2014

EXTENSIBLE CYBER CHALLENGE PLATFORM CONTROL APPLICATION

Jacob A. Fonseca
University of Rhode Island, jafonseca@cs.uri.edu

EXTENSIBLE CYBER CHALLENGE PLATFORM CONTROL APPLICATION

BY

JACOB A FONSECA

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER SCIENCE AND STATISTICS

UNIVERSITY OF RHODE ISLAND

2014

MASTER OF SCIENCE THESIS

OF

JACOB A FONSECA

APPROVED:

Thesis Committee:

Major Professor Victor Fay-wolfe

Lisa DiPippo

Stu Westin

Nasser H. Zawia
DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2014

ABSTRACT

Cyber challenges are an exciting way to teach and assess computer security skills and methods. The commonalities in cyber challenges revolve around network traffic simulation, score keeping, and event automation. The Open Cyber Challenge Platform centralizes these tasks into a single piece of software called a game server. Existing challenge platforms use proprietary mechanisms to accomplish these goals motivating a need to find a solution application. This design and implementation of a game server application will operate as the run time software of the Open Cyber Challenge Platform. The testing of our game server's components show that the implementation described here can successfully fill the needs of the Open Cyber Challenge Platform's requirements for a game server application.

ACKNOWLEDGMENTS

The work completed in this thesis was only made possible with the support of a fantastic group of outstanding and inspiring people. The graduate students, staff and faculty in the Department of Computer Science and Statistics are essential members of the team whose effort was required to complete this project. I would like to thank Travis, and Dan for sticking out the long haul with me while others fled for the hills. The technical skills and computer wizardry of Dr. Kevin Bryan provided me with inspiration and desire to do things the right way. Dr. DiPippo, Jessie, Brittnee, Jerry, Lorraine, and Beth, thank you for your patience and assistance while I took my sweet time getting this done.

My entire family has supported me throughout the journey of my academic career, but none more than my dad. His constant nagging and harassment has pushed me to advance myself in both depth and breadth. By exposing me to knowledge and skills outside my field, I am better able to apply myself to complete projects like this one. If he never has to say, "Just get it done," we will both be the happier for it. I can not thank him enough for his encouragement and patience over these last several years.

Finally, I would like to thank my major professor Dr. Victor Fay-wolfe, without whose constant support this work would not have been possible. His guidance through the mechanics of research and the grant writing process found us the resources needed for this project. I am thankful for his engagement in the success of his students and his commitment to support them in their endeavors. As a beneficiary of this support I could not have reached the milestone that this work represents.

TABLE OF CONTENTS

CHAPTER

Page

APPENDIX

LIST OF TABLES

Table **Page**

LIST OF FIGURES

CHAPTER 1

Introduction

A lack of security training for students and practitioners in networked computer environments has led to breaches of both private data and public trust in the systems that comprise the worlds digital infrastructure. The Open Cyber Challenge Platform (OCCP) provides a low-cost solution for training and education to individuals on forensic and cyber security related topics. Existing challenge platforms use proprietary game control systems that are not readily adaptable for re-use in the OCCP. No suitable alternatives exist to provide a reusable, configurable, open source application for controlling, monitoring, and supporting cyber challenges.

In this project we seek to solve this problem by implementing a foundational game server for the OCCP. A *game server* provides scheduling, network traffic generation, and record keeping for a cyber challenge. An implementation of a game server that can be integrated into new cyber challenges, while remaining re-usable for different types of challenges is presented.

1.1 Goals

The specific goals of this project are aligned to fulfill some of the needs of the OCCP. While the OCCP intends to support a wide range of cyber challenges the goals of this project are a subset of those of the OCCP project. This project will create a game server that is capable of providing the basic needs to conduct a cyber challenge as part of the OCCP framework. These needs include the following:

1. The game server will provide a central point of control for a cyber challenge.

2. The game server will implement execution of scheduled events, and the col-

lection and computation of score data.

3. The game server will implement tools to assist in the generation of realistic network traffic.

4. The game server will provide the needed interfaces to allow for configuration and operation by users of the appropriate skill levels.

5. The game server will be designed to allow for extension of the types of events that are able to be scheduled and scored.

6. The game server will provide a mechanism that supports the creation of additional user interfaces such as scoreboards or complex moderator controls.

1.2 Results

The game server described in this project was successful at meeting the goals that have been set out above. The implementation described herein was able to perform all the necessary functions that a game server should meet to control and support cyber challenges in the OCCP.

1.3 Outline

The rest of this work is described in the following sections. The Background section covers existing technologies that attempt to solve the same problem. Previous attempts at an implementation of this solution are also described. Chapter 3, Methods, describes the approach taken in this project with a description of the implementation details. Chapter 4, Results, evaluates the goals of this project against the implementation described. Finally in chapter 5, Discussion, we review the process and implementation of the solution as well as describe future work in this area.

CHAPTER 2

Background

Cyber challenges are a new tool in the development and education of a skilled technology workforce. They provide an engaging learning environment in which to practice cyber skills and experience realistic situations of cyber attack and defense. The skills exercised in a challenge are needed to defend, secure, and enhance business and government critical digital information systems. Cyber challenges go a step beyond traditional classroom exercises by providing a comprehensive view of how individual skills fit into a larger picture. The *Open Cyber Challenge Platform* (OCCP) aims to provide a low-cost reusable platform for conducting cyber challenges.

A platform on which to run cyber challenges requires several components. Cyber challenges, as described in the OCCP, are usually network based activities, comprised of some number of machines that may represent a small business or similar type of entity. Along with the machines that represent the simulated business services, the platform will require its own supervisory computers and software. This may include machines and software that are responsible for the timing of discrete event executions, network traffic generation, background activity, and simulation of other network users. These activities combined together form the central common component of a cyber challenge, the game server. Historically each challenge platform is custom built for the specific exercise being tested and results in a customized game server for each challenge of each platform.

Cyber challenges focus on the simulation of real world events such as attacks and defenses of computer and network systems. Occasionally a challenge may need to compress the amount of time spent in a simulation as compared to the real world. Events that may take days in real time need to be compressed to just a few minutes in the simulation. Analogous to other sports, a challenge will have a game clock that is disparate from real world time. The clock may be paused, resumed, slowed or sped up to adjust the experience of the challenge.

A game server controls most automated aspects of a challenge. It provides functionality such as event scheduling, network traffic generation, and score keeping. Scheduled events may include simulated user activity such as checking email and browsing websites or computer activity including computer and database backups. The events chosen are specific to the challenge being simulated or tested, while the scheduling and execution of events may be handled generically. Traffic generation is used to give a realistic feeling to a cyber challenge [1]. The creation of background network noise adds realism to the challenge participant's experience of a real world simulation. In addition to the background noise, some of the scheduled events of the challenge are activities which generate score values. Challenges can be scored from different perspectives or with a varied focus. The game server is responsible for recording each of these score events and computing tabulations appropriate for the challenge.

2.1 Cyber Challenge Composition

Cyber challenges may take on many forms. The principle concept is to exercise and teach those skills that are relevant to the security and operation of computer information systems. The classic forms of cyber challenges use a game format to simulate attack-defense, capture the flag and scavenger hunt style games. A challenge may incorporate a combination of both human and automated players. The automated players may be implemented by use of artificial intelligence or simple time scripted actions.

Cyber challenges frequently rely on the notion of grouping players into teams.

These teams are usually given a color coordination that reflects the intent of the team. The teams as described by Vigna in [2] provide a classical example of having red teams attack and blue teams defend. This may not always be the case though. In some popular games such as Team Fotress 2, the attack-defense game modes have the RED team defend and the BLU team attack [3]. The unique color designations of teams is not relevant to the game play of any particular challenge. We choose to use the former classic definitions when referring to team roles.

The concept of teams in a cyber challenge are not restricted only to players, but may be extended to non-player character (NPC) actions. The notion of a Gray Team is used frequently to identify actions or network traffic that are not especially relevant to the game play. Network traffic generated by a Gray Team may be described as background noise or provide a baseline for normal activity. The Gray Team is usually implemented as a set of automated actions, where those actions are not known to the players to be good or bad. Similarly, the White Team may describe moderators or referees that conduct scoring actions that take place during the challenge. In scenarios where players are grouped together in more than two teams other colors or names may be used for those teams as described in [2].

2.2 Related Works

Cyber challenges have existed for several years and most of them use some combination of hardware and software to accomplish the tasks required of a game server. There are several complete cyber challenge solutions such as US Cyber Patriot [4], US Cyber Challenge [5], and SANS NetWars [6]. While each of these solutions provide cyber challenges, they do not make the mechanisms of their platforms available for re-use. The proprietary nature of these applications does not make it possible to use their solution for the proposed problem. Also, the challenges of, [4], [5], and [6] do not allow the user to customize the experience for a particular challenge or to change the scenario for a new topic.

There are open source projects that may appear to provide some portion of the functionality required. Some may include simple schedulers such as cron [7], or more complete schedulers like the Quartz system [8]. These applications, while having the ability to schedule many events, do not encompass all of the tasks of a game server's scheduling needs, nor do they provide the level of control that is required for a cyber challenge. For instance, a challenge may require a departure from normal run time, such as, a challenge needing to be paused to allow users to catch up or be provided breaks or instructions. It may also be necessary to change the speed at which a challenge operates so as to increase or decrease the difficulty of a challenge.

One purpose of scheduling events is to generate network traffic. That traffic may be harmless or malicious. The BreakingPoint devices from IXIA [9] offer a complete solution for traffic generation of the type normally used in cyber challenges. These devices though are not specifically designed for use within a cyber challenge. They are of considerable expense, $50,000 dollars, and are not easily extended by custom applications and modules, which is a goal of this thesis project.

The Open Cyber Challenge Platform (OCCP) is a new platform being developed at the University of Rhode Island [10], that will advance the use of cyber challenges as a tool in teaching and assessing users. The goals of the OCCP are to address the cost, complexity, and other barriers to entry of using cyber challenges. The project aims to be a free to low cost. The various components of the OCCP will be implemented without the use of proprietary hardware or expensive commercial software. This project will be one component within the OCCP.

The OCCP defines several types of users for the platform: end users, scenario designers and application developers. Each of these user types categorizes activities that would be conducted when using the platform. The *end users* of the OCCP are the moderators and players that use the framework to conduct cyber challenges for learning or exercise. These users may not be particularly well versed in the inner workings of the platform, but they are interested in using the platform as a tool. The *scenario designers* construct, modify, or rebuild individual cyber challenges to be conducted using the platform. These individuals may be subject matter experts or educators that want to build new challenges for themselves and others to use. The product of a scenario designer's work is used within the OCCP to conduct a cyber challenge for the end users. The *application developers* contribute directly toward the enhancement of the OCCP as a platform. The work of a developer may contribute toward some or all of the components within the OCCP. An application developer may extend the functionality of the game server or build companion applications that integrate with the platform.

This project is inspired by two previous attempts to construct a game server for use in the OCCP. The first implementation centered around the use of the Nagios monitoring software [11]. The project implemented each aspect of the game server using both independent scripts and Nagios host agents. The Nagios server acted as a collection point where all data from the various components was reported. This method provided great flexibility and allowed for re-use of the Nagios infrastructure. Nagios supports many different types of host monitoring agents as well as the ability to receive messages from remote processes. Nagios provided system monitoring for the players, as well as a scoring mechanism for the moderators.

The initial tests with Nagios were successful. Monitoring of various components and score reporting worked well. Difficulties appeared when increasing the rate and volume of system monitoring and traffic generation. The initial attempt tried to use Nagios to both generate background traffic and monitor the results. Under normal circumstances for Nagios this activity is expected and functional. The difficulties arose when trying to simulate large volumes of traffic. Nagios' ability to perform behaviors like an end user helps administrators simulate and monitor the use of their systems. For example, Nagios has the ability to load a web page from a server and test the web page for various factors. This can be programmed on a schedule to repeat on a regular basis. As the tests succeed or fail, Nagios will report relevant information to the administrator. This ability is similar to the desired behavior in a cyber challenge and prompted the early trials of Nagios as the core to a game server implementation.

The Nagios platform was not designed with load testing or large scale traffic simulation in mind. Therefore the scheduling abilities of Nagios when attempting to generate large amounts of network traffic were insufficient to generate believable network traffic. Even when attempting to load a web-page several times a second Nagios performance would be unacceptable. Several work arounds were proposed, but all proved to be insufficient.

The second attempt to provide game server functionality to the OCCP revised the initial attempt by removing Nagios. The attempted solution required several independent programs to be run concurrently. The programs consisted of a set of commands that corresponded to some function of the game server. Each program would operate independently of the others. This approach highlighted a desirable property of a game server-like solution. This method takes advantage of parallelism. Each of the component applications operated independently in this attempt. Some game server operations have long wait times for input/output operations to complete. This method increased throughput of desired events resulting in more realistic network traffic generation.

Unfortunately, this approach did not allow for discrete control over individual functions of the game server. For example it was not possible to pause a challenge once it had begun, without restarting from the beginning. The ability to dynamically add or remove events was also not possible with this implementation. The user would be required to launch several terminal windows and run several command line applications to begin running a cyber challenge scenario. An improved solution would include both the centrality of the first attempt and the parallelism of the second.

2.3 Technologies

There are proprietary technologies available that may seem to provided a solution for the proposed problem. A solution could be proposed that is implemented primarily with hardware or some combination of hardware and software. This proposal is focused on a solution that does not use a costly hardware solution but is completely implemented in software. This constraint is a requirement of the larger OCCP project. The technologies selected have been identified as good candidates because of their low cost, ease of access and usefulness in solving specific technical challenges of the proposed solution. Certainly more than one solution is possible for the given problem, but the proposed solution will likely utilize some or all of the following technologies.

A family of operating systems, known as Linux systems, are based on the early UNIX platform. There are many variations of a Linux operating system, commonly called distributions. Each system uses a similar structure and method of operations. The commonality between various distributions is the Linux kernel. The kernel is the core application of a computer and is always running [12]. It provides mechanisms for all other applications to interact with the computer. The Linux kernel offers certain functionality, that is not present in all operating systems.

Modern Linux systems offer a spectrum of light weight virtualization by providing more granular access to services of the kernel. Containers and Namespaces are two such services that are provided by the kernel to facilitate virtualizing applications [13]. The use of these services allows the execution of an application to be contained from the host system without requiring the need for a complete guest virtual machine. This greatly reduces the memory and computation overhead of using virtualization to accomplish certain tasks.

Virtualization is used in many applications to reduce equipment costs and infrastructure expenses [12]. Virtualization can be accomplished on many different levels. The most commonly encountered type of virtualization is whole machine or whole operating system virtualization. In this type the entire function of a computer is replicated in software. A computer which runs this type of virtual machine is known as the host. While the computer that is being virtualized is referred to as the guest or *virtual machine*. An alternative to complete machine virtualization is a type of virtualization that may only encompass a single application or specific piece of an application. The benefits of light weight virtualization were heavily utilized within this project to reduce or eliminate the need for dedicated hardware.

An application programming interface (API) describes the behavior, inputs and output of a software component or application without describing its particular implementation [12]. An API can be added to an application to make the application or software component more extensible. While an API describes a set of rules to be followed, a specific implementation such as through a web service is required.

Web services expose an application's API over a network using one of several methods, such as the frequently used Hypertext Transfer Protocol (HTTP). Historically, Simple Object Access Protocol (SOAP) has been widely used to implement many of the first web service APIs on the Internet [14]. More recently the trend has led to web resources centered around the direct representational state transfer (REST) method. REST was first described by Fielding and Taylor in [15] although it does not prescribe an exact implementation. Application programmers that use this method on the Internet often describe their implementation as REST-ful or REST-like, indicating that their implementation may not completely conform to all REST practices.

REST takes advantage of the mechanisms of the underlying transport protocol HTTP. REST uses the HTTP verbs such as GET, PUT, DELETE, UPDATE, etc to describe the action being sought. The specification does not prescribe the encoding of the data to be transferred, but most applications use either extensible markup language (XML) or Javascript Object Notation (JSON) to encode the data being transferred. This is not a requirement though and other technologies such as MessagePack [16] have evolved to increase performance when transferring large amounts of data or transactions. Use of JSON has gained in popularity because of both its efficient storage and human readable text.

The technologies described here will be used to build our solution application. We will build upon the previous attempts at a game server by incorporating the lessons learned from those projects. Our project will take advantage of existing resources to enhance the speed at which we develop and test our solution.

CHAPTER 3

Methods

The OCCP *game server* is implemented as an application written in the Ruby Programming language. The application, while intended to meet the needs of the OCCP platform, may be used as an independent application. The game server does not immediately depend upon other components of the OCCP platform. This allows the development of the game server to move at an independent rate. The functions of the game server support cyber challenges using any platform the operator chooses.

The game server was developed using a hybrid combination of both Agile [17] and rapid application development methodologies [18]. This process eschews with the development of extensive documentation used in the waterfall method. Our implementation strategy is to produce working software over comprehensive documentation at the outset. As such, the methods we describe in this section are a combination of postmortem analysis of previous experience and our designs.

We begin by describing component blocks of the game server application. The application interfaces are described first, followed by the top level concepts of a Team and an Event. Next we describe the game server control of a scenario and the components that are used to support it. Finally we describe a procedure for testing the game server's ability to satisfy the goals of this project.

3.1 Interfaces

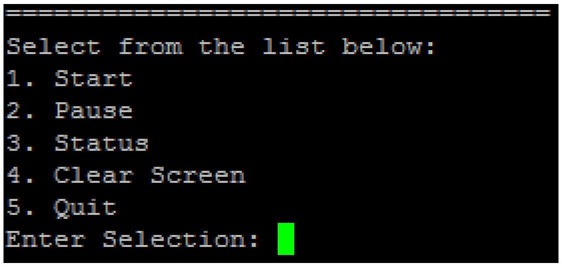
The game server implements several types of interfaces to be used by the various types of users. The *instance file* is an interface that allows for the definition of many functional aspects of the game server and it is the primary interface. The command line interface, or CLI, is used to provide run-time interaction with the program for all types of users. The two application programming interfaces (API) are made available to extend the usefulness of the game server by allowing external applications to interact with the game server and to extend the game server with new functionality. These interfaces provide a diverse set of mechanism with which to configure and interact with the game server.

3.1.1 Instance File

The instance file is an XML document that describes the configuration and run-time directives for the game server. The instance file, as described in the OCCP, is input to the game server that is meant to describe a single instance of the cyber challenge that is to be conducted. The game server parses this file and extracts information such as: the length of the challenge, which events should be triggered and at what times they should occur, how to score individual events, as well as other pre-configured parameters that are relevant to the challenge. The game server ensures that all required parameters are provided in the file before beginning a challenge. This step ensures that the challenge will not be stopped, after beginning, due to a configuration error. Any errors detected by the game server are reported and direct the user to the faulting line of the file to speed development of new instance files. An example file is provided in Appendix A.

3.1.2 Command Line Interface

The CLI becomes available to the users during run-time operation, after the program has successfully initiated and completely read the instance file. The CLI provides a very simplistic user experience. The objective of the CLI is to allow the user to retrieve basic status information and to perform basic challenge actions such as Run, Pause and Stop a challenge. The CLI is intended to be used only by moderators of a cyber challenge. While these users of the OCCP will have access



```
==================================
Select from the list below:
1. Start
2. Pause
3. Status
4. Clear Screen
5. Quit
Enter Selection:
```

Figure 1. CLI waiting for user input

to the console of the computer running the game server application, a graphical front-end would be a preferred method of access to the game server.

The CLI is implemented using a read-evaluate-print loop (REPL). The user is presented with several options to choose from as shown in Figure 1. Each time the user selects an option from the list, the program will take the appropriate action. Any output from the selected action will be displayed on the screen and the menu choices will be re-displayed for the user to select another action.

The command line interface provides a flexible method for interacting with the game server application. The text-only nature of this interface allows it to be remotely controlled through secure shell (SSH) or via the console of the computer running the application. This remote access is desirable within the OCCP where many of the application servers are virtualized and console access may be difficult or impossible. While the game server does not directly make the CLI available remotely, it does provide a web service interface to access the game server from another application or host.

3.1.3 Web Services

A RESTful web service API is exported by the game server to enable extension of the game servers capabilities. The API exported by the game server provides access to (1) the information parsed by the game server from the instance file, (2) run-time information and (3) the ability to affect changes in the game servers

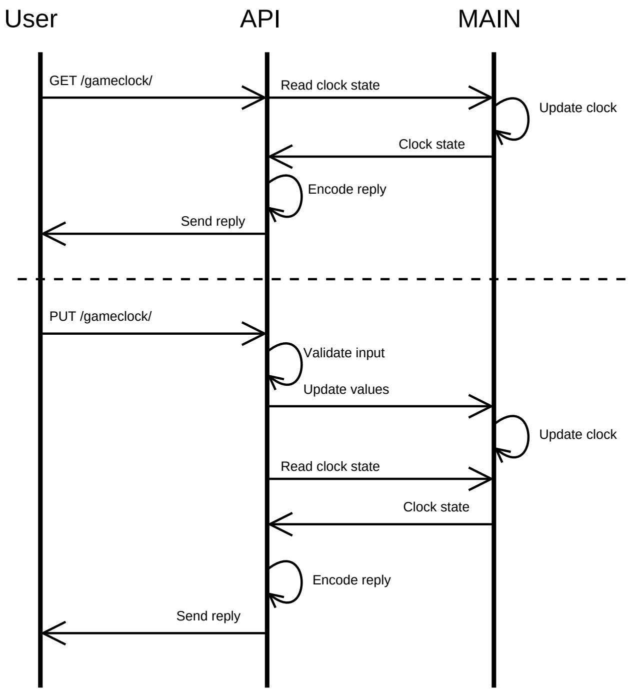


Figure 2. User API interactions for reading and updating values

execution. This interface is exposed through RESTful HTTP transactions. This allows external applications flexibility when interacting with the game server and allows them to extend the functionality offered through other interfaces.

The web service is designed around the notion of accessing and updating entities. Each entity is described by some uniform resource identifier (URI). This model allows authors of external applications to interact with the game server in a well defined manner. For the example shown in Figure 2, the current value of the game clock can be accessed through a URI similar to *http://server-dns/gameclock/*. A HTTP GET request to this URI will provide immediate access to the length of the scenario, the current time elapsed in the scenario and the current state of the game clock (e.g. running, paused, stopped). In this example the game clock may be updated using a HTTP PUT request, to reflect either a change in the length of the scenario or the state of the game clock.

Implementation of the API functionality is handled using an external library that facilitates low-level tasks such as handling the HTTP request-response mechanism. The Sinatra [19] library uses a domain specific language to describe the actions of each HTTP verb and URI combination. The use of this library provides for separation of duties within the implementation of the game server. Changes can be readily made to the API without affecting future development of the game server. The implementation details of the API will track the changes made within the game server for a top-down design paradigm. The library implements a small HTTP server in a separate thread from the main functionality of the game server providing an event driven (HTTP request) interface that does not cause blocking or interruptions in processing other actions.

3.2 Teams

The game server is designed using a multi-threaded implementation that allows individual components of the game server to operate in parallel. Parallel computation provides separation of duties between components and adds resilience to the program overall. Several of the main functions of the game server are separated into their own threads of execution. After the program is initialized the first thread to be separated is called the MAIN thread. This thread is responsible for maintaining control over the program's core data structures. The second thread to be launched is the RESTful API web service. This thread operates independently until the program terminates. The primary flow of execution now enters into presenting and responding to the command line interface. Most of the work of the application now flows from the MAIN thread as shown in Figure 3.

The program is structured relative to the design of the challenge as described in the instance file. The game server provides scenario designers several mechanisms to organize events and their related data. The most significant construct is the notion of a *Team*. In the context of the game server a Team is a grouping of events

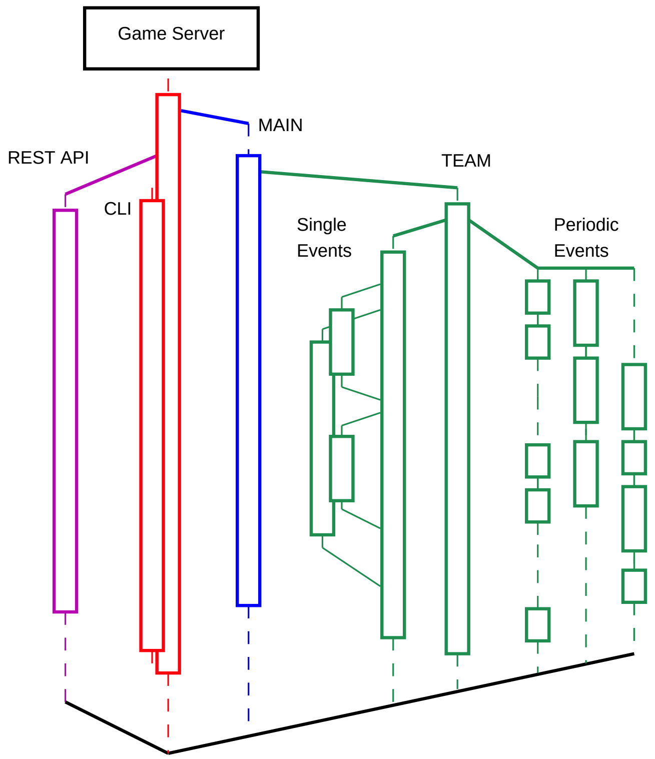


Figure 3. Thread relationships and separation of duties

that should be treated as a collection of similar activities. This concept of a team is derived from the concepts as described in Section 2.1. Each Team is treated as a separate grouping of threads using a hierarchical relationship. Each thread (CLI, MAIN, TEAM) shown in Figure 3 is designated as the responsible master for the data contained within it. When another component of the system needs to update a piece of information it must talk to the owner of that information.

This model provides some desirable traits for the game server to provide fine-grain control over groups of events. For example, consider a red team composed of attacking events and a gray team composed of events that simulate background traffic. A moderator may wish to stop the attacking events while permitting the background traffic to continue. By grouping the events together they can be controlled collectively providing the desired effect. After spawning the appropriate threads both the MAIN thread and any TEAM threads enter a processing loop that provides for inter-thread communication.

When using a multi-threaded approach communication between competing threads of execution must be handled very carefully. During the program's main execution the MAIN thread and any TEAM threads need to pass messages between the different components of the system. These messages include information related to scoring, logging, and system state. A thread-safe queue implementation was chosen to allow communication between different threads of execution without causing loss of data due to mishandled shared memory structures.

Each responsible thread listens to the top of its own queue like an inbox. Any other component can push a message onto a thread's inbox queue for it to process. The receiving thread will pick up each message off the queue in sequence, perform any necessary actions, and then return to the queue for the next message. In between messages the thread sleeps without polling. This reduces the overall overhead for the application by creating an event driven communication system between threads.

The messages passed between the threads use a message object format. Rather than just pass arbitrary strings or binary data each message is uniformly formatted into an object representing the message. The message object contains headers like sender and type of message as well as the body of the message. Using a uniform message format allows new message types to be introduced without breaking components that do not understand that type of message.

3.3 Events

Events as referred to here identify discrete actions that should be completed by the game server on behalf of the scenario designer. The events are defined within the instance file of a challenge description. These events may execute a custom program or perform some network activity like sending an email or pinging a machine. The *Events API* as described here allows the developer to extend the game server to natively support new Event types. This section describes the options that we have implemented in the game server to give the developer flexibility in defining events that may be used by the scenario designer.

The events implemented in the game server follow an implementation that combines both the Factory and Builder [20] patterns. Events are implemented as an abstract class that defines an interface to be sub-classed. This allows developers to extend the application with additional native event types. Each concrete event class must be implemented separately before it is able to be used within a cyber challenge. The abstract event class defines the required minimum attributes that an event object must possess for the game server to interact with the event. The required attributes are related primarily to scheduling the event within the challenge. Each concrete subclass may implement any additional attributes or methods that are relevant to that type of event.

A challenge designer encodes the events for a scenario in the instance file as XML descriptions. Here we use the Builder pattern to separate the encoding of the event from its instantiation and operation. After a concrete subclass has been established it is necessary to construct a handler for that event type. Each event as described within an XML block must specify the handler that is used to parse the event. The handler provides two functions: *parse-event()* and *run()*. The *parse-event()* function is passed the XML block description of the event. It is the responsibility of the parsing function to decode the XML of the block and instantiate the concrete event class that it describes. This allows the developer to decide both the attributes of the event as well as the specific XML that describes the event.

The handler also provides the *run()* function that is used to execute the event.

The function is passed the instantiated event object containing all the information decoded from the XML block as well as a handle to the application. The programmer of the handler decides how the event should be run, additional function calls to be made, or external applications to interact with. When the game server calls the run function the execution is passed into a new thread as described in Section 3.4.2. The return value of the run function is not monitored by the game server and it is the responsibility of the programmer to handle any errors. The application handle that is passed to the function is used to communicate with the rest of the application. Using this handle the programmer can send score results and log messages. This provides encapsulation of the events so that they do not interfere with one another.

3.4 Game Control

Cyber challenges, as envisioned within the OCCP framework, are designed to have discrete time frames. These time scales are described as a length of time to accumulate on the game clock, without specifying a particular start or stop time. This facet of a cyber challenge requires that the game server perform at minimum two specific functions. The first is that the game server must control the starting and stopping of the challenge. While this may seem like an obvious point, the application could alternatively be programmed to begin execution of the challenge as soon as it is launched as it is in [4] and [6]. The second function requires the game server to track the amount of time that has elapsed in the challenge. This information is used to stop the challenge once the given game length has been reached.

The desired situation envisioned by the OCCP framework has an expectation for deliberate control of the challenge. For example, a moderator may wish to control exactly when the challenge begins and ends. Each definition of a cyber challenge will provide some guidance on the length of that challenge. Usually this will be provided by the designer of the challenge using a relative time scale, such as 30 minutes or 3 hours for the length of the challenge. In a practical situation the cyber challenge will need to be setup and configured before the users or players are ready to begin. In a classroom environment this may mean that an instructor has configured and prepared the challenge before the beginning of the class period. For a higher skill level challenge it is likely that there will be a moderator who is preparing the challenge before the players arrive at the scenario. In both of these situations it is desirable to have a discrete START and STOP function so that execution will not begin before game play is ready to commence.

The game server is implemented to first read the provided instance file that describes the challenge. As the game server reads all of the configuration directives and commands that are relevant from the instance file it will begin parsing and executing those instructions to set up the challenge environment. Once the setup phase is completed the game server will stop and wait for user input before continuing its execution. State changes in the game clock may be generated either through the CLI or through the web service API.

3.4.1 Game Clock

The game clock acts as the authoritative source for time information throughout the entire application. This is an important feature to maintain an accurate auditing of the events and their order of execution. The game clock is a construct designed to allow the challenge to be played at arbitrary intervals and speeds. Like other games such as football, the game clock is separate from normal time or wall time. The wall time of a game may be several hours long while actual game play may be only half of that time. This discrepancy is caused by various motivations. Instructors or moderators may wish to describe or breakdown various components

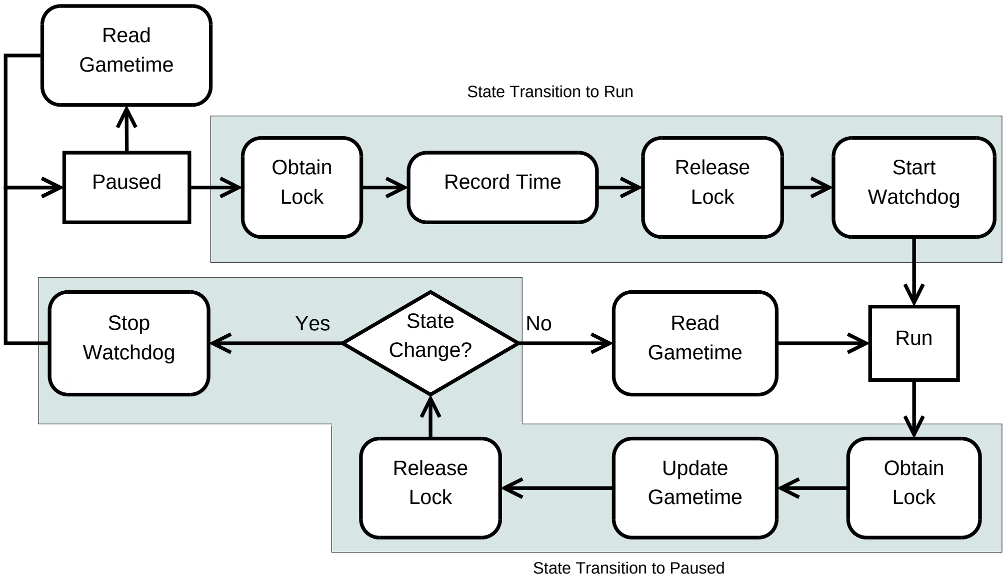


Figure 4. Game clock states during operation

of a cyber challenge during game play. This means that the moderator would stop game play by pausing the game clock, while the wall clock or normal time continues unabated.

The game clock is implemented using a stateful object representation. The clock object maintains an accessible state such as RUNNING or PAUSED and a time value counter. When the game clock undergoes a state change, such as RUNNING to PAUSED the *gametime* value is updated to represent the amount of time that has elapsed on the game clock. As shown in Figure 4, the gametime updates when the clock is transitioned between states or while being read in the RUNNING state. A watchdog process monitors the amount of time elapsed in the game. When the gametime reaches the value described in the instance file for the length of the scenario the clock is stopped. Because the game server is designed as a multi-threaded application the game clock's time value is protected from alteration by a mutex. Many components of the system may try to read the game clock simultaneously. This design, while simple, provides consistent simultaneous access to the game clock for each independent component.

3.4.2 Event Execution

The timely execution of events defined by the challenge is a core function of the game server. The specification of events includes the information necessary to compute when and how the events should be triggered. This information also describes whether the event occurs a single time or whether it occurs many times as a periodic event.

Events can be placed into two general categories, those events that occur only one time during a challenge and those events that are recurring or periodic. The designer of the scenario decides into which of these two categories an event may fall when specifying the event. In some cases a designer may want an event to occur a small number of times such as two or three. They may choose to specify the event the same number of times or the event may be specified as a periodic event. In these cases there is no distinction between a singular or periodic event. Singular events are specified individually in the instance file and may have unique options, whereas periodic event description are executed repeatedly. Each event is described by an XML block that, for a singular event, will be executed at most one time. Similarly, a single XML block may be used to describe a periodic event that may occur several times per scenario or several times per second. This distinction allows the designer to generate many repeating events. An example of the details of an event definition are shown in Appendix A lines 131-134.

Singular and periodic events are treated differently when being executed by the game server. The events are first separated into two lists for each Team. The initial list separation is performed during the setup phase of the game server before any user interaction has occurred. Singular events are stored in a priority-sorted list with the next time-to-execute event at the front of the list. Periodic events are all processed directly at the start of the challenge. The list of singular events and each periodic event is monitored in its own thread as shown in Figure 3. These monitoring threads are one component of the game server that requires the constant high precision of the game clock to function properly.

The singular events can easily be sorted and maintained by the next time-to-execute of each event. The top of this list is monitored by a single thread that, when running, spends most cycles sleeping for an amount of time that is equal to the difference between the gametime and the next events time-to-execute. At the correct time the event at the top of list is removed by the monitoring thread and launched into its own new thread. The monitoring thread then evaluates the next event and either launches it or goes to sleep for the appropriate amount of time. This method allows events to be quickly pulled off of the list and launched without long delays even when an event's specified launch times are very close together. By placing each event launch into a new thread the monitoring thread does not have to wait for the event to return. When the event does eventually complete any remaining tasks can be processed by that events thread without disrupting the monitoring thread.

The periodic events are handled slightly differently than singular events. Periodic events may specify both a rate of repetition and an amount of drift. The drift of a periodic events adds or subtracts a random amount of time from the expected next launch time. For example, an event may specify a repeat rate of every 5 seconds with a drift of ±2 seconds. Periodic events with both a repeat rate and drift would create a burdensome amount of computation to maintain an ordered list of events sorted by their next time to execute. Instead, immediately after the events are split into the periodic and singular groups a separate thread prepares each periodic event for execution. One new monitoring thread is created to schedule the individual execution of each periodic event. These monitoring threads are nearly identical to the one described for singular events. It is responsible for launching the event at the correct interval as specified by the designer.

Creating a scheduling thread for each event allows us to take advantage of parallelism. This allows events with high frequencies to be executed simultaneously without a large amount of computation to keep the events in a sorted list. This use of multi-threading allows individual schedulers to operate unimpeded by other functions of the game server including executing other events or responding to score queries. Using the lessons learned from the Nagios implementation described earlier, this multi-threaded approach was chosen to allow events to be executed simultaneously rather than sequentially.

The monitoring threads of both the periodic and singular events are dynamically controlled when the game clock undergoes a state change. This is accomplished through a combination of native thread controls and inter-thread message passing. The TEAM thread is denoted as the parent thread for each monitoring thread as shown in Figure 3. When a state change occurs the TEAM thread uses native thread commands to wake up all sleeping threads. As each monitoring thread awakes, it performs three checks: (1) it compares the amount of time just slept to how long it should have slept for, (2) if awoken early check the Team state, and (3) if in a PAUSED state record how long it has already slept before finally performing a thread yield. After completing a thread yield the monitoring thread will not execute again until specifically launched by its parent thread. This overrides the normal behavior of the system thread scheduler which may wake up the thread at the next execution cycle. Similarly, a transition to the RUNNING state signals the Team to wake up each of the monitoring threads for that Team. As each thread wakes up it checks to see if there is any remaining time balance that it should sleep for before the next execution of the event. This preserves any complex timing scenarios that existed before the last state transition.

The monitoring threads do not check the state of the game clock directly and instead watch the state of the Team. The state of each Team is globally synchronized to the game clock when a game clock state transition takes place. Individually though, a Team may have its state locally changed so as not to be in sync with the game clock. This feature allows individual teams to be paused or run without affecting other parts of the game server. In some scenarios it may be useful to pause one type of event that are collected together in a single Team. For example, a moderator may wish to continue background traffic while disabling any attacks by the red team. This fine grained control is achieved by passing a message directly to the desired Team indicating that it should perform a state change. No other part of the system is affected when a Team state change occurs locally.

3.5 Scoring

Scoring of cyber challenges presents unique difficulties, and no ideal solution has yet been presented in the field. In this implementation for the game server we provide a mechanism to provide extremely granular control of the recording of score events. How these events are combined and used to evaluate player understanding is beyond the scope of this project. The scores for individual score-able events are collected and calculated using the descriptions provided in the instance file by the scenario designer. This model will provide a mechanism to permit the evaluation of new scoring methods without requiring gross adaptation of the core of the game server.

Each event launched by the game server is able to be scored individually. The most basic categorization is whether the event was successful or a failure. A *score-event* is an atomic unit that describes a point value and a category label. An

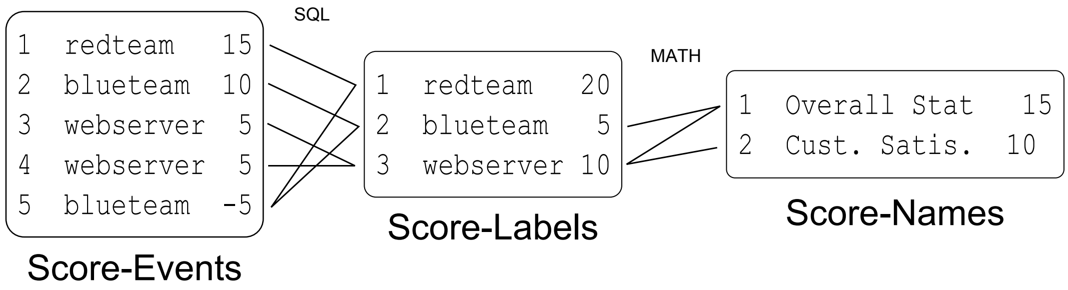


Figure 5. Relationship between scoring elements

event may define one or more score-events to be recorded if the event is successful or fails. This allows one event to add points to one or more categories each time it is completed. For example, if an event is successful it may add points to one category and subtract points from another category in the same operation. The naming of the categories is arbitrary and only serves the designer when creating a new computational model.

Each of the score-events is recorded in a SQL style table for accessing using arbitrary SQL queries. The game server accesses this table using a *score-label*. The default definition of a score-label is a SQL query that sums the values of all the scores with the same label. This provides an initial starting point for the designer. The instance file is capable of overriding this default SQL with any valid expression that returns a singular value. This allows the designer to perform complex SQL operations on scores that are recorded by the system. For example, the designer may choose to create a score-label that represents the values recorded in the last 5 minutes of the challenge or the scores of a 10 minute period that was 15 minutes previous.

The game server provides one more layer of computation to aid designers in creating meaningful scoring results of the challenges they design. A game server *score-name* is a mathematical expression that combines one or more score-labels as described previously. A score-name can use basic mathematical operations to create a final score from the aforementioned labels. Figure 5 shows the progression from score-events to final scores. An example of how these constructs are defined can be seen in Appendix A on lines 20-40.

This method of calculating scores should provide sufficient complexity to allow a designer to score a challenge in any manner that fits his or her needs. The final score-names are the values that are reported by the game server during competition, but the intermediate values can be recomputed from the original score data which is retained after the challenge is completed.

3.6 Creating Realism

Cyber challenges attempt to provide the players with a simulation of a situation that may be encountered in the real world. As such, one goal of this project is to provide scenario designers with tools to assist them in making scenarios which feel realistic. Below we describe two tools provided by the game server that may be used by designers to create realistic patterns in their scenarios. The first is Internet Protocol (IP) address randomization, which allows events to be specified with a selection of different IP addresses. The second is fine grained time control which allows the creation of non-repeating patterns.

3.6.1 Address Randomization

Every event that is triggered by the game server relies upon network communication to accomplish a described task. This network communication will originate from the host machine upon which the game server application is running. Many, if not most, applications expectantly use the default IP address of the host on which they are executed. In fact most applications that adhere to the Open Systems Interconnection (OSI) model [21] are not aware of the underlying communication protocols being used. The game server seeks to enhance these applications by providing a contained environment for them to execute within. Various aspects of the environment can be controlled by the game server to alter the behavior of the contained application.

This technique relies on the use of the Linux kernel *namespaces*. Namespaces were designed to provide isolation for processes running on the same kernel [22]. The use of namespaces in the game server application provides a type of light weight virtualization for the processes launched by the game server. The game server is able to programmitcally control environment settings that affect the launched applications. The namespaces container used in this implementation allows for control of the network stack within the kernel. Each event that is triggered by the game server is launched inside a namespace container. All containers are identical except for the IP address that is associated with the container.

The instance file provides an XML syntax for specifying a named pool of IP addresses. The address specification may be any combination of single addresses, list of addresses, or entire classless inter-domain routing (CIDR) notated blocks of addresses as shown in Appendix A on lines 70-89. An instance may have an unlimited number of IP address pools that contain an unlimited pool size. Any event described in the instance file may either specify a specific IP address or choose to have an address randomly selected from a named address pool.

Once an address has been assigned for an event execution a kernel namespace is created for that address. The kernel supports a fewer number of namespaces than IP addresses that exist. For this reason a namespace is not created until it is time to execute an event with a specific IP address. The game server creates the namespace and applies the IP address into the environment. The event is then executed inside that namespace. The game server tracks all namespaces that have been created and how many events are using the namespace as their execution environment.

A namespace that has been created and used by one or more events, but is no longer being used becomes a stale namespace. Each namespace that is tracked by the game server has an adjustable finite lifetime. If a new event wants to use the IP address associated with a namespace before its lifetime has expired, the namespace is moved from the stale list. The lifetime on the namespace is reset when the event is finished using it. At the end of a namespace lifetime the namespace is deconstructed to free resources within the system.

3.6.2 Timing Control

The timing of a sequence of events that occur in a cyber challenge presents another avenue to create a realistic feeling for the scenario. Events that are specified as repeating events must have an associated period at which they repeat. An amount of drift can be added to the event period to cause the event period to lengthen or shorten. This allows the event to be executed without a defined time signature that would be indicative of a simplistic automatic scheduling system.

Here we implement a drift that is added to the event period at each invocation. The drift, specified in seconds, represents the outer bounds of an interval in which to choose the drift value. The interval is randomly chosen at each event execution, and added to the statically defined period. This new value is then used to sleep the event between executions. This allows a scenario designer to choose repetition rates and drift intervals to create slightly abnormal timings or completely randomly timed event executions.

3.7 Testing and Metrics

In the previous sections we have described a solution to the problem outlined in Chapter 1. We described, in the context of the game server application, the functionality that is required to meet the goals of this project. This section identifies and explains the tests we conducted to evaluate the solution and the metrics we used to evaluate the effectiveness of the solution in meeting the goals. We begin by asking questions about the game server and determining the metrics we need to collect to answer those questions.

Question 1: Does the game server provide the ability to control the running states of a cyber challenge?

Start time, End time, Total wall time, Total time paused

These timings can be used to compare the expected and actual values associated with a scenario. The total wall time for a scenario, that has its state changed to PAUSED, should be extended by the amount of time the challenge is paused. The start time and end time indicate if the scenario runs for the correct amount of time.

Question 2: Do scheduled events execute at the correct time?

Actual execution time of each event

The actual time that an event is executed should match the values specified in the instance file for that event. Periodic events should execute at the rate specified in the instance file.

Question 3: Are scores for executed events correctly recorded?

Score-event records

The individual score-events that are recorded for each event should match those specified in the instance file. Periodic events should have a matching number of executions and score-event records.

Question 4: Do the final score results match the expected results?

Final score-names

The final calculated scores are defined as score-name calculations within the instance file. The recorded results of the score-names should match the final score results that are calculated separately by evaluating the instance file for the scenario.

Question 5: Do events that specify an IP address correctly use that address?

Question 6: Are events that specify a randomized IP address executed with different addresses?

Game server network traffic

The network traffic exiting the game server records the IP addresses assigned to events specified in the instance file. The events that specify an IP address should be detected within the network traffic analysis with that address. The events that specify a random address should be detected as having an address from within the specified address pool.

Question 7: Does event timing randomization adequately mask the signature of a periodic event?

Event execution time line

One of the goals of this project is to provide scenario designers with tools to create realistic network traffic. This test measures the game server's ability to achieve that goal. The event time line is compared with expected values for periodic events with and without an associated drift. The time between event executions for events with no drift should appear regular, with a frequency that is the same as that specified in the instance file. For events with a drift value, the timing distribution should show timings not occurring at a particular period, in contrast to non-drifted events.

Question 8: Can the game server execute the types of events that represent typical network traffic of a cyber challenge?

Game server network traffic

> The events defined in a scenario's instance file should be executed and detectable within the network traffic generated by the game server. The types of events defined in the instance file are representative of those encountered in other cyber challenges. This test measures the game server's ability to generate realistic network traffic.

Question 9: Do the interfaces of the game server provide the required functionality for each type of user?

Game server interface definitions

> The expected activities of each type of user was compared to the functionality specified by the interface definitions. Each type of user, as defined by the OCCP, should be able to perform all functions for their required skill level in the appropriate interface.

The metrics data to answer the questions presented above were collected using several tests. Each test requires the use of an instance file to specify the actions the game server should perform. Some of the tests require the comparison of expected data to actual recorded data. The instance file for a specific test is the source for the expected data. The data in the instance file may require additional computation performed outside of the game server to compare with recorded results. Some tests may produce more than one data metric to be collected simultaneously.

3.7.1 Game Server Operation Testing

The procedure for each test involves using the game server to run a specifically crafted instance file. The game server log file and score record database was collected at the end of each operation. At the end of each test we analyzed the collected information to generate the results of that test.

3.7.2 Network Testing

We utilized a virtualized target machine when conducting network tests. The target is configured to send and receive traffic from the game server. These two machines are the only computer hosts on the network to reduce network noise and aid in the correct identification of events. We used *tcpdump* [23] to collect the network traffic emitted by the game server and the target machine.

The testing for address assignment was conducted by using a single ping [24] sent from the game server to the target machine. The target machine should receive the ping and reply with the appropriate response. The game server should receive the reply message indicating that it has not only sent the ping with the source address but it is also listening on that address. The test was conducted for both single events and periodic events. The single events were executed by specifying a single address to use. Each of these events should only use that address when conducting the test. In the case of periodic events, each event was specified as using an IP address pool. The game server should randomly select an address from the pool and assign it to the event. If the game server fails to receive a reply then the game server has not correctly established a method for specifying IP addresses for events.

The traffic generated from the game server and from the target machine was compared to determine if the test was successful. Because additional network traffic may be present on a connection, the events were separated by an appropriate time interval. This eases comparison between the two sets of traffic and allows for the unique identification of events.

CHAPTER 4

Results

In this section we discuss the results obtained from development and testing of the game server. We revisit the questions asked in Section 3.7 and describe the implications of the metrics collected.

Question 1: Does the game server provide the ability to control the running states of a cyber challenge?

Run Time	Time Paused	Total Time	Actual Run Time
300	60	360	300

Table 1. Comparison of scenario run times

The game server was successfully able to control the running states of a scenario. As shown in Table 1 the actual run time of the scenario is equivalent to the desired run time. The game server provided the ability to pause a running scenario and to continue that scenario at a later time. The actual amount of time that the scenario was executed for matched the amount defined in the scenario file processed by the game server.

Question 2: Do scheduled events execute at the correct time?

Scheduled	Start Time	End Time	Event Name	Status
20	20.0936	20.1122	Event 1	SUCCESS
45	45.0016	45.0126	Event 2	SUCCESS
90	90.0793	90.0850	Event 3	SUCCESS

Table 2. Non-periodic event times and actual execution times in seconds

All events scheduled by the game server were executed at the times specified in the instance file. The data provided in Table 2 show a sample of multiple runs conducted with non-periodic events at various starting times. In each case the game server correctly launched the event at the specified time. Periodic events require an additional dimension to analyze. For the case of periodic events we also review the number of times that the event is executed with respect to the frequency specified for each event. In Table 3 we show the Start and End time for an event along with its associated frequency. The frequency describes the number of seconds between executions of the event. The table shows the number of events that should be executed for the given values of duration and frequency.

Name	Start	End	Freq.	First Run	Last Run	No. Events	Actual
Event 1	0	30	1	0.1545	30.0310	31	31
Event 2	30	60	1	30.0012	60.0124	31	31
Event 3	0	120	4	0.0785	116.0220	30	30
Event 4	0	120	0.5	0.2384	119.6288	240	240

Table 3. Periodic event times and actual execution times in seconds

We can see from the values presented in Table 3 that the game server has executed the periodic events in agreement with their specifications. The length of the test scenario was scaled to 120 seconds overall. In the data for Event 2 we can see the game server both correctly delays starting the event and correctly stops the event at the specified times. Events 3 and 4 have end times that coincide with the completion time of the scenario. The game server will execute one less event for Events whose end time is at or greater than the length of the scenario. The game server stops all executions at the beginning of the last second of a challenge, hence any event whose last execution occurs within or after the last second will not be run.

The data from single and periodic event executions show that the game server is able to execute events at the requested times. We also see that edge cases exist when an event is scheduled to run until the end of the scenario. These edge cases when documented do not impact the functionality of the game server.

Question 3: Are scores for executed events correctly recorded?

The game server stores each record for a scoring event in a table that identifies

Name	Runs	Pt Value	No. Records	Expected Pts	Total Pts
Event Count 1	1	1	1	1.0	1.0
Event Count 2	1	15	1	15.0	15.0
Event Count 3	1	0.5	1	0.5	0.5
Event Periodic 1	31	1	31	31.0	31.0
Event Periodic 2	31	3	31	93.0	93.0
Event Periodic 3	30	7	30	210.0	210.0
		-4	30	-120.0	-120.0
Event Periodic 4	240	0.8	240	192.0	192.0

Table 4. Event scores

the event and the score value. The data in Table 4 shows the results of analyzing the score table at the completion of the test. Each of the events listed is shown with the number of points associated with each execution. The event named Event Periodic 3 has two score values associated with each execution. The point values are associated with different score-labels defined in the instance file. The table shows that for each execution of an event in the Runs column, a score record was created in the Number of Records column. The Total Points column matches the expected points for each of the events. This result shows that the game server correctly records scores for each scheduled event execution.

Question 4: Do the final score results match the expected results?

The score results are calculated by the game server based on the definitions provided in the instance file. Five score results were defined in the instance file where each score used a different combination of math operators. In Table 5 the computations are calculated using the intermediate label calculations. The two labels used are shown with their summary value in the first two rows of Table 5. The labels are then shown being used in additional computations to provide a variety of scoring schemes. The multiplication score demonstrates how to use a weighted label computation in a score.

The game server correctly calculated all the values for the scores as shown

Score Name	Label Calculation	Expected Val	Actual Val
Red Team	redteam	-120.0	-120.0
Blue Team	blueteam	542.5	542.5
Addition	redteam + blueteam	422.5	422.5
Multiplication	redteam * 0.8 + blueteam * 0.5	175.25	175.25
Division	blueteam / redteam	-4.52	-4.52

Table 5. Final score calculations using intermediate score-labels

when comparing the Expected Values with the Actual Values columns. The score results depend on the computation provided for by the scenario designer. Invalid formulas generate an error message in the game server's log file produced for each instance run. The game server can use many types of mathematical operations in the computation of scores. The values show are a representative sample of all possible math operations.

Question 5: Do events that specify an IP address correctly use that address?

The game server's and target machine's network traffic was captured while executing the test instance file. An analysis of the traffic showed that each of the events that specified a particular IP address correctly used that address when executing network events. Each event in the instance file provided a unique IP address to be used when executing the event. The events were spaced apart by 30 seconds to aid in identification of the events within the traffic capture. The events were recorded on both sides of the connection, the game server side and the target machine side, as using the correct addresses.

The game server was successful in bi-directional communication with both the origin, the game server, and the target machine. This indicates that the events were both transmitted with the specified source IP address and that the game server correctly listened for network traffic with that address. The network traffic captured from the target machine showed the address resolution protocol (ARP) handshake that occurred before the test ping is sent. The traffic showed that for each execution of an event an ARP handshake occurred with a new media access control (MAC) address from the origin. The creation of unique MAC addresses was an unexpected result of using namespaces to encapsulate IP addresses.

Question 6: Are events that specify a randomized IP address executed with different addresses?

The results of testing to answer question 6 were the same for question 5. Each event was successfully assigned a random IP address from the pool that it specified. A comparison of the traffic from both the game server and the target machine showed the same results. Each message transmitted by the game server was replied to by the target machine and received by the game server.

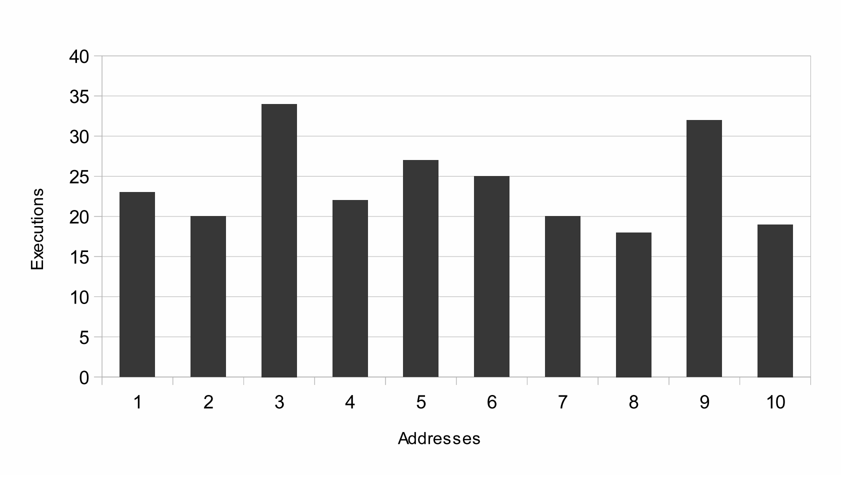


Figure 6. Distribution of 10 IP addresses for 240 events

The volume of traffic generated when testing periodic events was significantly higher than that created when testing single events. An analysis for the traffic generated was analyzed to confirm the results. Figure 6 shows the allocation of 10 IP addresses as randomly assigned to 240 event executions. The expected result would show a uniform distribution of addresses, but in this case the graph does not show that distribution. This is the result of a limited sample size. When the number of event executions increase the graph shows that the resultant allocations forms a uniform distribution. Figure 7 shows 10 addresses distributed over 14,400 events. This represents running an event every 0.5 seconds for 2 hours. The increased number of event executions shows that the game server does distribute the IP address to the events using a near uniform distribution. Any variation in the distribution can be attributed to the underlying pseudo-random number generator provided by the host system.

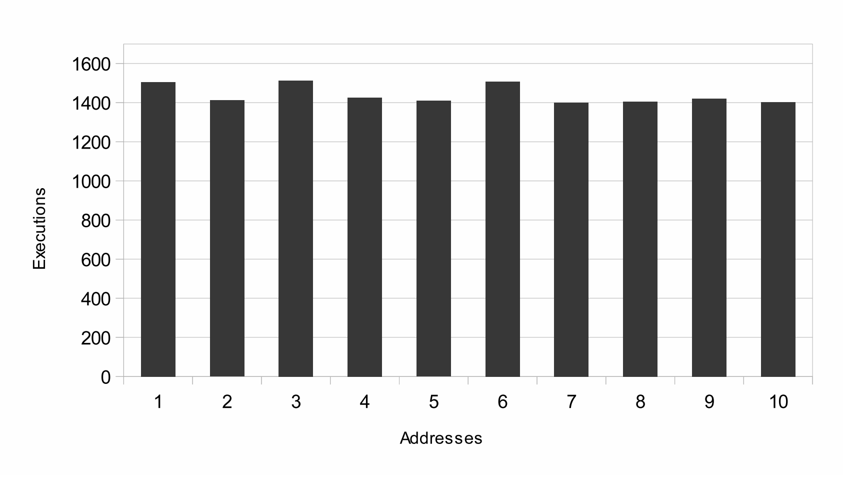


Figure 7. Distribution of 10 IP addresses for 14400 events

Question 7: Does event timing randomization adequately mask the signature of a periodic event?

This test measures the game server's ability to achieve the goal of this project to provide scenario designers with tools to create realistic network traffic. Events that occur at regular intervals can easily be detected through a heuristic analysis of network traffic. To determine if we can mask an event scheduled periodically we first look at the traffic generated when no masking is applied. Figure 8 shows the network activity of two events that are scheduled periodically by the game server. The difference between event arrivals is shown in the Y-axis and the length of the sample is shown in the X-axis. The event periods can be easily seen with visual inspection to occur at 3 second and 0.5 second intervals. The event with

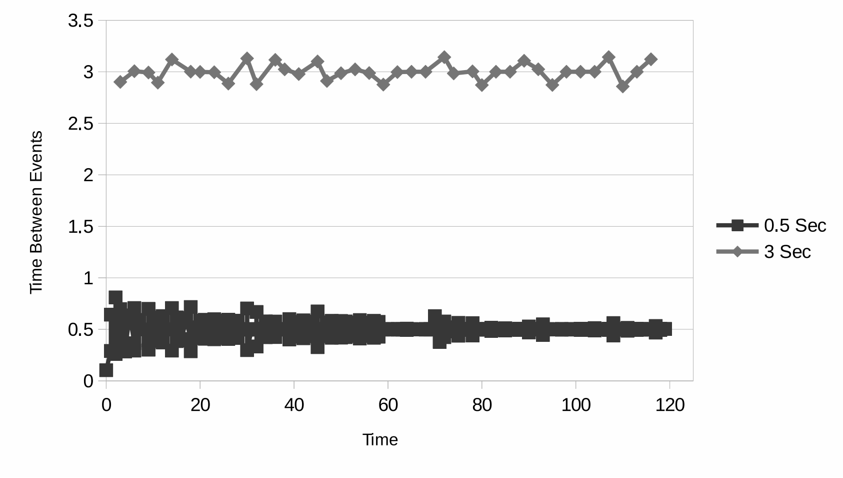


Figure 8. Periodic Events without Drift

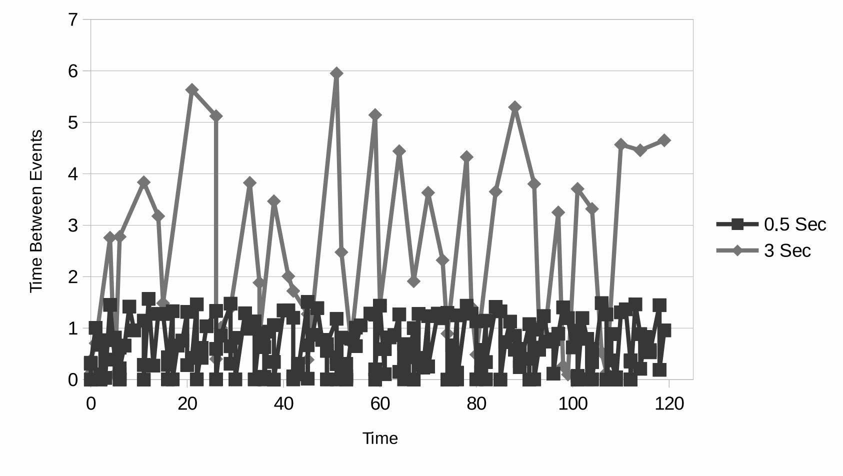


Figure 9. Periodic Events with Drift

a 0.5 second interval is not as clearly defined due to small latencies incurred by network communication. The amount of variation of results for the first 20 seconds can be attributed to the warmup period of the game server. The initial creation of namespaces takes approximately one-tenth of a second, but may be as long as three-tenths of a second.

We can now apply an amount of drift to each event execution to create a mask for the period. As shown in Figure 9 the period of the events is not immediately distinguishable. In this test we applied a drift of 3 seconds and 1 second to the 3

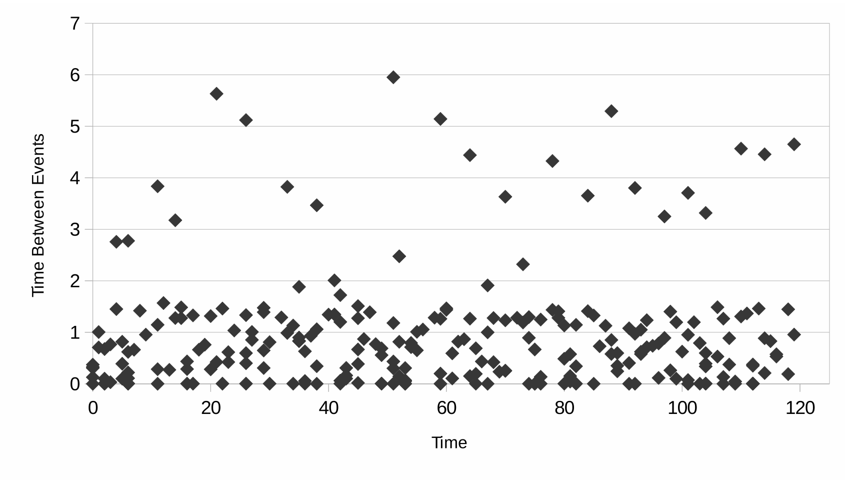


Figure 10. Unspecified Periodic Events with Drift

second event and the 0.5 second event respectively. The timings for the 0.5 second event are tightly clustered within the bounds of the period plus the drift in the range of 0 to 1.5 seconds, while the 3 second event appears to have a non-periodic schedule. The time between event executions for the 3 second event can also be short enough to make it indistiguishable from 0.5 second event executions. When we graph the same data without distinguishing between the events the impact of the timing randomization can be seen more easily. Figure 10 shows the events using the same mark and without a connecting line. This figure shows a more accurate picture of the apparent execution of events once a drift is applied.

After comparing the timings of events both with and without a suitable drift we can see that the game server can mask the periodic signature of an event. This requires the use of a drift value that is appropriate for the frequency of the event that is to be masked. The timings of the events shows that lower frequency events are more easily disguised than high frequency ones.

Question 8: Can the game server execute the types of events that represent typical network traffic of a cyber challenge?

The traffic that is generated by the game server is dependent on the events specified by the instance file. The current implementation of the game server supports one type of event natively, the Exec Event. An event of this type can specify a shell command to be run by the game server. When the event is run by the game server the command is passed to the system shell and executed. The return code of the event is used to determine whether the event is treated as succeeding or failing. Any application installed alongside the game server application can be run in this manner. Using this mechanism the game server is able to generate any network traffic required in a cyber challenge.

While testing the game server's ability to assign IP addresses we used the ping command to send ping messages to a target machine. This was accomplished by specifying the events in the instance file using the Exec Event type. The command was passed to the shell when executed and the return code provided to the game server upon completion. The game server then recorded a success or failure of the event and the appropriate score result.

The type of network traffic that can be generated by the game server is only constrained to the availability of external applications that can generate the desired type of traffic. Not all applications terminate with a return code that is suitable for the game server. To test these programs several bash-compatible script files were created. The script files were used to test the execution of other applications in addition to the ping command described above. These wrapper scripts handle any special input or output generated by the application and end with a return code that is appropriate for that execution.

The game server was successful in running the wrapper scripts to generate network traffic specified in the instance file. Table 6 shows a listing of the types of events that are generally used during a cyber challenge. Each event was successfully tested with the game server to record a result.

Program	Activity	Result
wget	Retrieve a web page	Successful
scp	Send a file to a server	Successful
nmap	Scan a network for hosts	Successful
sendmail	Send an e-mail message	Successful
curl	Login to web portal	Successful

Table 6. Event executed using the game server

Question 9: Do the interfaces of the game server provide the required functionality for each type of user?

The interfaces provided by the game server have unique target audiences as described in Section 3.1. Here we describe the relationship between users, their activities and the interfaces that support those activities. We examine each interface to describe the functionality provided and discuss the intended users of that interface as shown in Figure 11. Finally, we examine whether the functionality of that interface is appropriate for those users.

The instance file provides the primary interface with which to configure the game server. All users of the system will require at least the instance file to be able to use the game server to run a cyber challenge. The end users of the system will not be required to implement the instance file, but only need to provide a prepared instance file when running a challenge. The prepared instance file will contain all the necessary information to configure the game server. Therefore, for the end user, the instance file interface to the game server has met their needs of configuring the game server for a cyber challenge. Scenario designers will be the primary group of users building instance files to define cyber challenges. The scenario designer is responsible for choosing the configuration directives to include within the file that are apropriate for the cyber challenge they are defining. The instance file interface provides the complete set of configuration directives and options to configure the game server for the challenge they are trying to specify.

For the scenario designer the instance file interface provides all the functionality required by the scenario designer to configure every aspect of the game server. The developer will extend the instance file interface to meet new functionality added to the game server. The instance file uses a standard XML syntax to describe a cyber challenge. It is the responsibility of the developer to continue to implement new features and configuration directives in the instance file using this well defined interface. The instance file interface meets the needs of the developer by providing a standardized way to add new configuration directives that will be available for scenario designers.

The CLI interface is the simplest interface provided by the game server. The end user is presented with 5 options that control the game server and retrieve information about the state of a running cyber challenge. The controls provided with this interface meet the needs of all users by providing simple access to the basic functions needed while running a cyber challenge. Users may control the running state of the challenge by starting, pausing, resuming, or stopping the challenge. Users are also able to access the state of the game clock and the final scores computed by the game server. These activities encompass all the functionality required by users to operate a cyber challenge.

The web service API extends the game server to allow for additional development of user and application interfaces. The web service allows developers to create additional end user interfaces by providing access to information within the game server. With this interface it is possible to connect external applications to both the raw and computed data within the game server. The end users of the game server would not interact with the web service directly, but would use its functionality through secondary applications built on top of the web service's data feed. The web service allows developers to control and update a running cyber

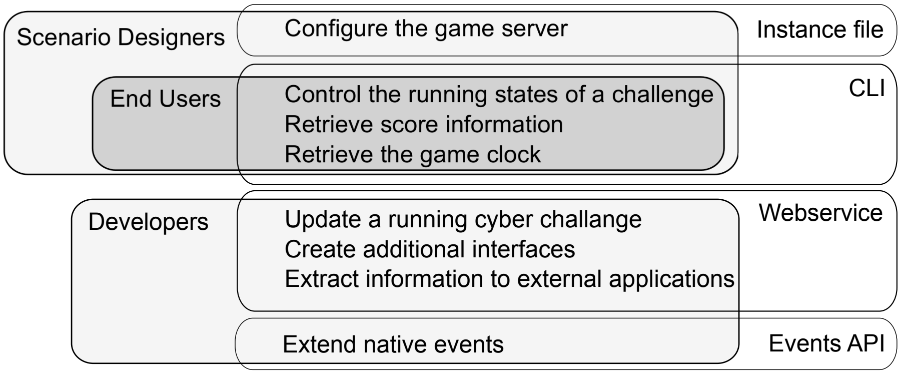

Scenario Designers
Configure the game server
Instance file
End Users
Control the running states of a challenge
Retrieve score information
Retrieve the game clock
CLI
Developers
Update a running cyber challange
Create additional interfaces
Extract information to external applications
Webservice
Extend native events
Events API

Figure 11. User and Interface functionalities

challenge by providing a well defined mechanism to allow configuration of some aspects of the game server.

The game server's final interface provides an API for developers to extend the functionality of the game server. The Events API allows developers to program new native events to be handled by the game server. This API allows the developer to specify the type of event and the method in which the event is run. The developer specifies the XML definition to be included in the instance file and provides a parser to instantiate the event object. The developer then specifies the run function for the event. When the game server executes the event it will call this run function. Within the function the developer has complete control over the running event. They may specify when and how to send scoring or logging events to the game server and they may specify any conditions that must be met in order to complete the event. This API allows the game server events to be extended without any restrictions on the developer. The functionality provided by this interface then meets the needs of a developer wishing to extend the game server with additional event functionality.

CHAPTER 5

Discussion

In the previous chapter we discussed the results obtained from testing the game server application. The testing we conducted sought to answer questions derived from the goals of this project about the game server and its capabilities. We will now evaluate the results of that testing to show that we have met our goals.

Our first goal is to have a game server that is capable of providing a central point of control for a cyber challenge. The results gathered from answering question 1 indicate that our implementation of the game server can perform that function. It is capable of setting up a challenge environment and controlling the run-time state of a scenario. The game server successfully enables moderators to control a cyber challenge to match their preferences. This capability may be used to enhance the educational value associated with participating in a cyber challenge.

The moderator of a cyber challenge is able to retrieve event and scoring information from the game server. In answering questions 2, 3, and 4 we established that the game server is capable of executing events, at their scheduled times, recording the scores for those events and computing final scores for the end users. These activities establish our goal number 2 of implementing a game server that provides usable functionality in scheduling and scoring events for a challenge.

The creation and use of cyber challenges is meant to enhance the end user's skills and abilities for use in real world situations. We have shown in answering questions 5, 6, and 7 that the game server provides the scenario designer useful tools in creating scenarios that have realistic qualities, achieving goal 3. Actual activities that may be performed by the game server are dependent on the creativity of the scenario designer. The game server provides a mechanism to support the types of traffic commonly found in cyber challenges. This makes it possible through the use of randomized addresses and timings to generate realistic network activity. The game server's ability to perform this function eases the burden placed on scenario designers when creating new scenarios.

Our goals for this project include making the game server able to be used successfully by the appropriate users while still remaining flexible enough to be extended with additional development. We have shown that the requirements for each type of user have been addressed by the game server when we answered question 9. This implementation provides all of the basic needs for all user skill levels to be able to interact with the game server in an appropriate manner, which satisfies goal 4. The interfaces of the game server provide a means for future developers to extend the functionality of the game server for scenario designers and create additional user interfaces for end users of the system, completing goals 5 and 6.

The results obtained from testing the game server show that we have successfully met the goals of this project. The game server is capable of performing all the required tasks needed to run a cyber challenge. We have shown that the game server is configurable and can control the running state of a challenge, as well as execute events specified by scenario designers. The outcome of this project was successful. In the following sections we discuss interesting results discovered while testing the game server application and future work for the continued development of the game server.

5.1 Network traffic

Namespaces are used for each event execution to create network traffic with a specified IP address. Each time an event with a random address assignment is executed it selects a new IP address to use for the execution. The game server then creates a namespace and configures it with the appropriate address. The creation and breakdown of a namespace takes a specific non-trivial amount of time to create depending on the resources available in the game server. A high-frequency event that uses some set of IP addresses will require the setup and breakdown of many namespaces for its event executions. This can lead to considerable load on the game server. In addition to the previously discussed constraint on the number of namespaces able to be created, the introduction of a namespace lifetime counter reduces the amount of work done to create and breakdown namespaces. Each namespace will stay alive with a specific IP address for the length of that lifetime. Any time an event re-uses that IP address and therefore namespace, the lifetime counter is reset. A repeating event whose frequency is less than the namespace lifetime will keep that namespace alive for the duration of the event. This significantly reduces the amount of resources consumed by the game server when using namespaces. The lifetime of the namespace is able to be configured globally for all namespaces. The actual lifetime depends on the number of high-frequency events that a challenge uses. During testing, a lifetime of thirty seconds produced a desirable balance for the types of traffic loads used in testing the game server.

When testing the IP address assignment to events by the game server we encountered a unique behavior that was not expected. We used namespace containers to create an execution environment for each event. The creation of a namespace also causes that namespace to have a unique MAC address assigned to the namespace as it is configured with an IP address. Normally, a host is connected to a network using a unique address in the OSI layer-2 model. This MAC address does not normally change and may be used to identify a host regardless of its IP address.

This is a desirable behavior for the game server. The creation of additional MAC addresses has the benefit that the traffic generated by the game server not only uses various IP addresses but also appears to be coming from different hosts. In a cyber challenge a player may decide to use MAC address filtering to block certain traffic from a host regardless of the IP address of that host. With this behavior the game server is able to act like many hosts there by increasing the realism of the traffic generated.

The ability of the game server to dynamically assign events new network addresses from the instance file provides critical functionality in using a single machine to generate network traffic in a cyber challenge. When the game server is deployed into a cyber challenge the scenario designer may specify which network segments the game server will be attached to. The designer can connect the game server to both the external interface of a network and the internal network. In this configuration the game server has the ability to generate traffic that imitates users of the network as well as attackers. Figure 12 shows how the game server can be connected to several different network segments. The designer can then create network traffic that is not affected by actions of the players. For example, a misconfiguration of the firewall may block access to particular parts of the network. Because the game server is directly connected to each network segment it can continue to function without interruption.

5.2 Event Timing

Event timing randomization is implemented using a drift when executing events. Randomization can be used to either disguise the fact that an event is occurring on a regular schedule or in the case of singular events to make the challenge re-usable by players without them being able to anticipate specific events. The amount of drift assigned to an event can be any value that the scenario de-

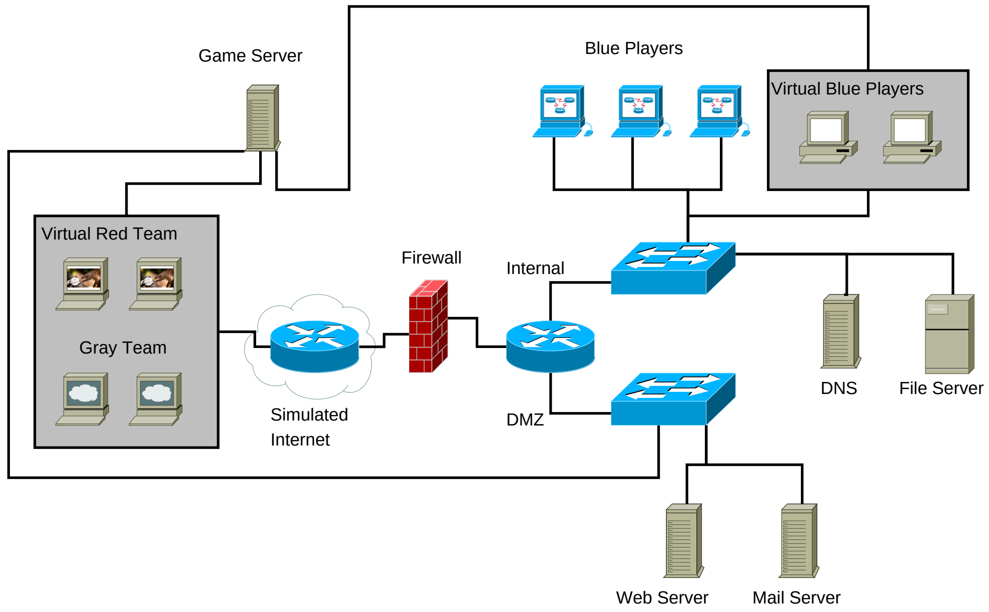


Figure 12. Sample OCCP scenario network

signer chooses. In the case of periodic events some interesting timing results can occur by setting the amount of drift to be significantly higher than the period of the event. In the current implementation of the game server the next time the event will be executed is computed at the last time that event is launched. When a drift value is computed that results in a negative amount of time until next execution the event is executed immediately. If the drift value is large it can cause the event to delay execution for an amount of time that is longer than its period. In this case the specified event will skip those periodic executions. This means that a periodic event does not guarantee a particular number of executions for a specified time period. The drift may cause more or fewer events to be executed for a given time period. The effect of an event using a drift larger than the period will result in a distribution that is mostly uniform with an exception at the 0 period. This is the result of events that are calculated with a negative time to launch being executed immediately.

5.3 Future Work

We have thoroughly tested each component of the software that comprises the game server solution that we have set out to build. The game server is meant to be used for a cyber challenge run on the OCCP. A complete test of the game server would ideally involve a full-scale cyber challenge complete with human players and moderators. This full scale testing would expand upon the time limited trials we have conducted. The game server, and the OCCP project as a whole, is in the early stages of recruiting scenario designers to build cyber challenges for the platform. We will seek to test the game server's capabilities at a future date when a mature cyber challenge becomes available. Given the current results of all tests performed we are confident that a full scale test will be successful.

The use of namespaces for light weight virtualization can be extended into other areas of the kernel. Projects like Docker [25] provide complete application containers. These containers share the system's running kernel while virtualizing all other aspects of the system. These Docker containers have the ability to have their own file systems, network namespaces, and process stacks. Support for the control and use of Docker containers would be a complementary add-on for the OCCP game server. Integration of Docker within the current framework of the game server would allow rapid extension of additional programs to schedule as events. This would require several modifications to the Docker mechanisms. The most important of these modifications would require more enhanced networking control of the Docker containers. The game server makes extensive use of kernel namespaces to provide address randomization. The Docker containers would need to be modified so that individual containers could be configured to use the appropriate addresses. This change would provide the added benefits of Docker containers while allowing the game server to provide the same level of address customization it now supports.

The web service API provided by the game server is intended to support the creation of additional front-end (FE) applications. A front end application would provide a more user-friendly web based user interface to the game server. A challenge moderator could use this application to preview upcoming event executions for each of the teams in a scenario. The application would also support more fine grained control over individual Team states. This would allow the moderator to pause a single team or group of teams. This is in contrast to the CLI that currently pauses all Team activity. The web service provided allows the front end application to have fine grained control over the events executed by each Team. It would be possible to create additional events to be added into a scenario Team on the fly or to cancel specific events. The web service API encapsulates the implementation of the game server so that development of the front end application could proceed in parallel with the continued development of the game server.

In addition to the moderator interface, the FE application could be used to display a graphical scoreboard. This spectator interface could show interesting information about the state of the cyber challenge. The information presented could include scores, upcoming events, and the status of past event successes or failures. This enhancement may encourage participation in cyber challenge exercises by increasing their visual appeal.

The creation of a front end application would also permit the creation of additional mechanisms with which to score a cyber challenge. A FE application would be capable of supporting a player interface. The interface for the players could allow for the submission of flags. A *flag* in the context of a cyber challenge is a piece of information that the player has discovered within the scenario. The game server's web service has the ability to receive additional score-events to be recorded in the score database. A FE application could present the players with a mechanism by which they submit those flags and earn points during the challenge.

The development of the game server is expected to continue in the OCCP project. The OCCP is a community driven project that provides open access to the source code and implementation of all its components. The strategies, tests, results, and lessons learned presented here will contribute to the overall success of the game server application as a foundation piece of the Open Cyber Challenge Platform.

LIST OF REFERENCES

[1] P. A. Buxbaum. The International Relations And Security Network. “Building a better ’cyber range’.” March 2012. [Online]. Available: http://www.isn.ethz.ch/Digital-Library/Articles/Detail/?id=127714

[2] G. Vigna, “Teaching network security through live exercises,” in *Security education and critical infrastructures.* Springer, 2003, pp. 3–18.

[3] “Team - Official TF2 Wiki — Official Team Fortress Wiki.” April 2013. [Online]. Available: https://wiki.teamfortress.com/wiki/Teams

[4] Air Force Association. “AFA CyberPatriot website.” [Online]. Available: https://www.uscyberpatriot.org/

[5] “US Cyber Challenge - competitions.” April 2014. [Online]. Available: http://www.uscyberchallenge.org/competitions/

[6] “SANS institute - netwars.” April 2014. [Online]. Available: https://www.sans.org/netwars

[7] P. Vixie, “Cron manual page, 4th berkeley distribution,” *The information from the crontab section (below and including the table) was taken (unedited, but with small additions) from the crontab manual pages. Type man*, vol. 1.

[8] *Quartz Scheduler 2.1.x Documentation*, Quartz, April 2014. [Online]. Available: http://quartz-scheduler.org/files/documentation/Quartz-2.1.x-Documentation.pdf

[9] Ixia. “Ixia BreakingPoint Storm.” April 2014. [Online]. Available: http://www.ixiacom.com/products/storm

[10] University of Rhode Island, Department of Computer Science and Statistics. “Open Cyber Challenge Platform.” November 2014. [Online]. Available: http://www.opencyberchallenge.net

[11] “About Nagios.” [Online]. Available: http://www.nagios.org/about

[12] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts.* Hoboken, NJ: J. Wiley & Sons, 2005.

[13] *Administrator’s Solutions Guide for Release 6*, Oracle, May 2014. [Online]. Available: http://docs.oracle.com/cd/E37670_01/E37355/E37355.pdf

[14] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, “Unraveling the web services web: an introduction to soap, wsdl, and uddi,” *IEEE Internet computing*, vol. 6, no. 2, pp. 86–93, 2002.

[15] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," *ACM Transactions on Internet Technology*, vol. 2, no. 2, pp. 115–150, May 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=514185

[16] S. Furuhashi. "Messagepack: Its like json. but fast and small." February 2014. [Online]. Available: http://msgpack.org/

[17] J. Highsmith, *Agile Software Development Ecosystems.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[18] J. Martin, *Rapid Application Development.* Indianapolis, IN, USA: Macmillan Publishing Co., Inc., 1991.

[19] A. Harris and K. Haase, *Sinatra: Up and Running.* O'Reilly Media, Inc., 2011.

[20] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Publishing Company, 1994.

[21] I. Recommendation, "200 (1994)— iso/iec 7498-1: 1994," *Information technology–Open Systems Interconnection–Basic Reference Model: The basic model.*

[22] W. Mauerer, *Professional Linux Kernel Architecture.* John Wiley & Sons, 2010.

[23] V. Jacobson, C. Leres, and S. McCanne, "The tcpdump manual page," *Lawrence Berkeley Laboratory, Berkeley, CA*, 1989.

[24] *ping(8), Linux Manual.*

[25] Docker, Inc. "Docker - Build, Ship, and Run Any App, Anywhere." November 2014. [Online]. Available: https://www.docker.com/

APPENDIX A

Game Server Instance File

Listing A.1. Example Instance File

```
<?xml version="1.0" encoding="UTF-8"?>
<occpchallenge>
  <!-- Describe the current scenario -->
  <scenario gameid="1"
    name="Red Team vs. Blue Team"
    type="Network Defense"
    description="Blue team defends the network against automated Red team attacks. This scenario focuses on protecting against weak passwords and unused services.">

  <!-- Length of time to run this scenario
     format {seconds, minutes, hours}   -->
  <length time="15" format="seconds" />

 <!-- Users for the web services console -->
 <users>
  <user name="moderator" pass="token"/>
  <user name="blueplayer" pass="token"/>
  <user name="spectator" pass="token"/>
 </users>

<!-- Score groups labels allow certain events to be grouped together for score calculation. Labels do not require any special string. SQL expressions must return a single row and value. If more than one row is returned only the first row is used -->
  <score-labels>
```

```
    <!-- Example Default query if none provided -->
    <score-label name="group1" sql="SELECT SUM(value) FROM SCORE WHERE groupname='group1'"/>
      <!-- Example Additional WHERE clause parameters create a 5 minute sliding window -->
      <score-label name="webserverstatus" sql="SELECT SUM(value) FROM SCORE WHERE groupname='webserverstatus' AND time > (time('now') - 300)"/>
      <!-- Example Default sum query is used -->
      <score-label name="redteam" />
      <score-label name="blueteam" />
      <!-- Example Query may use any columns in the score table-->
      <score-label name="teamtotal" sql='SELECT SUM(value) FROM SCORE WHERE groupname!="webserverstatus"'/>
  </score-labels>

<!-- Scores are calculated by making calculations from the groups of the labels above. Each score-name is treated as ERb template; use score-label as variables in calculation-->
<score-names>
 <score-name name="red-team" descr="Red Team" formula="redteam" />
 <score-name name="blue-team" descr="Blue Team" formula="blueteam" />
 <score-name name="addition" descr="Service Level" formula="redteam + blueteam" />
 <score-name name="multiplication" descr="Service Level" formula="redteam * 0.8 + blueteam * 0.5" />
 <score-name name="division" descr="Service Level" formula="blueteam / redteam" />
</score-names>

```

```
  <!-- Scoreboards show a particular grouping of scores to user accts with permission. This section is used by the web interfaces -->
  <scoreboards>
    <scoreboard name="moderator-board" update-rate="live">
      <score-name name="red-team" />
      <score-name name="blue-team" />
      <score-name name="service-level" />
    </scoreboard>
    <scoreboard name="spectator-board" update-rate="live">
      <score-name name="red-team" />
      <score-name name="blue-team" />
      <score-name name="service-level" />
    </scoreboard>
    <scoreboard name="player-board" update-rate="1min">
      <score-name name="red-team" />
      <score-name name="blue-team" />
      <score-name name="service-level" />
    </scoreboard>
  </scoreboards>

</scenario>

  <!-- Host describes the machine that is running the gameserver software. Each network port to be used must be named and listed to be used in events. -->
  <host label="gameserver" hostname="gameserver">
    <interface name="eth0" network="pub" />
    <interface name="eth1" network="prvt"  />
    <interface name="eth2" network="int" />
  </host>

  <!-- IP pools describe pools of ip addresses that an event can use as source addresses. These are useful for allocating to zombie traffic where the particular source address does not matter. Each pool will only contain unique addresses. If pool definition specifies overlapping addresses only one address is used in the pool. -->
<ip-pools>
  <!-- Simplest definition is a list of one address. Lists are comma separated IPv4 addresses of the format X.X.X.X with no CIDR. -->
  <pool name="int_2" network="prvt" cidr="24" gateway="">
    <address type="list">10.3.3.2</address>
  </pool>
  <!-- A range is defined by a  CIDR denoted address block count is the number of addresses to add to the pool -->
  <pool name="int_1" network="prvt" cidr="24" gateway="">
    <address type="range" count="12" addr="10.24.15.0/24" select="asc" />
    <address type="list">10.3.3.2, 128.15.23.4, 143.34.21.9</address>
    </pool>
  <pool name="pub_1" network="prvt" cidr="24" gateway="" >
    <address type="range" count="9" addr="16.0.0.0/8" select="asc" />
    <address type="range" count="23" addr="78.3.0.0/24" select="asc" />
    <address type="range" count="15" addr="200.15.14.0/24" select="asc" />
  </pool>
  <pool name="prvt_1" network="prvt" cidr="24" gateway="">
    <address type="range" count="11" addr="87.15.0.0/20" select="asc" />
  </pool>
</ip-pools>

  <!-- NOT USED, RESERVED FOR FEDERATION.
       Identify each location that team processes will execute. Each of these locations needs to be contacted to dispatch the appropriate team code to. These hosts need to specified in the host section. -->
  <!-- <team-hosts>
      <team-host name="Local"
                   hostname="localhost"
                   ip-addr="127.0.0.1"
                   port="24365" />
      <team-host name="RemoteHost1"
                   hostname="RemoteHostName"
                   ip-addr="0.0.0.0"
                   port="24365" />
  </team-hosts>
 -->

  <!-- Identify the handlers needed to run events
      name - is locally referenced in the instance file only
      class-handler - must specifiy the Class name of the handler
  -->
  <event-handlers>
    <handler name="exec-handler-1"
        class-handler="ExecHandler" />
    <handler name="metasploit-handler-1"
                  class-handler="MetasploitHandler"
                  server-hostname="host1"
                  server-ip=""
                  server-port="" />
  </event-handlers>

  <!-- Team describes a collection of events for organization
     name - the friendly name of the team -->
<team name="Blue Team">

  <!-- Identifies the name of the execution host -->
  <team-host hostname="localhost" />

  <!-- Rate at which events are executed from the event list -->
  <speed factor="1.0" />

  <!-- Block for events in this teams event list -->
  <team-event-list>
    <team-event name="Ping Test" id="" guid="" handler="exec-handler-1"  ipaddress="pub_1" starttime="1" endtime="10" frequency=".3" drift="0" command="ping -c 2 10.24.32.10">
      <score-atomic when="success" score-group="group1" points="5" />
      <score-atomic when="fail" score-group="group2" points="-3" />
    </team-event>
    <team-event name="ping" id="" guid="" handler="exec-handler-1" ipaddress="prvt_1" starttime="5" endtime="9999999" frequency="2" drift="0" command="ping -c 1 10.24.32.10">
      <score-atomic when="success" score-group="group1" points="5" />
      <score-atomic when="fail" score-group="group2" points="-3" />
    </team-event>
    <team-event name="Which" id="" guid="" handler="exec-handler-1" ipaddress="int_1" starttime="8" endtime="9999999"
```

```
          frequency="0" drift="0" command="ping -c 2 10.24.32.10">
      <score-atomic when="success" score-group="group1" points="25" />
      <score-atomic when="fail" score-group="group2" points="-1" />
    </team-event>
  </team-event-list>

</team>

<team name="Red Team">

  <!-- Identifies the name of the execution host -->
  <team-host hostname="localhost" />

  <!-- Rate at which events are executed from the event list -->
  <speed factor="1.0" />

  <!-- Block for events in this teams event list -->
  <team-event-list>
    <team-event name="Ping Test" id="" guid="" handler="exec-handler-1" ipaddress="pub_1" starttime="10" endtime="9999999" frequency="0" drift="0" command="ping -c 4 127.0.0.1">
      <score-atomic when="success" score-group="group1" points="5" />
      <score-atomic when="fail" score-group="group2" points="-3" />
    </team-event>
    <team-event name="Arping" id="" guid="" handler="exec-handler-1" ipaddress="int_1" starttime="5" endtime="9999999" frequency="0" drift="0" command="arping -c 1 10.24.32.10">
```

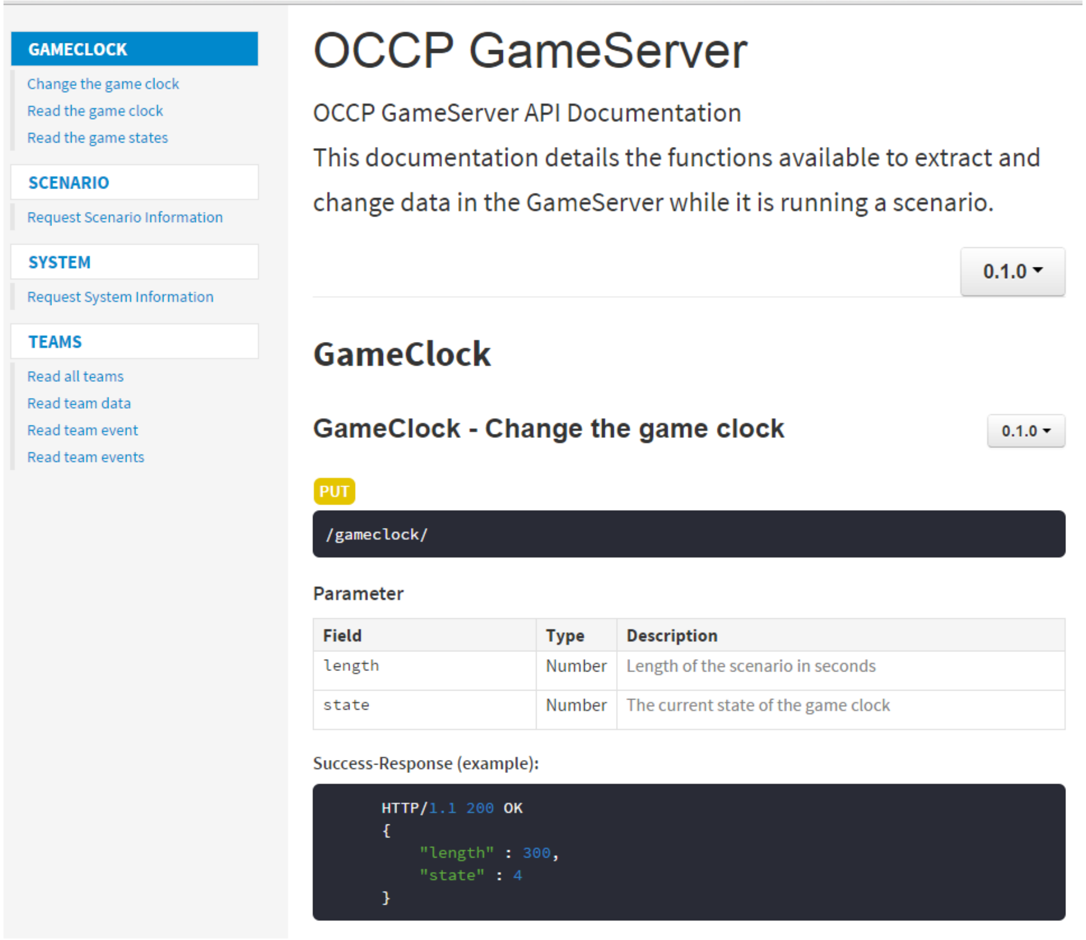
```
      <score-atomic when="success" score-group="group1" points="5" />
      <score-atomic when="fail" score-group="group2" points="-3" />
    </team-event>
    <team-event name="Which" id="" guid="" handler="exec-handler-1" ipaddress="int_1" starttime="0" endtime="9999999" frequency="0" drift="0" command="which ping">
      <score-atomic when="success" score-group="group1" points="25" />
      <score-atomic when="fail" score-group="group2" points="-1" />
      </team-event>
    </team-event-list>

  </team>

</occpchallenge>
```

APPENDIX B

Game Server Web Service API



OCCP GameServer

OCCP GameServer API Documentation

This documentation details the functions available to extract and change data in the GameServer while it is running a scenario.

GameClock

GameClock - Change the game clock

```
/gameclock/
```

Parameter

Field	Type	Description
length	Number	Length of the scenario in seconds
state	Number	The current state of the game clock

Success-Response (example):

```
HTTP/1.1 200 OK
{
    "length" : 300,
    "state" : 4
}
```

Figure B.13. Game server documentation website

The complete game server web service API is available at the following URL.

- `http://students.cs.uri.edu/~jafonseca/gameserver/api/`

BIBLIOGRAPHY

"About Nagios." [Online]. Available: http://www.nagios.org/about

ping(8), Linux Manual.

"Team - Official TF2 Wiki — Official Team Fortress Wiki." April 2013. [Online]. Available: https://wiki.teamfortress.com/wiki/Teams

"SANS institute - netwars." April 2014. [Online]. Available: https://www.sans.org/netwars

"US Cyber Challenge - competitions." April 2014. [Online]. Available: http://www.uscyberchallenge.org/competitions/

Air Force Association. "AFA CyberPatriot website." [Online]. Available: https://www.uscyberpatriot.org/

Buxbaum, P. A. The International Relations And Security Network. "Building a better 'cyber range'." March 2012. [Online]. Available: http://www.isn.ethz.ch/Digital-Library/Articles/Detail/?id=127714

Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., and Weerawarana, S., "Unraveling the web services web: an introduction to soap, wsdl, and uddi," *IEEE Internet computing*, vol. 6, no. 2, pp. 86–93, 2002.

Docker, Inc. "Docker - Build, Ship, and Run Any App, Anywhere." November 2014. [Online]. Available: https://www.docker.com/

Fielding, R. T. and Taylor, R. N., "Principled design of the modern web architecture," *ACM Transactions on Internet Technology*, vol. 2, no. 2, pp. 115–150, May 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=514185

Furuhashi, S. "Messagepack: Its like json. but fast and small." February 2014. [Online]. Available: http://msgpack.org/

Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Publishing Company, 1994.

Harris, A. and Haase, K., *Sinatra: Up and Running.* O'Reilly Media, Inc., 2011.

Highsmith, J., *Agile Software Development Ecosystems.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

Ixia. "Ixia BreakingPoint Storm." April 2014. [Online]. Available: http://www.ixiacom.com/products/storm

Jacobson, V., Leres, C., and McCanne, S., "The tcpdump manual page," *Lawrence Berkeley Laboratory, Berkeley, CA*, 1989.

Martin, J., *Rapid Application Development.* Indianapolis, IN, USA: Macmillan Publishing Co., Inc., 1991.

Mauerer, W., *Professional Linux Kernel Architecture.* John Wiley & Sons, 2010.

Administrator's Solutions Guide for Release 6, Oracle, May 2014. [Online]. Available: http://docs.oracle.com/cd/E37670_01/E37355/E37355.pdf

Quartz Scheduler 2.1.x Documentation, Quartz, April 2014. [Online]. Available: http://quartz-scheduler.org/files/documentation/Quartz-2.1.x-Documentation.pdf

Recommendation, I., "200 (1994)— iso/iec 7498-1: 1994," *Information technology–Open Systems Interconnection–Basic Reference Model: The basic model.*

Silberschatz, A., Galvin, P. B., and Gagne, G., *Operating System Concepts.* Hoboken, NJ: J. Wiley & Sons, 2005.

University of Rhode Island, Department of Computer Science and Statistics. "Open Cyber Challenge Platform." November 2014. [Online]. Available: http://www.opencyberchallenge.net

Vigna, G., "Teaching network security through live exercises," in *Security education and critical infrastructures.* Springer, 2003, pp. 3–18.

Vixie, P., "Cron manual page, 4th berkeley distribution," *The information from the crontab section (below and including the table) was taken (unedited, but with small additions) from the crontab manual pages. Type man*, vol. 1.