path” output is ADLES ensuring the proper instances are being used, since vSphere will by default grab the first instance with the given name anywhere on the infrastructure.



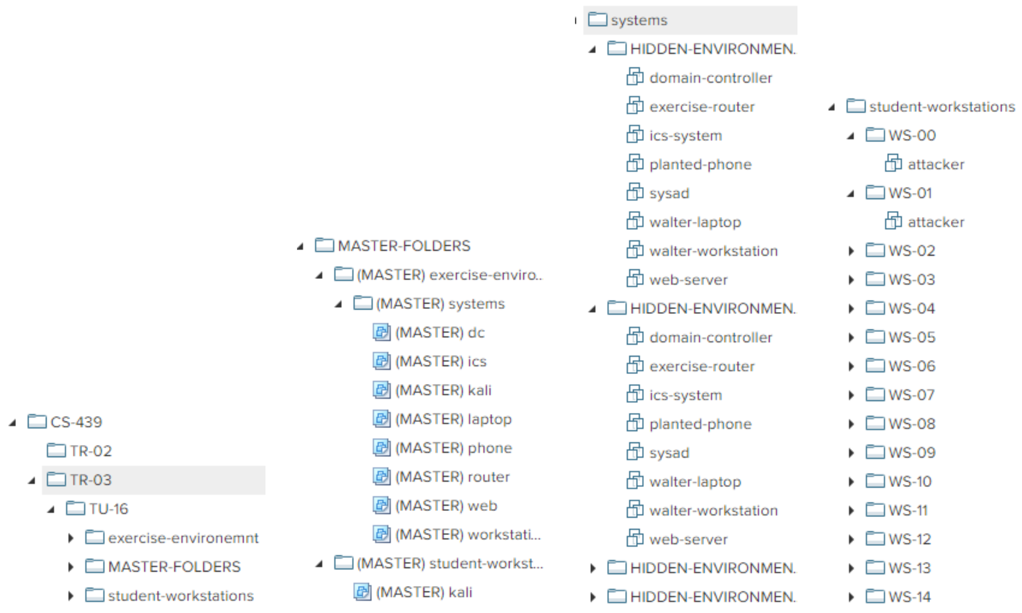
```
[2017-04-09 10:15:38] root     DEBUG    Creating folder 'WS-14' in folder 'student-workstations'
[2017-04-09 10:15:38] root     INFO     Generating services for base-type folder instance 'WS-14'
[2017-04-09 10:15:38] root     INFO     Generating service 'attacker' in folder 'WS-14'
[2017-04-09 10:15:38] root     DEBUG    Traversing path '/(MASTER) student-workstations/(MASTER) st
/(MASTER) kali' from folder 'MASTER-FOLDERS'
[2017-04-09 10:15:38] root     INFO     Cloning VM '(MASTER) kali' to folder 'WS-14' as 'attacker'
[2017-04-09 10:15:39] root     DEBUG    Elapsed time for clone_vm: 0.453020 seconds
[2017-04-09 10:15:39] root     DEBUG    Traversing path 'attacker' from folder 'WS-14'
['Attacker-net', 'Mobile-ISP']
[2017-04-09 10:15:39] root     DEBUG    Editing NICs for VM 'attacker'
[2017-04-09 10:15:39] root     DEBUG    Changing 'Network adapter 1' on VM 'attacker'
[2017-04-09 10:15:39] root     DEBUG    Changing PortGroup to: 'Attacker-net-GENERIC-14'
[2017-04-09 10:15:39] root     DEBUG    Reconfiguring VM 'attacker'
[2017-04-09 10:15:39] root     DEBUG    Changing 'Network adapter 2' on VM 'attacker'
[2017-04-09 10:15:39] root     DEBUG    Changing PortGroup to: 'Mobile-ISP-GENERIC-14'
[2017-04-09 10:15:39] root     DEBUG    Reconfiguring VM 'attacker'
[2017-04-09 10:15:40] root     INFO     Finished deploying environment
```

Figure 7.2: Commandline output from pentest tutorial deployment phase

The resulting environment

Figure 7.3 contains snips of the environment folders in the vSphere console. On the far left, the directory structure ”CS-439/TR-03/TU-16”, is the same as that listed in the specification. The exercise-environment and student-workstations parent-type folders can be seen, in addition to the folder containing the Master instances. Second from the left are the Master instances converted to templates. Templates are denoted in vSphere with a light blue icon instead of the white squares used for regular VMs. Third from the left is the expanded systems parent-type folder. This has fifteen ”HIDDEN-ENVIRONMENT-XX” folders, each of which contain eight instances, which matches the specification. Finally, fourth from the left is the expanded student-workstations parent-type folder. This contains fifteen ”WS-XX” folders, each of which has a single Kali instance, matching what was defined in the specification for the scenario.

If all went well, the deployed instances should match their Master instances in all aspects except their names, network configurations, their template status, and potentially the host they are deployed on. In Figure 7.4, a deployed instance (left) is compared with the Master template instance it came from (right). The Guest OS, VM hardware version, number of CPUs and RAM, and tools status are all the same. The only differences are the name, network configurations, and template status. Template status can be seen by the status window to the left of the system information, with the template being a light blue icon, and the VM being a window with the text ”powered off”. The Master instance has the ”(MASTER)” prefix, while the deployed instance has the name defined in the folder in the specification. The Master's network configurations are attached to portgroups matching the networks defined in the networks section of the



```
CS-439
  TR-02
  TR-03
    TU-16
      exercise-environemnt
      MASTER-FOLDERS
      student-workstations
```

```
MASTER-FOLDERS
  (MASTER) exercise-enviro..
    (MASTER) systems
      (MASTER) dc
      (MASTER) ics
      (MASTER) kali
      (MASTER) laptop
      (MASTER) phone
      (MASTER) router
      (MASTER) web
      (MASTER) workstati...
  (MASTER) student-workst...
    (MASTER) kali
```

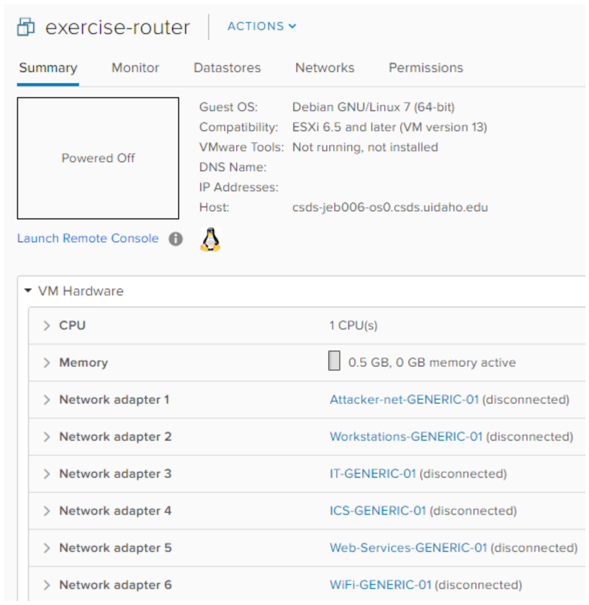
```
systems
  HIDDEN-ENVIRONMEN.
    domain-controller
    exercise-router
    ics-system
    planted-phone
    sysad
    walter-laptop
    walter-workstation
    web-server
  HIDDEN-ENVIRONMEN.
    domain-controller
    exercise-router
    ics-system
    planted-phone
    sysad
    walter-laptop
    walter-workstation
    web-server
  HIDDEN-ENVIRONMEN.
  HIDDEN-ENVIRONMEN.
```

```
student-workstations
  WS-00
    attacker
  WS-01
    attacker
  WS-02
  WS-03
  WS-04
  WS-05
  WS-06
  WS-07
  WS-08
  WS-09
  WS-10
  WS-11
  WS-12
  WS-13
  WS-14
```

Figure 7.3: Environment results from penetration testing tutorial

specification. The deployed instance instead has the GENERIC portgroups that were created during deployment, based on the configurations in networks.

Figure 7.5 shows two deployed instances, an attacker Kali VM, and a planted-phone Android VM. These are further examples of what deployed instances look like.



exercise-router ACTIONS

Summary Monitor Datastores Networks Permissions

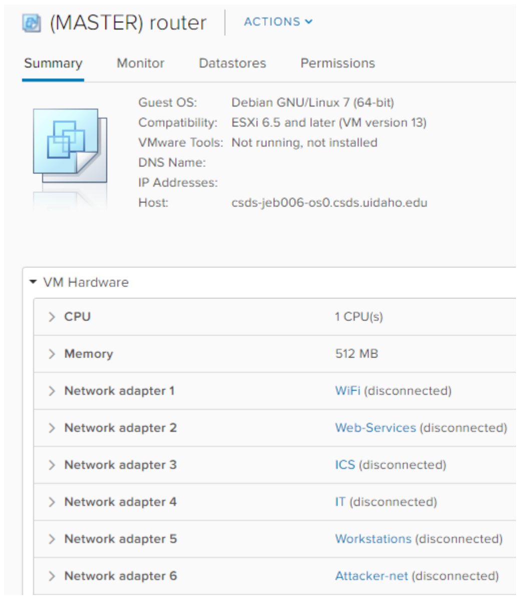
Powered Off

Guest OS:	Debian GNU/Linux 7 (64-bit)
Compatibility:	ESXi 6.5 and later (VM version 13)
VMware Tools:	Not running, not installed
DNS Name:	
IP Addresses:	
Host:	csds-jeb006-os0.csds.uidaho.edu

Launch Remote Console

VM Hardware

CPU	1 CPU(s)
Memory	0.5 GB, 0 GB memory active
Network adapter 1	Attacker-net-GENERIC-01 (disconnected)
Network adapter 2	Workstations-GENERIC-01 (disconnected)
Network adapter 3	IT-GENERIC-01 (disconnected)
Network adapter 4	ICS-GENERIC-01 (disconnected)
Network adapter 5	Web-Services-GENERIC-01 (disconnected)
Network adapter 6	WiFi-GENERIC-01 (disconnected)



(MASTER) router ACTIONS

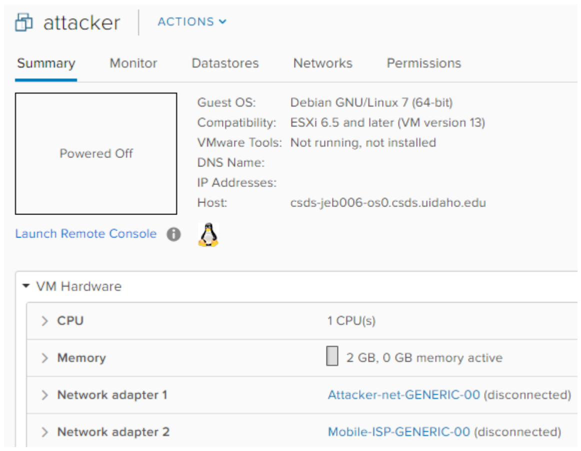
Summary Monitor Datastores Networks Permissions

Guest OS:	Debian GNU/Linux 7 (64-bit)
Compatibility:	ESXi 6.5 and later (VM version 13)
VMware Tools:	Not running, not installed
DNS Name:	
IP Addresses:	
Host:	csds-jeb006-os0.csds.uidaho.edu

VM Hardware

CPU	1 CPU(s)
Memory	512 MB
Network adapter 1	WiFi (disconnected)
Network adapter 2	Web-Services (disconnected)
Network adapter 3	ICS (disconnected)
Network adapter 4	IT (disconnected)
Network adapter 5	Workstations (disconnected)
Network adapter 6	Attacker-net (disconnected)

Figure 7.4: Comparison of master template and one of the generated pentest routers



attacker ACTIONS

Summary Monitor Datastores Networks Permissions

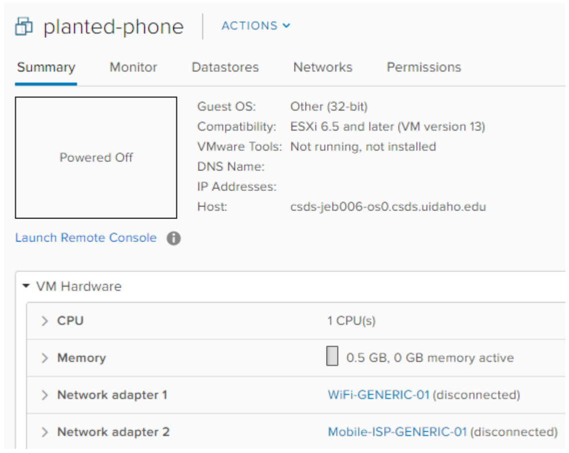
Powered Off

Guest OS:	Debian GNU/Linux 7 (64-bit)
Compatibility:	ESXi 6.5 and later (VM version 13)
VMware Tools:	Not running, not installed
DNS Name:	
IP Addresses:	
Host:	csds-jeb006-os0.csds.uidaho.edu

Launch Remote Console

VM Hardware

CPU	1 CPU(s)
Memory	2 GB, 0 GB memory active
Network adapter 1	Attacker-net-GENERIC-00 (disconnected)
Network adapter 2	Mobile-ISP-GENERIC-00 (disconnected)



planted-phone ACTIONS

Summary Monitor Datastores Networks Permissions

Powered Off

Guest OS:	Other (32-bit)
Compatibility:	ESXi 6.5 and later (VM version 13)
VMware Tools:	Not running, not installed
DNS Name:	
IP Addresses:	
Host:	csds-jeb006-os0.csds.uidaho.edu

Launch Remote Console

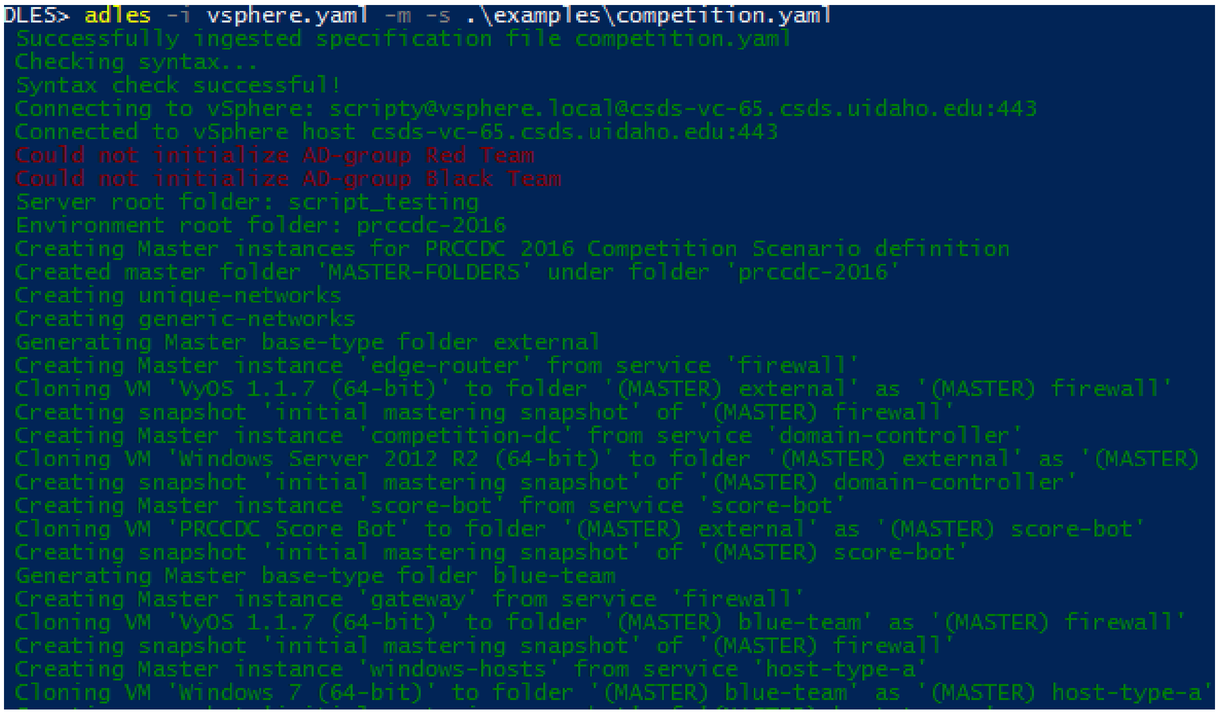
VM Hardware

CPU	1 CPU(s)
Memory	0.5 GB, 0 GB memory active
Network adapter 1	WiFi-GENERIC-01 (disconnected)
Network adapter 2	Mobile-ISP-GENERIC-01 (disconnected)

Figure 7.5: Two examples of generated virtual machines for the pentest tutorial

7.3 Results for Cyber Defense Competition

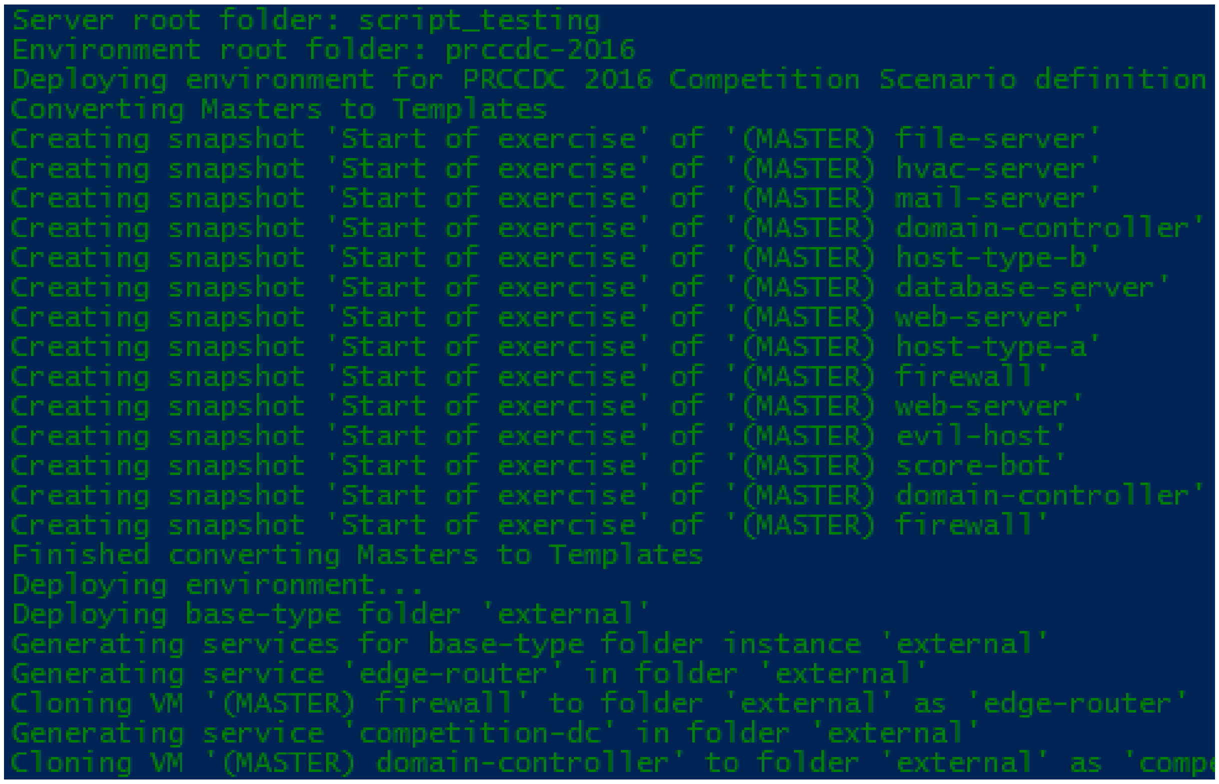
The tests for the Cyber Defense Competition scenario were run using PowerShell on the Windows environment. In Figure 7.6, the Mastering phase steps can be followed: syntax check, connect to platform, initialize groups (which fails due to no AD instance setup), creation of folder for Master instances, and creation of portgroups for networks. The cloning of the base instances from their defined templates then occurs for all services defined in folders, and snapshots are taken once cloning is complete.



```
DLES> adles -i vsphere.yaml -m -s .\examples\competition.yaml
 Successfully ingested specification file competition.yaml
 Checking syntax...
 Syntax check successful!
 Connecting to vSphere: scripty@vsphere.local@csds-vc-65.csds.uidaho.edu:443
 Connected to vSphere host csds-vc-65.csds.uidaho.edu:443
 Could not initialize AD-group Red Team
 Could not initialize AD-group Black Team
 Server root folder: script_testing
 Environment root folder: prccdc-2016
 Creating Master instances for PRCCDC 2016 Competition Scenario definition
 Created master folder 'MASTER-FOLDERS' under folder 'prccdc-2016'
 Creating unique-networks
 Creating generic-networks
 Generating Master base-type folder external
 Creating Master instance 'edge-router' from service 'firewall'
 Cloning VM 'VyOS 1.1.7 (64-bit)' to folder '(MASTER) external' as '(MASTER) firewall'
 Creating snapshot 'initial mastering snapshot' of '(MASTER) firewall'
 Creating Master instance 'competition-dc' from service 'domain-controller'
 Cloning VM 'Windows Server 2012 R2 (64-bit)' to folder '(MASTER) external' as '(MASTER)
 Creating snapshot 'initial mastering snapshot' of '(MASTER) domain-controller'
 Creating Master instance 'score-bot' from service 'score-bot'
 Cloning VM 'PRCCDC Score Bot' to folder '(MASTER) external' as '(MASTER) score-bot'
 Creating snapshot 'initial mastering snapshot' of '(MASTER) score-bot'
 Generating Master base-type folder blue-team
 Creating Master instance 'gateway' from service 'firewall'
 Cloning VM 'VyOS 1.1.7 (64-bit)' to folder '(MASTER) blue-team' as '(MASTER) firewall'
 Creating snapshot 'initial mastering snapshot' of '(MASTER) firewall'
 Creating Master instance 'windows-hosts' from service 'host-type-a'
 Cloning VM 'Windows 7 (64-bit)' to folder '(MASTER) blue-team' as '(MASTER) host-type-a'
```

Figure 7.6: Mastering commandline output

In Figure 7.7, the output from the beginning of the deployment phase is seen. Master instances have a snapshot taken and are converted to Templates. Then the deployment starts, with each service instance being cloned from its respective Master instance and reconfigured to use Generic version of networks if necessary. The network reconfiguration is not seen due to verbose output being disabled for brevity. It is similar to what is seen in the previous scenario's deployment phase output.



```
Server root folder: script_testing
Environment root folder: prccdc-2016
Deploying environment for PRCCDC 2016 Competition Scenario definition
Converting Masters to Templates
Creating snapshot 'Start of exercise' of '(MASTER) file-server'
Creating snapshot 'Start of exercise' of '(MASTER) hvac-server'
Creating snapshot 'Start of exercise' of '(MASTER) mail-server'
Creating snapshot 'Start of exercise' of '(MASTER) domain-controller'
Creating snapshot 'Start of exercise' of '(MASTER) host-type-b'
Creating snapshot 'Start of exercise' of '(MASTER) database-server'
Creating snapshot 'Start of exercise' of '(MASTER) web-server'
Creating snapshot 'Start of exercise' of '(MASTER) host-type-a'
Creating snapshot 'Start of exercise' of '(MASTER) firewall'
Creating snapshot 'Start of exercise' of '(MASTER) web-server'
Creating snapshot 'Start of exercise' of '(MASTER) evil-host'
Creating snapshot 'Start of exercise' of '(MASTER) score-bot'
Creating snapshot 'Start of exercise' of '(MASTER) domain-controller'
Creating snapshot 'Start of exercise' of '(MASTER) firewall'
Finished converting Masters to Templates
Deploying environment...
Deploying base-type folder 'external'
Generating services for base-type folder instance 'external'
Generating service 'edge-router' in folder 'external'
Cloning VM '(MASTER) firewall' to folder 'external' as 'edge-router'
Generating service 'competition-dc' in folder 'external'
Cloning VM '(MASTER) domain-controller' to folder 'external' as 'compe
```

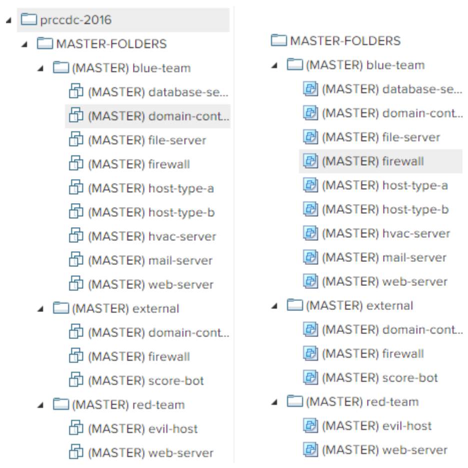
Figure 7.7: Deployment commandline output

The resulting environment

In figure 7.8, the image on the left is the instances created during the Mastering phase. On the right are the same instances converted into Template VMs during the deployment phase, meaning they cannot be modified, powered on, configured, or otherwise changed. This is the expected result from running the deployment phase, as it converts all the Master instances to templates before deploying the environment.

Figure 7.9 shows the resulting instances from deploying the competition example. The competition infrastructure with scorebots (far left), blue teams with their hosts and services (middle), and red team hosts (far right) can all be seen, and match up with what was defined in the specification for the exercise.

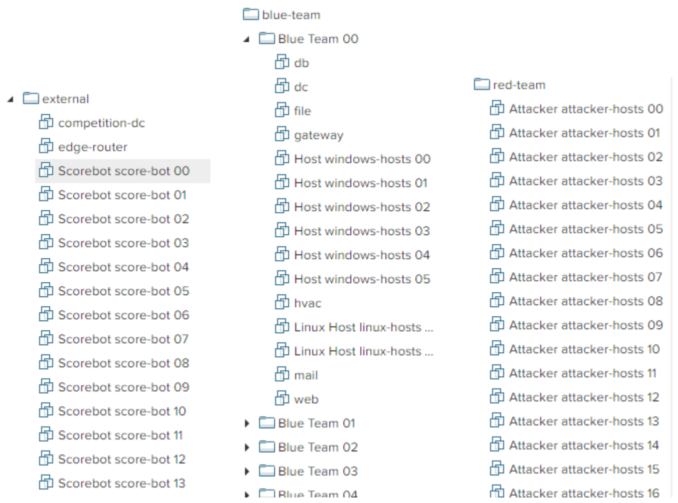
The comparison being done in Figure 7.10 is similar to that done in the previous scenario: Template on the left generated the instance on the right, and their networks differ. In this case, the instance generated uses unique-type networks, which is different from the generic networks used in the previous scenario.



```
prccdc-2016
  MASTER-FOLDERS
    (MASTER) blue-team
      (MASTER) database-se...
      (MASTER) domain-cont...
      (MASTER) file-server
      (MASTER) firewall
      (MASTER) host-type-a
      (MASTER) host-type-b
      (MASTER) hvac-server
      (MASTER) mail-server
      (MASTER) web-server
    (MASTER) external
      (MASTER) domain-cont...
      (MASTER) firewall
      (MASTER) score-bot
    (MASTER) red-team
      (MASTER) evil-host
      (MASTER) web-server
```

```
MASTER-FOLDERS
  (MASTER) blue-team
    (MASTER) database-se...
    (MASTER) domain-cont...
    (MASTER) file-server
    (MASTER) firewall
    (MASTER) host-type-a
    (MASTER) host-type-b
    (MASTER) hvac-server
    (MASTER) mail-server
    (MASTER) web-server
  (MASTER) external
    (MASTER) domain-cont...
    (MASTER) firewall
    (MASTER) score-bot
  (MASTER) red-team
    (MASTER) evil-host
    (MASTER) web-server
```

Figure 7.8: Pre-deployment and post-deployment Masters for the competition

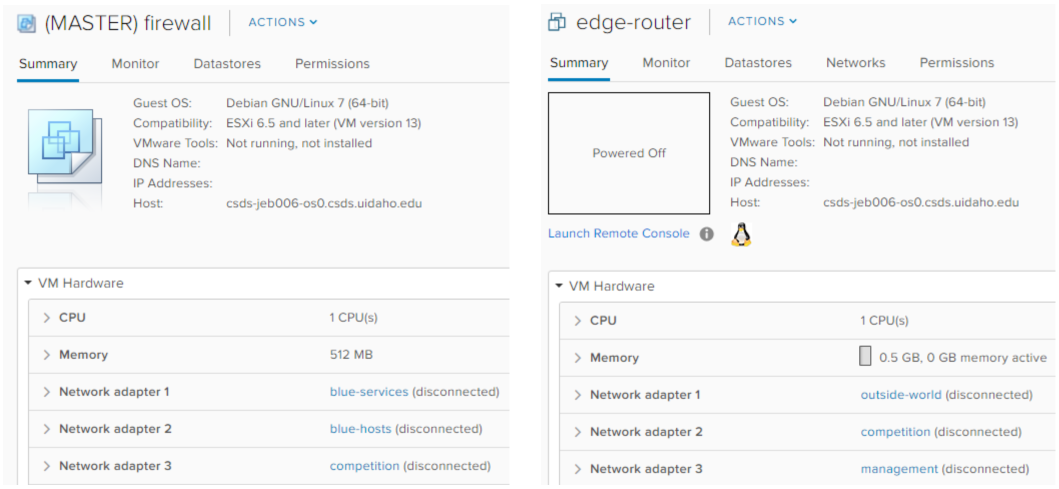


```
external
  competition-dc
  edge-router
  Scorebot score-bot 00
  Scorebot score-bot 01
  Scorebot score-bot 02
  Scorebot score-bot 03
  Scorebot score-bot 04
  Scorebot score-bot 05
  Scorebot score-bot 06
  Scorebot score-bot 07
  Scorebot score-bot 08
  Scorebot score-bot 09
  Scorebot score-bot 10
  Scorebot score-bot 11
  Scorebot score-bot 12
  Scorebot score-bot 13
```

```
blue-team
  Blue Team 00
    db
    dc
    file
    gateway
    Host windows-hosts 00
    Host windows-hosts 01
    Host windows-hosts 02
    Host windows-hosts 03
    Host windows-hosts 04
    Host windows-hosts 05
    hvac
    Linux Host linux-hosts ...
    Linux Host linux-hosts ...
    mail
    web
  Blue Team 01
  Blue Team 02
  Blue Team 03
  Blue Team 04
```

```
red-team
  Attacker attacker-hosts 00
  Attacker attacker-hosts 01
  Attacker attacker-hosts 02
  Attacker attacker-hosts 03
  Attacker attacker-hosts 04
  Attacker attacker-hosts 05
  Attacker attacker-hosts 06
  Attacker attacker-hosts 07
  Attacker attacker-hosts 08
  Attacker attacker-hosts 09
  Attacker attacker-hosts 10
  Attacker attacker-hosts 11
  Attacker attacker-hosts 12
  Attacker attacker-hosts 13
  Attacker attacker-hosts 14
  Attacker attacker-hosts 15
  Attacker attacker-hosts 16
```

Figure 7.9: Environment results from competition example



Guest OS: Debian GNU/Linux 7 (64-bit)
Compatibility: ESXi 6.5 and later (VM version 13)
VMware Tools: Not running, not installed
DNS Name:
IP Addresses:
Host: csds-jeb006-os0.csds.uidaho.edu

VM Hardware

CPU	1 CPU(s)
Memory	512 MB
Network adapter 1	blue-services (disconnected)
Network adapter 2	blue-hosts (disconnected)
Network adapter 3	competition (disconnected)

Guest OS: Debian GNU/Linux 7 (64-bit)
Compatibility: ESXi 6.5 and later (VM version 13)
VMware Tools: Not running, not installed
DNS Name:
IP Addresses:
Host: csds-jeb006-os0.csds.uidaho.edu

VM Hardware

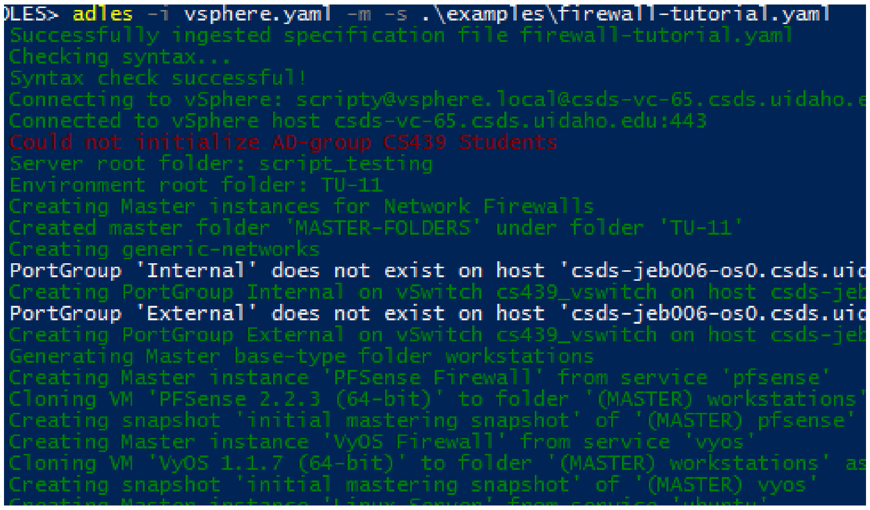
CPU	1 CPU(s)
Memory	0.5 GB, 0 GB memory active
Network adapter 1	outside-world (disconnected)
Network adapter 2	competition (disconnected)
Network adapter 3	management (disconnected)

Figure 7.10: Comparison of Blue Team gateway template with deployed edge router instance

7.4 Results for Other Tutorials

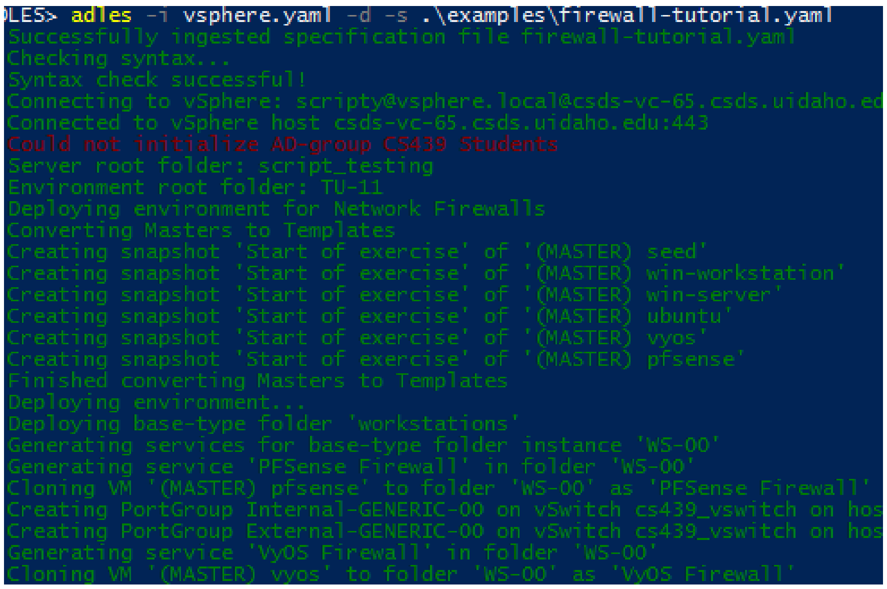
The commandline output from the two example tutorial scenarios is shown here to provide further illustration of what the execution of ADLES looks like.

Network Firewalls



```
LES> adles -i vsphere.yaml -m -s .\examples\firewall-tutorial.yaml
 Successfully ingested specification file firewall-tutorial.yaml
 Checking syntax...
 Syntax check successful!
 Connecting to vSphere: scripty@vsphere.local@csds-vc-65.csds.uidaho.e
 Connected to vSphere host csds-vc-65.csds.uidaho.edu:443
 Could not initialize AD-group CS439 Students
 Server root folder: script_testing
 Environment root folder: TU-11
 Creating Master instances for Network Firewalls
 Created master folder 'MASTER-FOLDERS' under folder 'TU-11'
 Creating generic-networks
 PortGroup 'Internal' does not exist on host 'csds-jeb006-os0.csds.uic
 Creating PortGroup Internal on vSwitch cs439_vswitch on host csds-jeb
 PortGroup 'External' does not exist on host 'csds-jeb006-os0.csds.uic
 Creating PortGroup External on vSwitch cs439_vswitch on host csds-jeb
 Generating Master base-type folder workstations
 Creating Master instance 'PFSense Firewall' from service 'pfsense'
 Cloning VM 'PFSense 2.2.3 (64-bit)' to folder '(MASTER) workstations'
 Creating snapshot 'initial mastering snapshot' of '(MASTER) pfsense'
 Creating Master instance 'VyOS Firewall' from service 'vyos'
 Cloning VM 'VyOS 1.1.7 (64-bit)' to folder '(MASTER) workstations' as
 Creating snapshot 'initial mastering snapshot' of '(MASTER) vyos'
```

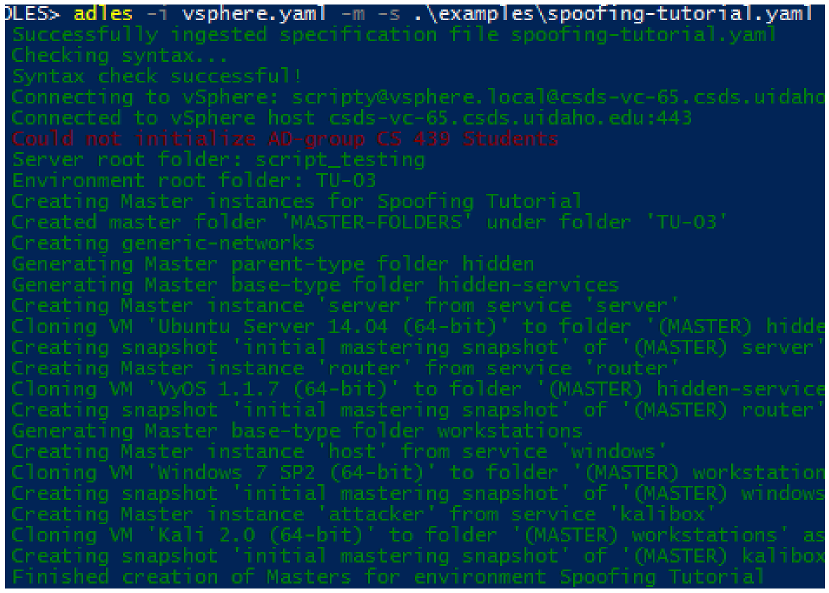
Figure 7.11: Commandline output for network firewalls mastering phase



```
LES> adles -i vsphere.yaml -d -s .\examples\firewall-tutorial.yaml
Successfully ingested specification file firewall-tutorial.yaml
Checking syntax...
Syntax check successful!
Connecting to vSphere: scripty@vsphere.local@csds-vc-65.csds.uidaho.ed
Connected to vSphere host csds-vc-65.csds.uidaho.edu:443
Could not initialize AD-group CS439 Students
Server root folder: script_testing
Environment root folder: TU-11
Deploying environment for Network Firewalls
Converting Masters to Templates
Creating snapshot 'Start of exercise' of '(MASTER) seed'
Creating snapshot 'Start of exercise' of '(MASTER) win-workstation'
Creating snapshot 'Start of exercise' of '(MASTER) win-server'
Creating snapshot 'Start of exercise' of '(MASTER) ubuntu'
Creating snapshot 'Start of exercise' of '(MASTER) vyos'
Creating snapshot 'Start of exercise' of '(MASTER) pfsense'
Finished converting Masters to Templates
Deploying environment...
Deploying base-type folder 'workstations'
Generating services for base-type folder instance 'WS-00'
Generating service 'PFSense Firewall' in folder 'WS-00'
Cloning VM '(MASTER) pfsense' to folder 'WS-00' as 'PFSense Firewall'
Creating PortGroup Internal-GENERIC-00 on vSwitch cs439_vswitch on hos
Creating PortGroup External-GENERIC-00 on vSwitch cs439_vswitch on hos
Generating service 'VyOS Firewall' in folder 'WS-00'
Cloning VM '(MASTER) vyos' to folder 'WS-00' as 'VyOS Firewall'
```

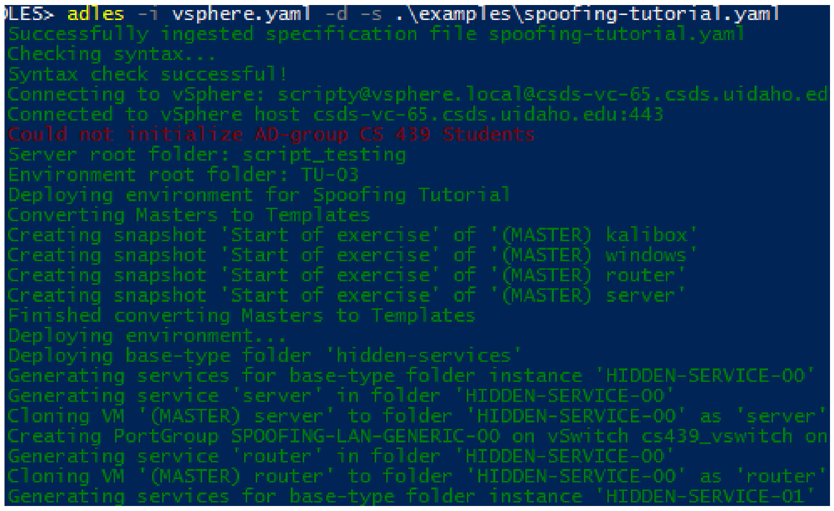
Figure 7.12: Commandline output for network firewalls deployment phase

Spoofing



```
DLES> adles -i vsphere.yaml -m -s .\examples\spoofing-tutorial.yaml
 Successfully ingested specification file spoofing-tutorial.yaml
 Checking syntax...
 Syntax check successful!
 Connecting to vSphere: scripty@vsphere.local@csds-vc-65.csds.uidaho
 Connected to vSphere host csds-vc-65.csds.uidaho.edu:443
 Could not initialize AD-group CS 439 Students
 Server root folder: script_testing
 Environment root folder: TU-03
 Creating Master instances for Spoofing Tutorial
 Created master folder 'MASTER-FOLDERS' under folder 'TU-03'
 Creating generic-networks
 Generating Master parent-type folder hidden
 Generating Master base-type folder hidden-services
 Creating Master instance 'server' from service 'server'
 Cloning VM 'Ubuntu Server 14.04 (64-bit)' to folder '(MASTER) hidde
 Creating snapshot 'initial mastering snapshot' of '(MASTER) server'
 Creating Master instance 'router' from service 'router'
 Cloning VM 'VyOS 1.1.7 (64-bit)' to folder '(MASTER) hidden-service
 Creating snapshot 'initial mastering snapshot' of '(MASTER) router'
 Generating Master base-type folder workstations
 Creating Master instance 'host' from service 'windows'
 Cloning VM 'Windows 7 SP2 (64-bit)' to folder '(MASTER) workstation
 Creating snapshot 'initial mastering snapshot' of '(MASTER) windows
 Creating Master instance 'attacker' from service 'kalibox'
 Cloning VM 'Kali 2.0 (64-bit)' to folder '(MASTER) workstations' as
 Creating snapshot 'initial mastering snapshot' of '(MASTER) kalibox
 Finished creation of Masters for environment Spoofing Tutorial
```

Figure 7.13: Commandline output for spoofing mastering phase



```
DLES> adles -i vsphere.yaml -d -s .\examples\spoofing-tutorial.yaml
Successfully ingested specification file spoofing-tutorial.yaml
Checking syntax...
Syntax check successful!
Connecting to vSphere: scripty@vsphere.local@csds-vc-65.csds.uidaho.ed
Connected to vSphere host csds-vc-65.csds.uidaho.edu:443
Could not initialize AD-group CS 439 Students
Server root folder: script_testing
Environment root folder: TU-03
Deploying environment for Spoofing Tutorial
Converting Masters to Templates
Creating snapshot 'Start of exercise' of '(MASTER) kalibox'
Creating snapshot 'Start of exercise' of '(MASTER) windows'
Creating snapshot 'Start of exercise' of '(MASTER) router'
Creating snapshot 'Start of exercise' of '(MASTER) server'
Finished converting Masters to Templates
Deploying environment...
Deploying base-type folder 'hidden-services'
Generating services for base-type folder instance 'HIDDEN-SERVICE-00'
Generating service 'server' in folder 'HIDDEN-SERVICE-00'
Cloning VM '(MASTER) server' to folder 'HIDDEN-SERVICE-00' as 'server'
Creating PortGroup SPOOFING-LAN-GENERIC-00 on vSwitch cs439_vswitch on
Generating service 'router' in folder 'HIDDEN-SERVICE-00'
Cloning VM '(MASTER) router' to folder 'HIDDEN-SERVICE-00' as 'router'
Generating services for base-type folder instance 'HIDDEN-SERVICE-01'
```

Figure 7.14: Commandline output for spoofing deployment phase

7.5 Discussion

Performance

Overall, while overall performance of the system is decent, there are a few obvious bottlenecks. The API calls made to vCenter, while fast on a local network, slowed runtime by 5-10 times when connecting over a remote VPN connection from off-campus. The impact of this is significantly lessened by the time spent cloning instances, but it is still noticeable, especially for small instances. The other bottleneck is the time it takes to clone instances. Each clone takes anywhere from a few seconds to several minutes depending on the size of the Template's disk, making the cloning of 250 VMs quite slow. One solution to this is to parallelize the cloning, enabling more efficient use of resources, especially in environments with clustered datastores. However, care must be taken during implementation, as it would be easy to start 15 clones of the same Template, which would severely contest the disk hosting the Template's files and significantly worsen performance.

Application in RADICL

The system has already seen some use in RADICL this semester with several of the scripts that utilize the vSphere wrapper. Next year, it will be utilized by students in CS439 for the creation of their tutorials. The course teacher will introduce students to the specification in the first week of class, along with the introduction to virtualization, the vSphere platform and how the course is run. At the end of the semester, the specifications they write for those tutorials will be bundled with the tutorial itself, licenced CC BY SA, and uploaded to a public GitHub repository, making them accessible to educators globally.

Chapter 8: Related Work

8.1 Terraform

Terraform is an open-source tool developed by Hashicorp to safely and predictably manage infrastructure[36]. At its core, it is "infrastructure as code". Declarative configuration files are used to describe the desired infrastructure. These files are used by Terraform to automatically build or change the platform(s) described to match the desired configuration. It supports a wide variety of cloud and virtualization platforms, including Docker, Amazon AWS, Microsoft Azure, and beta support for vSphere.

Terraform is designed for companies running distributed web applications and services. Common local platforms, such as vSphere, Microsoft Hyper-V, and Xen, are either not supported or have limited community support. This limits its usefulness for solving the problem of creating education environments, because it does not provide the flexibility to use local platforms or the adaptability to make use of the infrastructure and resources already in place at a school.

8.2 Provisioners

As was discussed in the background, some provisioners have the ability to manage and create infrastructure, similar to ADLES. One of these is Puppet, a system and software configuration management tool made by Puppet[52]. The basic version of Puppet is free and open-source, while the feature-rich Puppet Enterprise is paid and proprietary. Some platforms, such as Docker, KVM and vSphere, can be managed using Puppet. However, some of these are locked to the Enterprise version, while others are supported only by the community. Additionally, most configurations are not platform-agnostic, and complexity can vary by platform. While this works fine for large enterprises with experienced developers and administrators, it is not ideal for educators, let alone students. This fact gets to the heart of the issue: provisioning systems are designed for enterprises, not education. Their design focus is on the deployment of large environments with well-defined configurations that see little change once written. This contrasts sharply with educational use cases, such as duplicated student environments seen in the tutorial examples, or the constant changes in configurations that are often seen between (or even during) semesters.

8.3 EduRange

EduRange is a National Science Foundation (NSF)-funded Evergreen State College project that aims to create a cloud-hosted collection of cybersecurity exercises to supplement classroom lectures, labs, and other activities[7, 66]. The project is building an open-source cybersecurity education platform utilizing Amazon's AWS platform to create exercises. This platform, similar to ADLES, uses YAML specifications to define exercises that are then instantiated on AWS. Once instantiated, students can login to the environment and carry out the exercise.

Although EduRange is cloud-based and features a unique competitive "puzzle" environment to facilitate learning major cybersecurity concepts, it has some major drawbacks that limit its educational flexibility. First, the system is limited to the exercises provided by the EduRange team. There is currently no capability of defining specifications so that instructors can use to design and create their own exercises based on their unique teaching objectives. Second, networking configurations and choice of host OSs are limited, restricting the types of exercises that can be created. Finally, the scope of the design of the system and specification only includes tutorials, precluding the possibility of using the platform to conduct cybersecurity competitions.

8.4 Open Cyber Challenge Platform

The OCCP is a NSF-funded University of Rhode Island project to create a low-cost attack/defend cybersecurity education platform[64]. It was developed because their background research revealed that there was no low-cost and suitable option (or platform) on which colleges or high schools could conduct cyber challenges (e.g Cyber Defense, CTFs). The platform is designed around challenges, and consists of four "teams": a Red team that attacks the network, a Blue team that defends the network, a Gray team representing normal traffic, and a White team that scores and manages the platform. These are specified using a custom XML configuration file, similar to how ADLES uses YAML specifications. Once instantiated on a machine running VMware, the platform consists of a scoring server and a virtualized network, including switching, routing, and services. Students (the Blue team) connect to the environment using a virtual-private network from their own VMs or workstations. The Red and Gray teams are automated scripts, configured using the same file used to instantiate the infrastructure. This design closely aligns to what was originally envisioned for ADLES, and even influenced early development of the system.

There are some limitations to this platform. It is designed to be used by schools with limited resources, and thus is restricted to using host-based virtualization platforms such as VMware Workstation and VirtualBox. While it is somewhat generalized through its use of the OVF VM format, it cannot make effective use of lab infrastructure or cloud platforms, as the networking configuration and VM creation is designed around host-based hypervisors. This also limits its use for live CDC events, as they require infrastructure for hundreds of students, and dozens of human (non-scripted) Red team members. Finally, the capabilities of the implementation are limited, only supporting VMware Workstation and a handful of configurations. The platform does not appear to be under active development, so its capabilities are likely to stay in this limited state for some time.

8.5 NICE Challenge Project

The National Initiative for Cybersecurity Education (NICE) Challenge Project is designed to create a flexible set of challenge environments and supporting infrastructure with a low barrier of use[48]. It is sponsored by the National Institute of Standards and Technology (NIST) and the National Security Agency (NSA), and is managed by California State University, San Bernardino. It is a cloud-based platform that hosts a variety of pre-built cybersecurity educational tutorials spanning a variety of topics. Instructors can request a free account and set up access for their class. Their students can then log in and carry out the tutorial exercise. Once the students complete the exercise, the platform will automatically verify the actions the students performed, and assign a score the instructor can use to grade the student.

There are a number of issues that prevent this project from having the educational flexibility and support as ADLES. The project code is proprietary and closed-source, with no way to locally set up an environment to teach with, making schools restricted to the resources NICE has available. Custom scenarios are an impossibility, limiting the topics instructors can teach, and making competitions impossible. Instructors cannot run several classes in parallel, severely limiting the ability for a school to teach multiple classes.

8.6 DETERLab

Started by the University of Southern California in 2004, DETERLab is a cybersecurity research testbed that now spans dozens of institutions[47, 46]. It is a secure and generalized platform for running security research experiments. The capabilities of the testbed include multi-resolution virtualization of experiment resources, federation capabilities to connect disparate resources from partner institutions, multi-party experimentation, and the ability to safely connect risky experiments to the Internet. Finally, its orchestration of experiments provides deterministic control over various components in an experiment. It enables researchers to collaborate with colleagues spread around the country and conduct realistic large-scale cybersecurity experiments. Additionally, the platform is available to teachers to use for running tutorials for their classes.

There are some limitations that limit DETERLab's use for hands-on cybersecurity education. The design of the system and its features are geared toward research and experimentation. While it has the ability to be used for education, it's a secondary objective. Furthermore, only Windows XP and older

Ubuntu Linux hosts can be used for experiments, significantly limiting it's use to represent diverse or specialized environments.

8.7 Vagrant

Vagrant is an open-source tool by Hashicorp to create reproducible development environments[37]. It uses an easy-to-distribute specification file in Ruby-syntax to describe the configuration of a VM. This is then used to deterministically create and configure a VM that matches the described configuration. The creation is performed using a single command, "vagrant up", and the tear down process is performed using "vagrant destroy." It is very similar to ADLES in a number of respects, and was actually the original inspiration for the project.

However, there are limitations that precludes its use for educational environments. It is designed for single virtual machines, with multi-machine configurations being possible, but very difficult to set up.[1] Only desktop virtualization platforms are supported, restricting its use to individual workstations and preventing the use of any hardware infrastructure or resources available to an institution. Finally, networking between various independent instances on different workstations is difficult to set up properly, and depends heavily on the network configuration of the workstations and infrastructure being used.

8.8 SEED

SEED is an NSF-funded project started in 2002 to develop hands-on cybersecurity and IT educational exercises[25, 23, 24]. These exercises, known as SEED labs, consist of a pre-packaged VM, a file containing a step-by-step tutorial walking students through the exercise, and any additional materials needed to conduct the exercise, such as source code or binaries. They are freely available for anyone to download and use, and are compatible with the VirtualBox and VMware virtualization platforms, making them accessible to educators globally. There are currently over 30 labs available, with subjects ranging from System and Network security to Cryptography.

An instructor with access to a cloud platform or local infrastructure, such as those that use RADICL, would want to create instances of these machines for each student, instead of requiring students to create their own VMs. Using ADLES, they could easily create these instances by simply specifying the SEED machine template, the desired exercise materials, perhaps some scoring criteria, and what infrastructure to use. Additionally, the instructor could easily recycle the instances for another exercise by using ADLES to reset the environment to the start of an exercise.

[1]The author can personally attest to this, having spent an entire summer attempting to do such a configuration.

Chapter 9: Future Work

9.1 Project Vision

The "grand vision" of ADLES is to serve as the framework for a public Free and Open-Source Software (FOSS) repository of pre-built exercises, competitions, and classes. This repository would enable many schools that would otherwise not have the required talent or resources, such as high schools, community colleges, and elementary schools, to bring high-quality cybersecurity and IT education to their students and communities. This could serve to make great strides in resolving the shortage of skilled security workers, and increase the number of software developers and IT specialists knowledgeable about security.

A sizable area of future work, then, is designing and establishing this repository. Successful examples of this idea in practice are GitHub[31], the Docker Hub[21], Hashicorp's Atlas repository of Vagrant boxes[38], and FossHub[28].

Another area is the creation of more examples, including tutorials, competitions, and other creative uses of the platform, such as cybersecurity research experiments.

9.2 Specification Extensions

This section discusses potential extensions to the specifications, including monitoring, resource federation, and several other potential extensions.

Monitoring Extensions

Monitoring extensions would add data collection configurations to relevant areas of the specifications, enabling the implementation of high-fidelity data collection. This would greatly enhance the system's research applicability and enable other extensions, such as fully automated grading of results, visualization of exercises, collection of research data, data analytics, and generation of corpora. Some examples of these configurations are:

1. Secondary interfaces on services for aggregating their log data, such as Windows Event Logs, Unix Syslog, application logs, etc.

2. Network packet captures. These could be obtained by enabling promiscuous mode on a vSwitch, or enabling a SPAN monitoring port to aggregate the network traffic.

3. Configuration of a centralized logging server to collect data, such as Splunk or ELK, including specifying how the data aggregated should be "frozen" for inclusion with a package.

4. Configuration of Virtual Machine Introspection (VMI) on supported platforms for a high-fidelity view of exercises during execution.

5. Instrumentation of the platforms and aggregation of the resulting log data, including the logs created by ADLES itself.

Cyber-physical Resource and Lab Environment Federation

1. Further Resource extensions for cyber-physical testbeds, and integration of Resources into more aspects of the exercise and package specifications. Examples of resources include testbeds for: ICS, Wireless, USB devices, and car computers.

2. Addition of ability to federate connections between separate lab environments, enabling the sharing of testbed resources, virtualization infrastructure, and collaboration between educational institutions. This could be implemented by extending the current Resources section or the addition of a new section.

TracerFIRE is a good example of how these extensions could be utilized in the real-world[1]. An ideal lab design that makes use of these extensions is shown in figure 9.1.

Other extensions

1. Visualization of an exercise in progress, notably for competitive environments.

2. Define and implement automated scoring or grading of exercises. The "scoring criteria" defined for instances in folders is an initial stab at this.

3. Extension of the Groups section in the exercise specification with explicit specification of user roles and permissions.

4. Fully integrate and clearly define the role of exercise materials and other aspects of the Package specification.

5. Collaboration and communications for an exercise, e.g video conferencing, TeamSpeak, IRC channel, or a Discord server.

9.3 System Improvements

Improvements to current system

1. Improved documentation on how to make a package, how to setup a platform for system, etc.

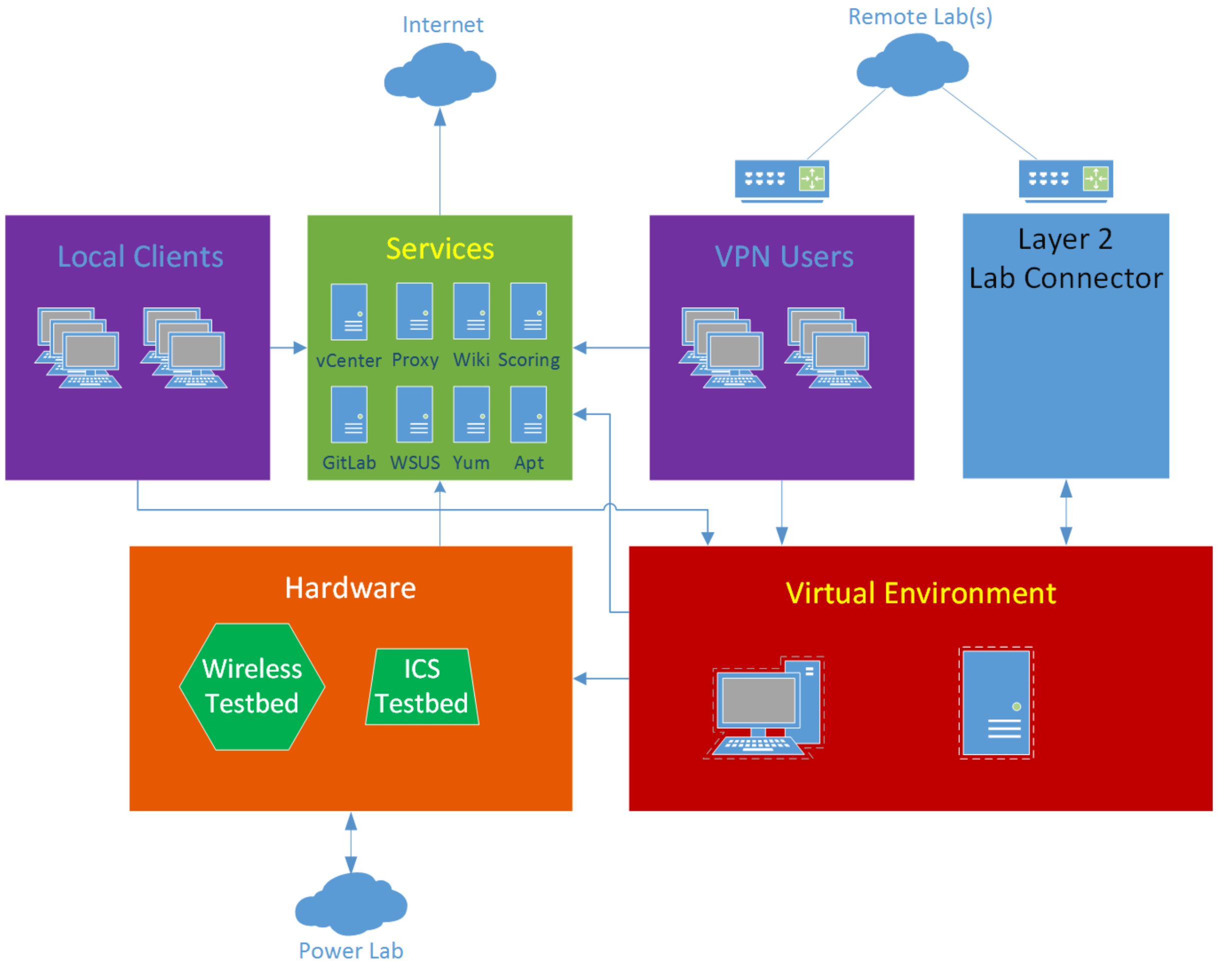


Figure 9.1: Network architecture of an ideal cybersecurity educational laboratory

2. Finish the implementation of groups and permissions for the vSphere platform interface.

3. Redo the syntax verification component. Currently, any changes in spec involve a non-trivial number of changes to the source code of the component. This is brittle and makes verifying new extensions difficult, as most implementers will not bother updating the component with the new syntax of their extensions.

4. Two-way transformation. Scan an existing environment and generate the specification for it.

5. Graphical user interface. This would improve the experience for users who are not familiar with commandline interfaces or for systems that restrict access to a commandline. Ideally, this would be web-based for portability and to enable easy hosting on a remote server. However, in the short term, this could be accomplished with a minimal amount of work using the EasyGUI Python module.

Support for additional platforms

Expanding ADLES to platforms other than vSphere is paramount to its success, and therefore an important area of future work. Adding capability to use a platform involves the following:

- Design and implement an Interface class for the platform, and put its hooks into PlatformInterface
- Design and implement a wrapper class, if necessary
- Add a new configuration to the services section of the exercise specification
- Add a new entry for configuring the platform in the infrastructure specification
- Add any relevant materials required for the service to the package specification

Examples of platforms that would be the most critical to implement for educational purposes are the following:

1. Docker: good for simulating large environments, with low resource overhead and quick load times[19].
   (a) Docker Machine
   (b) Docker Compose
   (c) Docker Swarm
2. Hyper-V server: free academic license, good for schools invested in the Microsoft ecosystem.
3. Vagrant: brings the platform to workstations and personal machines. Enables interaction with VirtualBox, desktop Hyper-V, and VMware Workstation.
4. Xen: free and open-source, scalable, robust. Rich introspection possibilities for monitoring extensions using VMI provided by the Xen API[5].
5. KVM: free and open-source, good for schools with a strong Linux background. LibVMI provides rich VMI possibilities here, as well.
6. Various cloud platforms, such as Microsoft Azure, Amazon AWS, Google Cloud Platform, or DigitalOcean. Clouds are dynamic, scalable, and cost only for the time utilized, making them perfect for short-lived tutorials or competitions.

Other Improvements

1. Visualization of what the network and service structure of a given exercise or package specification will look like without actually building the environment, including any cyber-physical testbeds, connected labs, and monitoring components if their corresponding extensions are implemented.
2. Ability to pause/freeze an in-progress exercise, ideally as a simple commandline argument.
3. Public repository of packages.
4. More examples:
   (a) Examples of other types of competitions, notably CTFs
   (b) Experiment examples
   (c) Greater variety of tutorials
5. Simplify system setup for educators beyond what a Python package provides
   (a) Vagrantfile that builds a lightweight VM running the system
   (b) Dockerfile that builds a lightweight Docker image running the system

Chapter 10: Summary and Conclusions

The field of cybersecurity is difficult to teach, due to the wide breadth of possible topics and the nuances that cannot possibly be captured in lectures. Hands-on exercises have been proven to be a very effective method of teaching cybersecurity that both engages learners and provides them with a skill-set immediately applicable in the real-world. These exercises usually run using virtualization technologies, such as VMware vSphere, and are configured and constructed manually by educators. However, creating these exercises often requires specialized IT knowledge and a significant amount of time and effort to build. This has cascaded to cause further issues with a lack of security, portability, and determinism.

In this thesis, a system that could potentially solve these problems was described: ADLES. The system uses formal declarative "specifications" describing an educational exercise to build an environment in a deterministic manner. These specifications are portable, making the sharing of exercises a possibility, and the system is platform-agnostic and extensible, enabling others to build the same environments utilizing their own platform infrastructure. Educators who previously had to manually build exercises can now use the automation provided by ADLES to construct these exercises with a minimal amount of effort.

The "grand vision" of this work is to enable high schools, community colleges, and elementary schools, regardless of technical skill and hardware capabilities, to provide high-quality cybersecurity and IT education to their students and communities. While today such education is a rarity, it is possible to envision a future where this education is as commonplace as traditional science and literary education. This would make great strides toward resolving the current shortage of skilled security workers, and increase the number of software developers and IT specialists knowledgeable about security.

References

[1] Benjamin Anderson, Kevin Nauer, Wellington Lee, J.T. McClain, and Rob Abbott. Tracer FIRE cyberforensic training platform. https://www.osti.gov/scitech/servlets/purl/1251138, 2015.

[2] Ansible. https://www.ansible.com/.

[3] Ansible documentation - YAML syntax. https://docs.ansible.com/ansible/playbooks.html.

[4] Apache license, Jan 2004.

[5] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5):164–177, October 2003.

[6] Oren Ben-Kiki, Clark Evans, and Ingy dot Net. YAML Ain't Markup Language (YAML) Version 1.1. http://yaml.org/spec/1.1/, Jan 2005.

[7] Stefan Boesen, Richard Weiss, James Sullivan, Michael E. Locasto, Jens Mache, and Erik Nilsen. EDURange: Meeting the pedagogical challenges of student participation in cybertraining environments. In *Proceedings of the 7th USENIX Conference on Cyber Security Experimentation and Test*, CSET'14, pages 9–9. USENIX Association, 2014.

[8] Michael Braun and Chris Ocker. Penetration testing, Mar 2017. CS 439: Applied Security Concepts.

[9] S. Caltagirone, P. Ortman, S. Melton, D. Manz, K. King, and P. Oman. Design and implementation of a multi-use attack-defend computer security lab. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, volume 9, pages 220c–220c, Jan 2006.

[10] Sergio Caltagirone, Paul Ortman, Sean Melton, David Manz, Kyle King, and Paul W Oman. RADICL: A Reconfigurable Attack-Defend Instructional Computing Laboratory. In *Security and Management*, pages 97–103, 2005.

[11] National Collegiate Cyber Defense Competition. http://www.nationalccdc.org.

[12] Peter Chapman, Jonathan Burket, and David Brumley. Picoctf: A game-based computer security competition for high school students. In *3GSE*, 2014.

[13] Chef. https://www.chef.io/chef/.

[14] Jessica A. Chisholm. Analysis on the perceived usefulness of hands-on virtual labs in cybersecurity classes, 2015.

[15] OverTheWire community. OverTheWire Wargames. `www.overthewire.org/wargames`.

[16] W. A. Conklin, R. E. Cline, and T. Roosa. Re-engineering cybersecurity education in the us: An analysis of the critical factors. In *2014 47th Hawaii International Conference on System Sciences*, pages 2006–2014, Jan 2014.

[17] J. D. Day and H. Zimmermann. The OSI reference model. *Proceedings of the IEEE*, 71(12):1334–1340, Dec 1983.

[18] DEFCON CTF archive. `https://www.defcon.org/html/links/dc-ctf.html`.

[19] Docker. `https://docs.docker.com/`.

[20] Docker Compose - overview. `https://docs.docker.com/compose/overview/`.

[21] Docker Hub. `https://hub.docker.com/`.

[22] Compose file version 3 reference. `https://docs.docker.com/compose/compose-file/`.

[23] Wenliang Du. Summary of the seed labs - for authors and publishers. `http://www.cis.syr.edu/~wedu/seed/Documentation/Textbook_Author_Publisher.pdf`.

[24] Wenliang Du. *SEED: A Suite of Instructional Laboratories for Computer SEcurity EDucation*. Syracuse University, Jan 2011.

[25] Wenliang Du. SEED: Hands-on lab exercises for computer security education. *IEEE Security & Privacy Magazine*, 9(5):70–73, 2011.

[26] Oren en Kiki, Clark Evans, and Ingy dot Net. YAML Ain't Markup Language (YAML) Version 1.2. `http://www.yaml.org/spec/1.2/spec.html`, Oct 2009.

[27] Verizon Enterprise. 2016 data breach investigations report. Technical report, Verizon Enterprise, Apr 2016.

[28] FossHub. `https://www.fosshub.com/about.html`.

[29] Alessio Gaspar, Sarah Langevin, William Armitage, R. Sekar, and T. Daniels. The role of virtualization in computing education. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '08, pages 131–132. ACM, 2008.

[30] Gabe Gibler and Colton Hotchkiss. Network firewalls, Feb 2017. CS 439: Applied Security Concepts.

[31] GitHub. `https://github.com/`.

[32] George E. Glasson. The effects of hands-on and teacher demonstration laboratory methods on science achievement in relation to reasoning ability and prior knowledge. *Journal of Research in Science Teaching*, 26(2):121–131, 1989.

[33] Christopher Goes. ADLES: GitHub. `https://github.com/GhostofGoes/ADLES`, 2017.

[34] Christopher Goes. ADLES: Python Package Index. `https://pypi.python.org/pypi/ADLES`, 2017.

[35] Forbes Guthrie, Scott Lowe, and Kendrick Coleman. *VMware vSphere design.* John Wiley & Sons, 2013.

[36] HashiCorp. Terraform. `https://www.terraform.io/intro/`.

[37] HashiCorp. Vagrant. `https://www.vagrantup.com/`.

[38] Atlas. `https://atlas.hashicorp.com/boxes/search`.

[39] David L Haury and Peter Rillero. *Perspectives of Hands-On Science Teaching.* ERIC, 1994.

[40] ISACA. State of cybersecurity implications for 2016. Technical report, ISACA, Feb 2016.

[41] Ananth Jillepalli, Nagarjuna Nuthalapati, Matt Kirkland, and Jonathan Buch. Domain controlling, Feb 2017. CS 439: Applied Security Concepts.

[42] Kyle King, David Manz, Paul Ortman, Doug Shikashio, and Paul Oman. A rapidly reconfigurable computer lab for software engineering security experiments and exercises. In *Proceedings of the 19th Conference on Software Engineering Education and Training Workshops*, CSEETW '06, pages 24–. IEEE Computer Society, Apr 2006.

[43] Wellington K. Lee, Tyler Morris, Andrew Chu, Katrina Gilmore, Joshua Russ, and Aliyah Carter. *ABQ ThunderBird Cup v3.0 Alpha Worksop: Workshop Analysis 2016.* Sandia National Labs, Nov 2016.

[44] L. F. Ludwig and D. F. Dunn. Laboratory for emulation and study of integrated and coordinated media communication. In *Proceedings of the ACM Workshop on Frontiers in Computer Communications Technology*, SIGCOMM '87, pages 283–291. ACM, 1988.

[45] Dale L. Lunsford. Virtualization technologies in information systems education. *Journal of Information Systems Education*, 20(3):339–348, Fall 2009.

[46] J. Mirkovic and T. Benzel. Teaching cybersecurity with deterlab. *IEEE Security Privacy*, 10(1):73–76, Jan 2012.

[47] Jelena Mirkovic, Terry V Benzel, Ted Faber, Robert Braden, John T Wroclawski, and Stephen Schwab. The DETER project: Advancing the science of cyber security experimentation and test. In *Technologies for Homeland Security (HST), 2010 IEEE International Conference on*. IEEE, 2010.

[48] NICE challenge project. `https://www.nice-challenge.com/`.

[49] Commission on Enhancing National Cybersecurity. Report on securing and growing the digital economy. Technical report, National Institute of Standards and Technology, Dec 2016.

[50] Pacific Rim Collegiate Cyber Defense Competition. `https://www.prccdc.org/`.

[51] Tom Preston-Werner. Semantic versioning 2.0.0. `http://semver.org/`.

[52] Puppet. `https://puppet.com/`.

[53] PyPI: Python Package Index. `https://pypi.python.org/pypi`.

[54] Tim Rains, Matt Miller, and David Weston. Exploitation trends: From potential risk to actual risk. In *RSA Conference*, Apr 2015.

[55] Eric Steven Raymond. *The Art of Unix Programming*. Pearson Education, Inc, Sep 2003.

[56] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address allocation for private internets. RFC 1918, IETF, Feb 1996.

[57] Ammar H. Safar and Fahad A. AlKhezzi. Beyond computer literacy: Technology integration and curriculum transformation. *College Student Journal*, 47(4):614–626, 12 2013.

[58] Sushil K Sharma and Joshua Sefchek. Teaching information systems security courses: A hands-on approach. *Computers & Security*, 26(4):290–299, Jun 2007.

[59] Kirill Simonav. PyYAML. `http://pyyaml.org/wiki/PyYAML`.

[60] Flexera Software. Vulnerability review 2016. Technical report, Flexera Software, Mar 2016.

[61] Michael Suby and Frank Dickson. The 2015 (ISC)2 global information security workforce study. Technical report, (ISC)2, Apr 2015.

[62] Guido van Rossum. Python programming language. `https://www.python.org/`.

[63] VMware. pyVmomi. `https://github.com/vmware/pyvmomi`, 2014.

[64] Richard H Wagner. Designing a network defense scenario using the open cyber challenge platform. `http://digitalcommons.uri.edu/theses/73`, 2013.

[65] Richard Weiss, Jens Mache, and Erik Nilsen. Top 10 hands-on cybersecurity exercises. *J. Comput. Sci. Coll.*, 29(1):140–147, October 2013.

[66] Richard S. Weiss, Stefan Boesen, James F. Sullivan, Michael E. Locasto, Jens Mache, and Erik Nilsen. Teaching cybersecurity analysis skills in the cloud. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 332–337. ACM, 2015.

[67] Tyler Wittreich. Spoofing tutorial, Mar 2015. CS 439: Applied Security Concepts.

Appendix A: Package Specification

```yaml
metadata:
  timestamp: "date"   # REQUIRED   Date package was created in UTC
   ↪ format: YYYY-MM-DD
  tag: "tag-name"     # REQUIRED   Unique identifier for this package
  name: "name of package"     # Suggested  Human-readable name for the
   ↪ package  [default: filename]
  description: "description"  # Suggested  Human-readable detailed
   ↪ description of package
  version: "0.0.0"            # Suggested  Semantic version of the
   ↪ package: major.minor.bugfix

contents:
  environment: "enviro-spec.yaml"   # REQUIRED    Name of the file
   ↪ containing the environment specification
  infrastructure: "infra-spec.yaml" # Suggested   Infrastructure
   ↪ configuration (This can be specified in environment spec as well)
  scoring: "scoring-criteria.yaml"  # Suggested   Scoring criteria (This
   ↪  can be specified in environment spec as well)
  results: "path/to/dir"     # Suggested   Relative path to directory
   ↪ containing: Network PCAPs, Collected logfiles, Quiz/test results,
   ↪ Scoring results, Student & Instructor feedback, and Chat Logs
  templates: "path/to/dir"   # Suggested   Relative path to directory
   ↪ containing: VM OVFs, Docker Images/Dockerfiles, provisioning
   ↪ scripts/packages, other payloads
  materials: "path/to/dir"   # Suggested   Relative path to directory
   ↪ containing: presentations, exercise scripts, instructions/guides/
   ↪ documentation
```

Listing A.1: Package Specification

Appendix B: Exercise Specification

```
# ** Document Metadata **
metadata:
  name: "name"                # Suggested   Human-readable title for the
   ↪  specification [default: filename]
  description: "description"  # Suggested   Human-readable detailed
   ↪ description of the specification
  activity: "the activity"    # Optional    The activity the
   ↪ specification is being used for, e.g "CS 439" or "PRCCDC"
  prefix: "GLOBAL-PREFIX"     # REQUIRED    Globally unique prefix that
   ↪ distinguishes this exercise's environments from others on the same
   ↪  infrastructure
  date-created: "date"        # Optional    UTC format: YYYY-MM-DD (
   ↪ Example: 2016-10-12)
  version: "0.0.0"            # Suggested   Semantic version of the
   ↪ document: major.minor.bugfix (Refer to: http://semver.org/)
  infra-file: "filename.yaml"          # REQUIRED    YAML file
   ↪ specifying the infrastructure used to create the exercise
   ↪ environment (See: infrastructure-specification.yaml for syntax)
  folder-name: "/Path/To/Folder-Name"  # Suggested   Path of the folder
   ↪  that will contain the exercise, relative to root defined in the
   ↪ infrastructure configuration


# ** Groups/Teams **
# User groups, such as teams, students or instructors
# These are used to apply permissions to the resources and machines in
   ↪ the generated virtual exercise environment
groups:
  Group Regular Example:
    # The three different methods of specifying users for a regular
   ↪ group
    ad-group: "Users"       # Option A  AD group must exist
    filename: "a-file.json" # Option B  File format: specifications/user
   ↪ -json-specification.json
    user-list: [ "user-a", "user-b" ] # Option C  List of usernames

  # For creating batch groups from a common template base, the "template
   ↪ " type can be used
  Group Template Example:
    instances: 10           # REQUIRED  Number of groups created from
   ↪ this template. This marks a group as a template.
    ad-group: "Group [X]"   # Option A  '[X]' is the instance number. AD
   ↪  group must exist
    filename: "users.json"  # Option B  File format: specifications/user
   ↪ -json-specification.json


# ** Services **
```

```yaml
# Definition of the services that will be created in the exercise environment, such as hosts, servers, and routers
# Case insensitive EXCEPT for names of sources, such as templates or images
# Networks are attached to interfaces in order of their definition, unless explicitly mapped
# Three types of services: template, container, compose
# template:     Templatized VM in vSphere
# container:    Docker container
# compose:      Docker Compose file
services:
  all-service-types:  # Configurations that can appear in any service definition
    note: "A note"           # Optional    Human-readable note visible by  end-user, such as default username/password
    network-interfaces: []  # Optional    List of network interfaces and  their optional configurations  [default: template or container specific]
    provisioner:  # Optional
      name: "name"  # REQUIRED  Name of provisioning tool, e.g Ansible, Chef, Puppet
      file: "file"  # REQUIRED  File to use for provisioner, e.g Playbook, Cookbook, Manifest
    resource-config:  # Optional  Resource allocation configurations for  the service
      cores: 0    # Optional  Number of CPU cores
      memory: 0   # Optional  Amount of RAM in MB
      storage: 0  # Optional  Amount of persistent storage in GB
  template-based-service:   # Option A
    template: "name"  # REQUIRED
    template-config:  # Optional    Configuration of Template settings using key-value pairs
      key: "value"
    guest-extensions: no    # Optional    Guest extensions will be installed or enabled (e.g VMware Tools)
  container-based-service:  # Option B
    dockerfile: "file"  # Option A    Dockerfile to build a image
    image: "name/tag"   # Option B    Name and Tag of a pre-built image
  compose-based-service:    # Option C
    compose-file: "filename.yml"  # REQUIRED


# ** Resources **
# Cyber-physical resources that are to be utilized. These will be attached to folders, networks, or groups as needed.
# Examples: A wireless testbed or SCADA testbed in RADICL, a transformer  in the power lab.
resources:
  resource-p:
    lab: "power-lab"           # REQUIRED    Name of the lab the resource  is associated with
    resource: "transformer"   # REQUIRED    Name of the specific
```

```
   ↪ resource


# ** Networks **
# Definitions of virtual networks (Layers 2 and 3 of the OSI model)
# Non-Private (RFC 1918) networks will result in a warning
# IP version (IPv4/IPv6) is implicitly defined by subnet address format
networks:
  unique-networks:    # Networks that are instantiated once and only
   ↪ once. Think of them as singletons.
    network-label:           # REQUIRED    Unique label used to identify
   ↪ the network (Replace "network-label" with the name of the network)
      description: "blah"    # Optional    Human-readable description of
   ↪ network
      subnet: "x.x.x.x/x"    # Suggested   IP network address and mask:
   ↪ SUBNET-IP/CIDR
      vlan: 0                # Optional    VLAN tag of the network. Must
   ↪ be a value < 2000. [default: globally unique value > 2000]
      vswitch: "name"        # Optional    Name of virtual switch used
   ↪ for the network [default: set in infrastructure-config or
   ↪ VsphereInterface]
  generic-networks:   # New networks are created per instance of a
   ↪ folder
    # This instance counter is global across folders. Each instance of a
   ↪  given folder will refer to the "global" value of the base at that
   ↪  index.
    # Example: a generic network for instance 5 of folder "hidden" will
   ↪ be the same network as instance 5 of folder "workstations"
    network-label:
      description: "description"
      subnet: "x.x.x.x/x"
      vswitch: "vswitch name"
      increment: no          # Optional    Increment the subnet value for
   ↪  each unique instance created   [default: no]


# ** Folders **
# Assemblages of objects in a hierarchical structure
# PHASES - Usually, there are two phases to creation of an exercise
   ↪ environment
#   Mastering     "Master" instances are created and configured by
   ↪ humans
#   Deployment    Full deployment of envrionment, using the Master
   ↪ instances created in the Mastering phase
# PERMISSIONS - Permissions are applied hierarchically. Groups with
   ↪ permissions to a given folder will have permissions for all of it'
   ↪ s children as well.
folders:
  parent-folder:  # Folders that contain other folders (replace "parent-
   ↪ folder" with name of the folder)
    group: group-label           # Optional    User group that will have
   ↪ permissions to the folder
```

```
    master-group: group-label   # Optional    User group for the pre-deployment masters [default: group specified for the folder]
    description: "description" # Optional    Human-readable description of folder
    enabled: yes                # Optional    Flag to selectivly disable a folder, so changes can be easily tested [default: yes]
    instances: 10               # Optional    Same configurations as for base-type folders
    base-folder:  # Folders that contain services (replace "base-folder" with name of the folder)
      group: group-label          # REQUIRED    User group that will have permissions to the folder
      master-group: group-label   # Optional    User group for the pre-deployment masters [default: group specified for the folder]
      description: "description" # Optional    Human-readable description of folder
      enabled: yes                # Optional    Flag to selectivly disable a folder, so changes can be easily tested [default: yes]
      instances:  # Optional    Makes folder a template that is copied N-times (NOTE: instances can also simply be an integer representing N)
        number: 10            # OPTION A    Number of instances = integer
        size-of: group-label  # OPTION B    Number of instances = Size of named group
        prefix: "prefix"      # Optional    String to prepend to named instance numbers [default: name of folder]
      services:   # REQUIRED    Define services that the base folder will contain
        service-instance-name:
          service: service-label  # REQUIRED    Label as defined in services
          instances: 10           # Optional    Same configurations as for base-type folders
          networks: ["subnet-a", "subnet-b"]    # Optional    Networks to attach the service instance to (Case sensitive!)
          provisioner-file: "file"  # Optional  Override provisioner configuration file for a service
          scoring:  # Optional    Scoring for the service (e.g a competition, or verification of homework)
            ports: [0]            # Suggested   Ports used for scoring
            protocols: ["proto"]  # Suggested   Protocols used for scoring
            criteria: "file.yaml" # REQUIRED    Criteria used to score the service
```

Listing B.1: Exercise Specification

Appendix C: Infrastructure Specification

```
# VMware vSphere
vmware-vsphere:
  hostname: "hostname"            # REQUIRED    Hostname of the vCenter
   ↪ server
  port: 0                         # Suggested   Port used to connect to
   ↪ the vCenter server [default: 443]
  login-file: "vsphere.json"      # Suggested   Login information used
   ↪ to connect to the vCenter server [default: prompt user]
  datacenter: "datacenter name"   # Suggested   Name of the Datacenter
   ↪ on which to create environment
  datastore: "datastore name"     # Suggested   Name of Datastore to use
   ↪  for environment VMs
  template-folder: "folder path"  # REQUIRED    Path from server root to
   ↪  folder that contains VM templates
  server-root: "folder name"      # Suggested   Name of folder
   ↪ considored to be "root" for the platform
  vswitch: "vswitch name"         # Suggested   Name of vSwitch to use
   ↪ as default
  host-list: ["a", "b"]           # Optional    List of names of ESXi
   ↪ hosts to use [default: first host found in the datacenter]

# Docker
docker:
  url: "host:port"        # Suggested   URL to the Docker server [
   ↪ default: unix:///var/run/docker.sock]
  tls: true               # Optional    Use TLS to connect to the Docker
   ↪  server [default: True]
  registry: # Optional
    url: "url://"         # REQUIRED  URL of the registry
    login-file: "r.json"  # REQUIRED  JSON file containing login
   ↪ information for the Docker registry server

# Amazon Web Services
amazon-aws:
  cred-file: "path"     # REQUIRED  Path to file with AWS access
   ↪ credentials
  config-file: "path"   # Suggested   Path to file with service
   ↪ configurations

# Digital Ocean
digital-ocean:
  token-file: "path"    # REQUIRED  Path to file containing access token

# Microsoft Hyper-V Server
hyper-v:
  hostname: "hostname"    # REQUIRED    Hostname of the Hyper-V server
  port: 0                 # Suggested   Port used to connect to Hyper-V
   ↪ server
  login-file: "path"       # REQUIRED    Path to file containing login
   ↪ information for the Hyper-V server
  version: "v2"            # Suggested   Version of the Hyper-V API to
   ↪ use
  vswitch: "name"          # Suggested   Name of VirtualEthernetSwitch to
   ↪  use as default
```

Listing C.1: Infrastructure Specification

Appendix D: Penetration Testing Tutorial Specification

```yaml
metadata:
  name: "Penetration Testing"
  description: "Tutorial on properly conducting a penetration test"
  activity: "CS 439 - Applied Security Concepts"
  prefix: "CS439-TR03-TU16"
  date-created: "2017-03-29"
  version: "0.3.4"
  infra-file: "infra.yaml"
  folder-name: "CS-439/TR-03/TU-16"  # CS 439, Tutorial Round 3,
    ↪ Tutorial 16

groups:
  Students:
    ad-group: "CS439 Students"
  Instructors:
    user-list: ["brau", "ocke"]

# Note with default credentials needed for exercise setup is preserved
    ↪ from template
services:
  kali:   # What the students will be using
    template: "Kali Linux 2016.2 (64-bit)"
  ics:    # Industrial Control System
    template: "Windows XP SP0"
  web:    # Apache webserver
    template: "Ubuntu Server 14.04 (64-bit)"
  laptop: # Walter's Laptop
    template: "Windows XP SP2"
  phone:  # Planted phone
    template: "Android 4.4"
  workstation: # Walter's workstation
    template: "Windows 7 SP2 (64-bit)"
  dc:     # Domain Controller
    template: "Windows Server 2012 (64-bit)"
  router: # Router
    template: "VyOS 1.1.7 (64-bit)"

networks:
  generic-networks:
    Attacker-net:
      description: "Used by the attacker"
      subnet: "192.168.0.0/24"
      increment: yes
    Mobile-ISP:
      description: "Attacker <-> Phone"
      subnet: "172.16.0.0/19"  # Unusual subnet mask is intended to make
    ↪  subnets more "random"
      increment: yes
    Workstations:
      description: "This is the internal corporate network"
      subnet: "192.168.1.0/24"
    WiFi:
      description: "Wireless access for company employees"
      subnet: "192.168.5.0/24"
    Web-Services:
      description: "Internet services for the company"
      subnet: "192.168.4.0/24"
    ICS:
      description: "SCADA communication and management network"
      subnet: "192.168.3.0/24"
    IT:
      description: "Where the system administrator lives"
      subnet: "192.168.2.0/24"

folders:
  student-workstations:
    description: "Workstations visible to and used by students during
    ↪ the tutorial"
    group: Students
    master-group: Instructors
    instances:
      number: 15
      prefix: "WS-"
    services:
      attacker:
        service: kali
        networks: ["Attacker-net", "Mobile-ISP"]
  exercise-environemnt:
    description: "Systems the students will explore during the exercise,
    ↪  but should not see"
    group: Instructors
    systems:
      group: Instructors
      instances:
        number: 15
        prefix: "HIDDEN-ENVIRONMENT-"
      services:
        exercise-router:
          service: router
          networks: ["Attacker-net", "Workstations", "IT", "ICS", "Web-
    ↪ Services", "WiFi"]
        sysad:
          service: kali
          networks: ["IT"]
        ics-system:
          service: ics
          networks: ["ICS"]
        web-server:
          service: web
          networks: ["Web-Services"]
        walter-workstation:
          service: workstation
          networks: ["Workstations"]
        domain-controller:
          service: dc
          networks: ["Workstations"]
        planted-phone:
          service: phone
          networks: ["WiFi", "Mobile-ISP"]
        walter-laptop:
          service: laptop
          networks: ["WiFi"]
```

Listing D.1: Penetration Testing Tutorial Specification

Appendix E: Competition Specification

```yaml
metadata:
  name: "PRCCDC 2016 Competition Scenario definition"
  description: "The competition scenario environment"
  activity: "Pacific Rim Collegiate Cyber Defense Competition"
  prefix: "PRCCDC"
  date-created: "2016-11-01"
  version: "2.11.2"
  infra-file: "infra.yaml"
  folder-name: "Competitions/PRCCDC/prccdc-2016"

groups:
  Blue Team:  # Defenders
    instances: 14
    filename: "examples/blue-teams.json"
  Red Team:   # Attackers
    ad-group: "Red Team"
  Black Team: # Infrastructure team
    ad-group: "Black Team"

# Blue and Red teams know from their team packets what the default
   ↪ logins are for all services
services:
  score-bot:
    template: "PRCCDC Score Bot"
  evil-host:
    template: "Kali 2.0 (64-bit)"
  host-type-a:
    template: "Windows 7 (64-bit)"
  host-type-b:
    template: "Windows 10 (64-bit)"
  host-type-c:
    template: "Ubuntu Desktop 14.04 (64-bit)"
  firewall:
    template: "VyOS 1.1.7 (64-bit)"
    note: "Default Username: vyos   Password: vyos"
  domain-controller:
    template: "Windows Server 2012 R2 (64-bit)"
  web-server:
    template: "Ubuntu Server 14.04 (64-bit)"
  database-server:
    template: "Fedora 22 (64-bit)"
  file-server:
    template: "Windows Server 2008 R2 (64-bit)"
  mail-server:
    template: "Windows Server 2003 (32-bit)"
  hvac-server:
    template: "Solaris 9 (32-bit)"

```

```
networks:
  unique-networks:
    competition:
      subnet: "172.30.0.0/16"
      vswitch: "competition_vswitch"
    outside-world:
      subnet: "172.20.0.0/16"
      vswitch: "competition_vswitch"
    management:
      subnet: "10.10.10.0/24"
      vswitch: "competition_vswitch"
  generic-networks:
    blue-hosts:
      subnet: "10.0.0.0/24"
      vswitch: "competition_vswitch"
    blue-services:
      subnet: "192.168.0.0/24"
      vswitch: "competition_vswitch"
      increment: yes

folders:
  external:
    group: "Black Team"
    description: "The open competition network with red teams and
   ↪ scorebots"
    services:
      edge-router:
        description: "Connection to Internet"
        service: firewall
        networks: ["outside-world", "competition", "management"]
      competition-dc:
        description: "Provides DNS and DHCP"
        service: domain-controller
        networks: ["competition", "management"]
      # Scorebots communicate over management network to randomly select
   ↪  which team gets pinged from which scorebot
      score-bot:
        description: "Used to score teams"
        service: score-bot
        networks: ["competition", "management"]
        instances:
          number: 14
          prefix: "Scorebot "
  blue-team:
    instances:
      number: 14
      prefix: "Blue Team "
    group: "Blue Team"
    services:
      gateway:
        description: "Gateway connecting Blue teams to main competition
   ↪ network"
```

```
      service: firewall
      networks: ["competition", "blue-hosts", "blue-services"]
    windows-hosts:
      description: "Windows hosts that the blue team members use to
 ↪  access their services"
      instances:
        prefix: "Host "
        number: 6
      service: host-type-a
      networks: ["blue-hosts"]
    linux-hosts:
      description: "Linux hosts that the Blue team members use to
 ↪  access their services"
      instances:
        prefix: "Linux Host "
        number: 2
      service: host-type-b
      networks: ["blue-hosts"]
    dc:
      description: "Domain Controller for Blue Team network. Runs DNS,
 ↪   DHCP, and Active Directory."
      service: domain-controller
      networks: ["blue-services"]
    web:
      description: "Web server hosting the team's page"
      service: web-server
      networks: ["blue-services"]
      scoring:
        ports: [80, 443]
        protocols: ["http", "https"]
        criteria: "criteria-file.yaml"
    db:
      description: "Database"
      service: database-server
      networks: ["blue-services"]
    mail:
      description: "Mail Server"
      service: mail-server
      networks: ["blue-services"]
      scoring:
        ports: [25]
        protocols: ["smtp"]
        criteria: "criteria-file.yaml"
    hvac:
      description: "HVAC Server"
      service: hvac-server
      networks: ["blue-services"]
      scoring:
        ports: [22]
        protocols: ["ssh"]
        criteria: "criteria-file.yaml"
    file:
      description: "File Server"
      service: file-server
      networks: ["blue-services"]
      scoring:
        ports: [21, 445]
        protocols: ["sftp", "smb"]
        criteria: "criteria-file.yaml"
red-team:
  group: "Red Team"
  services:
    attacker-hosts:
      instances:
        number: 40
        prefix: "Attacker "
      description: "Host used by Red Team to attack Blue teams"
      service: evil-host
      networks: ["competition"]
    collaboration-server:
      description: "Server for Red Team collaboration, including Armitage, SFTP, and IRC"
      service: web-server
      networks: ["competition"]
```

Listing E.1: Competition Specification

Appendix F: Network Firewalls Tutorial Specification

```yaml
metadata:
  name: "Network Firewalls"
  description: "A tutorial on the usage and configuration of network
   ↪ firewalls"
  activity: "CS 439 - Applied Security Concepts"
  prefix: "CS439-TR03-TU16"
  version: "0.3.0"
  infra-file: "infra.yaml"
  folder-name: "CS-439/TR-02/TU-11"  # CS 439, Tutorial Round 2,
   ↪ Tutorial 11

groups:
  Students:
    ad-group: "CS439 Students"
  Instructors:
    user-list: ["gibl", "hotc"]

# Default passwords are preserved from templates
services:
  pfsense:
    template: "PFSense 2.2.3 (64-bit)"
  ubuntu:
    template: "Ubuntu 16.04 (64-bit)"
  vyos:
    template: "VyOS 1.1.7 (64-bit)"
  win-workstation:
    template: "Windows 10 (64-bit)"
  win-server:
    template: "Windows Server 2012 R2 (64-bit)"
  seed:
    template: "Ubuntu 12.04 SEED (32-bit)"

networks:
  generic-networks:
    Internal:
      description: "Internal 'Local' network that is being protected"
      subnet: "192.168.0.0/24"
      vswitch: "cs439_vswitch"
    External:
      description: "External untrusted network that is treated as the '
   ↪ Internet' for this exercise"
      subnet: "172.16.0.0/16"
      vswitch: "cs439_vswitch"

folders:
  workstations:
    group: Students
    master-group: Instructors
    instances:
      number: 15
      prefix: "WS-"
    services:
      PFSense Firewall:
        service: pfsense
        networks: ["Internal", "External"]
      VyOS Firewall:
        service: vyos
        networks: ["Internal", "External"]
      Linux Server:
        service: ubuntu
        networks: ["Internal"]
      Windows Server:
        service: win-server
        networks: ["Internal"]
      Windows Workstation:
        service: win-workstation
        networks: ["Internal"]
      SEED Workstation:
        service: seed
        networks: ["Internal"]
```

Listing F.1: Network Firewalls Tutorial Specification

Appendix G: Spoofing Tutorial

```yaml
metadata:
  name: "Spoofing Tutorial"
  description: "Learning about ARP spoofing and DNS spoofing, and
    ↪ implementing DNSSEC and DNSCrypt"
  activity: "CS 439 - Applied Security Concepts"
  prefix: "CS439"
  date-created: "2016-11-06"
  version: "0.6.1"
  infra-file: "infra.yaml"
  folder-name: "CS-439/TR-01/TU-03"  # CS 439, Tutorial Round 1,Tutorial
    ↪  03

groups:
  Students:
    ad-group: "CS 439 Students"
  Instructors:
    filename: "examples/tutorial_instructors.json"

services:
  windows:
    note: "Username: User   Password: Windows1"
    template: "Windows 7 SP2 (64-bit)"
  kalibox:
    note: "Username: root   Password: toor"
    template: "Kali 2.0 (64-bit)"
  server:
    template: "Ubuntu Server 14.04 (64-bit)"
  router:
    template: "VyOS 1.1.7 (64-bit)"

networks:
  generic-networks:
    SPOOFING-LAN:
      description: "Used for spoofing tutorial"
      subnet: "192.168.1.0/24"
      vswitch: "cs439_vswitch"

folders:
  hidden:
    description: "Services for each student that they do not see"
    group: Instructors
    hidden-services:
      group: Instructors
      instances:
        number: 15 # size-of: Students
        prefix: "HIDDEN-SERVICE-"
      services:
        server:
          service: server
          networks: ["SPOOFING-LAN"]
        router:
          service: router
          networks: ["SPOOFING-LAN"]
  workstations:
    description: "Workstations students use for the tutorial"
    group: Students
    master-group: Instructors
    instances:
      number: 15 # size-of: Students
      prefix: "WS-"
    services:
      host:
        service: windows
        networks: ["SPOOFING-LAN"]
      attacker:
        service: kalibox
        networks: ["SPOOFING-LAN"]
```

Listing G.1: Spoofing Tutorial Specification

Appendix H: Experiment Example Specification

```yaml
metadata:
  name: "Network Security Visualization Test Data Collection"
  description: "Testbed for collection of realistic network data for Network Security Visualization research"
  activity: "Experiments-2017"
  prefix: "EXPERIMENT"
  date-created: "2017-02-10"
  version: "0.3.0"
  infra-file: "infra.yaml"
  folder-name: "NSV-Research/nsv-experiment"

groups:
  Researchers:
    ad-group: "NSV Research Team"

services:
  vyos:
    template: "VyOS 1.1.7 (64-bit)"
    note: "Username: vyos   Password: vyos"
  pfsense:
    template: "PFSense 2.2.3 (64-bit)"
    note: "Web Interface is used to configure this. Username: admin Password: pfsense"
  dc:
    template: "Windows Server 2012 R2 (64-bit)"
    note: "Domain Controller"
  dns:
    template: "RHEL 6 (32-bit)"
    note: "DNS server"
  ntp:
    template: "FreeBSD 10 (32-bit)"
    note: "NTP server"
  server2012:
    template: "Windows Server 2012 R2 (64-bit)"
  apache:
    template: "Ubuntu Server 16.10 LAMP (64-bit)"
    note: "Linux Apache MySQL PHP (LAMP) server"
  nginx:
    template: "NGINX"
    note: "NGINX load balancer for servers"
  windows:
    template: "Windows 7 SP2 (64-bit)"
    note: "Username: Tester   Password: Windows1"
  kali:
    template: "Kali 2016.2 (64-bit)"
    note: "Username: root     Password: toor"

networks:
  unique-networks:
    SERVER-NET:
      description: "Network for testbed servers hosting data accessed by clients"
      subnet: "10.0.0.0/24"
    CLIENT-NET:
      description: "Network for testbed clients accessing the server data"
      subnet: "192.168.0.0/24"
    WAN-NET:
      description: "In-between network that acts as the internet for this simulation"
      subnet: "172.16.0.0/16"
    SERVER-WAN:
      description: "Network that connects the SERVER-NET router with the WAN-NET router"
      subnet: "192.168.100.0/30"
    CLIENT-WAN:
      description: "Network that connects the CLIENT-NET router with the WAN-NET router"
      subnet: "192.168.200.0/30"

folders:
  clients:
    description: "Clients for the testbed"
    group: Researchers
    services:
      windows-client:
        service: windows
        instances: 40
        networks: ["CLIENT-NET"]
      linux-client:
        service: kali
        instances: 5
        networks: ["CLIENT-NET"]
      domain-controller:
        service: dc
        networks: ["CLIENT-NET"]
  servers:
    description: "Servers for the testbed"
    group: Researchers
    services:
      windows-web:
        service: server2012
        instances: 15
        networks: ["SERVER-NET"]
      apache-web:
        service: apache
        instances: 20
        networks: ["SERVER-NET"]
      nginix-server:
        service: nginx
        networks: ["SERVER-NET"]
      domain-controller:
        service: dc
        networks: ["SERVER-NET"]
    wan:
      description: "Simulation of services on the Internet, such as DNS
    ↪ and NTP"
      group: Researchers
      services:
        dns-server:
          service: dns
          instances: 3
          networks: ["WAN-NET"]
        ntp-server:
          service: ntp
          instances: 3
          networks: ["WAN-NET"]
    routers:  # Ideally, these would use RIPv2 to share routing
    ↪ information
      description: "Routers connecting clients and servers"
      group: Researchers
      services:
        client-router:
          service: pfsense
          networks: ["CLIENT-NET", "CLIENT-WAN"]
        server-router:
          service: vyos
          networks: ["SERVER-NET", "SERVER-WAN"]
        wan-router:
          service: vyos
          networks: ["WAN-NET", "CLIENT-WAN", "SERVER-WAN"]
```

Listing H.1: Experiment Example

Appendix I: vsphere-info Script Source Code

```python
"""Query information about a vSphere environment and objects within it.

Usage:
    vsphere-info [options]

Options:
    -h, --help          Prints this page
    --version           Prints current version
    -n, -no-color       Do not color terminal output
    -v, --verbose       Emit debugging logs to terminal
    -f, --file FILE     Name of JSON file with server connection information

Examples:
    vsphere-info -vf logins.json

"""

import logging

from docopt import docopt

from adles.vsphere import vm_utils, vsphere_utils
from adles.utils import script_setup, resolve_path, prompt_y_n_question
from adles.vsphere.folder_utils import enumerate_folder, format_structure

__version__ = "0.6.1"


def main():
    args = docopt(__doc__, version=__version__, help=True)
    server = script_setup('vsphere_info.log', args, (__file__, __version__))

    thing_type = str(input("What type of thing do you want to get information on?"
                           " (vm | datastore | vsphere | folder) "))

    # Single Virtual Machine
    if thing_type == "vm":
        vm, vm_name = resolve_path(server, "vm", "you want to get information on")
        logging.info(vm_utils.get_vm_info(vm, detailed=True, uuids=True, snapshot=True, vnics=True))

    # Datastore
    elif thing_type == "datastore":
        ds = server.get_datastore(str(input("Enter name of the Datastore [leave blank "
                                            "for first datastore found]: ")))
        logging.info(vsphere_utils.get_datastore_info(ds))

    # vCenter server
    elif thing_type == "vsphere":
        logging.info("%s", str(server))

    # Folder
    elif thing_type == "folder":
        folder, folder_name = resolve_path(server, "folder")
        if "VirtualMachine" in folder.childType \
                and prompt_y_n_question("Want to see power state of VMs in the folder?"):
            contents = enumerate_folder(folder, recursive=True, power_status=True)
        else:
            contents = enumerate_folder(folder, recursive=True, power_status=False)
        logging.info("Information for Folder %s\nTypes of items folder can contain: %s\n%s",
                     folder_name, str(folder.childType), format_structure(contents))

    # That's not a thing!
    else:
        logging.info("Invalid thing: %s", thing_type)


if __name__ == '__main__':
    main()
```

Listing I.1: vsphere-info script