2012

The CDC in a box project

Jason Aaron Cross
Iowa State University

The CDC in a box project

by

Jason Aaron Cross

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Co-majors: Computer Engineering;

Information Assurance

Program of Study Committee:

Doug Jacobson, Major Professor

Thomas Daniels

Clifford Bergman

Iowa State University

Ames, Iowa

2012

DEDICATION

This thesis is dedicated to my wife, Kara, and to my parents, Drs. Betty and Timothy Cross. Their encouragement and support were invaluable during my graduate work.

TABLE OF CONTENTS

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with this thesis. First, I'd like to thank Dr. Doug Jacobson for his direction and for funding my research. Participating in the CDCs that he started was a great experience and an interesting research topic. Second, I'd like to thank the ISEAGE staff, especially Matthew Sullivan and Max Peterson. Matt provided great feedback on my ideas, as well as helping me to understand how the CDCs work. Max entrusted me with many of the details of the scenario for the CDC competition he was in charge of. By experimenting during his CDC, I created extra work for him and introduced uncertainties which are never pleasant for a CDC director. The insight I gained from the experience was valuable as I was able to try out many of my ideas. In addition, Max is utilizing his skill as a web developer to develop IScorE. It is becoming a robust web application that will greatly enhance CDCs to come. It will be a great part of the CDC in a box project.

ABSTRACT

ISU held the first Cyber Defense Competition (CDC) in the spring of 2005. Since then, the number of competitions per year has grown, as well as the complexity of the competitions. As cyber security becomes increasingly important to both the public and private sectors, other higher learning institutions may wish to run their own CDCs. In order to support such efforts, we are developing the CDC in a box. This project would provide software, tools, and support to help other institutions run CDCs. As part of this effort, I identify short comings of the current ISU CDCs and provide solutions along with results. In addition, the core tools and software needed to create the CDC in a box are introduced, as well as the motivation behind their creation.

CHAPTER 1. OVERVIEW

As cybersecurity becomes a growing concern among both the private and public sectors, the need for skilled computer security professionals has increased. The types of cybersecurity jobs fall into several areas of focus. Two of the most common are: 1. Traditional defense in which the professional protects systems and data from attack and misuse. 2. Penetration testing where the professional attempts to defeat the system's defenses. These jobs require skills that allow the professional to understand how cyber attackers, or hackers, work. In other words, they themselves need to know how to hack.

At Iowa State University (ISU), several cyber defense competitions (CDCs) are held each year. These competitions allow students to set up computer systems and then defend them against a group of hackers. These hackers are made up of professionals and more experienced students. By participating as either a defender or an attacker, a student gains hands on experience practicing cybersecurity.

There are four CDCs every year at ISU. The first is between teams of students at ISU. Both undergraduate and graduate students can participate. The next is for community colleges (C3DC) in the area. The third is the national competition (NCDC) where teams from other universities as well as the top performers at the first and second CDCs participate. The fourth is a part of the IT-Olympics event. These teams are made up of high school students. The purpose of the CDCs is to help students of every skill level learn more about the IT field, teamwork, and cybersecurity. As such, many of the students participate in the CDCs for fun without any formal classwork or training.

As cybersecurity becomes increasingly common in higher learning curriculums, other higher learning institutions may want to run their own cyber defense competitions to give their students hands on practice. To support this, I have begun the development of the CDC in a box

project. This will be an ongoing project at ISU that will provide both tools and guidance to run CDCs.

1.1 Introduction

A successful CDC requires: people, logistics, hardware, and software. Each of these components is equally important.

1.1.1 People

There are several groups of people at a CDC broken up into various teams: blue, white, red, and green. There are also volunteers and spectators that aren't part of any team.

1.1.1.1 Blue Teams

Blue teams are made up of the defenders. The blue teams compete against each other. Scoring is determined to gauge their performance. At the end of the competition, the team with the highest score wins. For some competitions, each blue team has an adviser. This person assists the teams with logistics and his or her own knowledge, but is not allowed to help the team during the competition.

1.1.1.2 White Team

White team members run the competition. They are essentially both the hosts and the referees. A competition leader is chosen and he or she oversees the running of the competition while delegating to the rest of the white team. The white team determines the schedule of events and performs the scoring as well as acts as a judge for any disputes. The white team also awards points to the blue teams based on how good their network and systems documentation is. They also award points based on the blue teams' reports submitted every two hours detailing how their systems were attacked.

1.1.1.3 Red Team

Red team members act as hackers and attack the blue teams' systems. If they break into a blue team's system and steal data, that team loses points. The data they steal are known as flags. So, the goal of the red team is to play a game of capture the flag. They also attempt to plant flags on the blue teams' systems. This represents malicious changes that the attackers could make to the system with their administrative access. They also award points to the blue teams based on both their technical performance and their team spirit. A red team leader is chosen who directs the rest of the team.

1.1.1.4 Green Team

Green team members act as users of the blue teams' systems. They award points to the blue teams based on how well their services work. A green team leader is chosen who directs the rest of the team.

The green team leader assigns anomalies to the blue teams. These anomalies are used to simulate real world events and requests. For example, adding/deleting/disabling users, setting up new services, restoring files, or solving a problem.

The green team leader along with the green team score the user documentation provided to them by the blue team. The documentation is scored for accuracy, completeness, professionalism, and ease of use.

1.1.1.5 Other People

Other volunteers usually help run the competition without being an official part of it. For example, someone may staff and maintain a snack bar where participants can get refreshments. In addition, there are commonly spectators who drop by and are curious about the competition. It is common for them to walk around talking to the blue team members about their experience. At ISU the spectators are welcome, but depending on the rules of a competition, they may not be allowed to directly interact with the teams.

1.1.2 Logistics

A CDC requires substantial planning and preparation from all involved, especially the white team. A venue must be chosen that is large enough and can support all the necessary computer hardware. Dates and schedules need to be decided ahead of time. The way teams prepare their systems for the competition as well as to how to get those systems to the competition must be decided.

1.1.3 Software

For the competition systems, any operating system or software can be used as long as it is allowed by the competition rules. In general, ISU allows free, trial, and ISU provided software. The white team systems provide the core of the competition network through an ISEBox system. ISEBox is a single physical server running several virtual machines each running ISEFlow. ISEBox creates a virtual internet that is used for the competition network. Additional systems are used by the white team to run the competition, including: an e-mail server, a web server, a scoring server, several proxy servers, and a DNS server.

1.1.4 Hardware

The needs of each CDC may vary. However, in general computers are needed for everyone involved. These systems need not be powerful as they are simply for letting each member interact with the competition systems. It is common to require each person participating to bring a laptop.

The hardware necessary to run the competition systems is more substantial. Generally server class systems are needed for each blue team. In lieu of one powerful system, blue teams may use several less powerful systems. The white team also needs to have at least one server class system to run ISEBox as well as some support systems.

Networking hardware is also needed. Wireless is generally not used, so ethernet switches are needed to connect everyone to the competition network. In addition, this may require long network cable runs.

1.1.5 A typical ISU CDC

Several months before the competition is to begin, potential participants are asked to register blue teams for the event. About one month before the competition, the white team sends the rules, scenario, and set up information to all registered blue teams. Volunteers build several vulnerable systems to be given to each blue team. These systems may have both common administrative mistakes, such as running a service as an administrative user, or may contain backdoors or other malicious programs left by hackers after a system compromise. The blue teams then remotely access the systems they are given. In addition, they are allowed to create a few systems completely from scratch. Sometimes the choice of operating system and software is given to the blue team. Other times it is specified in the scenario.

Blue teams have free reign to set up their network topology as they please. Many choose to put a firewall in front of their systems. In addition, blue teams may set up extra systems and services not required by the scenario. However, each team is given limited resources which may make it challenging to implement additional systems and services. Commonly teams will build a Windows Active Directory Domain and so will set up a server as a Windows Active Directory Domain Controller.

The majority of the blue teams' effort goes into securing their systems and ensuring their services work as described in the scenario. If teams have additional time, they may set up an intrusion detection system on their network. The white team documentation and green team documentation are generally left to the end of the set up process to write. Teams that have difficulty securing and running their services commonly do not write this documentation, or it may be sparse and incomplete. The green team documentation is of great importance, as it gives instructions to the green team on how to use the blue team's services. Without this documentation, the green team may not be able to properly access the blue team's services resulting in lost points. Both sets of documentation are also scored.

The white team sets up the venue so that it is ready the day before the competition. Blue teams then arrive and have a day to finish preparing their systems on the competition network. At this time the blue teams connect directly to the competition network and are able to test

their services just like the green team will test them. During this time, volunteers are available to help blue teams set up their services. Blue teams are allowed to receive help from anyone before the competition, but once it starts, help is forbidden. Blue teams commonly need help with their firewall setup as many wait to set the firewall up until this time.

The competition starts at 8 AM. At 8 AM the red team begins to enumerate and attack the blue teams' systems. The automated service scanner is started, so teams gain points for every service that is available. The green team begins usability checks and may start assigning anomalies at will. At 8:30 the green team and white team documentation are due. Teams receive score penalties for turning in the documentation late. The white team documentation is graded by the white team when time is available. The green team documentation is graded at the end of the competition.

Throughout the competition the red team gains access to the blue teams' systems. Generally they are directed to not completely destroy a blue team system until the very end of the competition. Besides capturing and planting flags, red team members usually make their presence on the systems known to the blue teams. For example, they may deface the website's home page and shut down the web service. Thus, the blue teams are alerted to the fact that they have been compromised. They must then fix the vulnerability that the red team member used to break in, fix their website, and then bring the service back online.

Throughout the competition blue teams submit reports every two hours on the malicious activity they detect and mitigate on their systems. These reports are sent to, and scored by, the white team.

Blue team members must work together to keep up with all the tasks in the competition. Every two hours intrusion reports need to be submitted, services need to be kept up and available, green team usability reports need to be checked and possibly contested, any work not completed before the competition starts needs to be completed, and green team anomalies need to be handled as they are given out by the green team.

The competition usually runs until about 3 or 4 PM. At this time, most vulnerable systems have been compromised by the red team. After some cleanup, the red team debriefs the blue teams on the vulnerabilities they discovered and possible mitigation steps to prevent or fix

them.

In order to allow the participants to focus on the event, breakfast and lunch, as well as a snack bar, are provided. In addition, participants are given t-shirts as both keepsakes and to identify their team status. Blue shirts are given to the blue team, red ones to the red team, and green ones to the green team. Throughout the competition, blue team members are not supposed to talk to either red team or green team members without the permission of the white team or the green team leader. Each type of team has their own area. The blue teams are all grouped together. The red team is set apart from the blue teams and are visibly hidden from them. The green team is put wherever is convenient, usually right next to the blue teams. The white team is also set apart and has an area where the server systems can be secured. In addition, this area is set apart so that green, red, and white team members can have discussions without blue team members overhearing.

1.2 Areas needing enhancement in ISU CDCs

Before beginning the development of CDC in a box for use by others, it was necessary to review the existing ISU CDCs and determine where they needed to be enhanced. I identified several areas that needed attention: scoring, the green team, the competition network, and the competition scenarios.

1.2.1 Problems with scoring

The total team score is made up of several components: uptime checks, intrusion reports, usability scores, anomaly scores, flags remaining, system documentation, user documentation, and the red team evaluation. Since the goal of the competition is to protect the flags, or sensitive data, losing flags or having them planted results in large point loses. Commonly blue teams that lose flags early on would quit and leave in the middle of the competition. Basically, they gave up since they didn't think they could win.

Scoring was done manually using paper and spreadsheets. This made the scoring prone to human error and involved a lot of work from white team to compile. In addition, teams didn't know how they stood in the competition compared to other teams.

The green team usability scores were done using paper. Blue teams often didn't know they had been graded, and many never looked at their usability scores. This meant they lost points for services that were working, but for whatever reason the green team couldn't figure out how to access or use. This commonly occurred since green team members would neglect to read the documentation they were given by the blue teams.

The uptime scores were determined by an automated system, but it wasn't designed to collect uptime information at any specific time. White team members had to check the system to award points periodically.

The scoring system used was complex due to the use of a multiplier and didn't work well with how the green team functioned.

1.2.2 Problems with the green team

Historically the green team members were non-technical spectators that were interested in the competition[1]. In addition, those interested in helping with the competition that had either participated in a CDC before as a blue team member or were technically skilled generally wanted to join the red team. This meant the green team typically had a few technically skilled members with mostly regular computer users. These users could do simple tasks once shown, like log into a server and run an application. However, they commonly would neglect to read the documentation provided by the blue teams, and would not always understand it.

Usability scoring was also plagued by vague requirements, often resulting in green team members subjectively taking points away because they didn't like something about the system. For example, 5 points could be awarded if a system was up and working. They may not like that a certain feature was disabled, so they would take off 1 point. Another green team member would not necessarily take a point away for that.

Checking if e-mail was working properly was also difficult. Blue teams were supposed to be able to send mail between each others mail servers, but that was hard for green team to check, as they would have needed access to a fully working blue team mail server in order to check the other blue team mail servers.

Many of the usability checks performed by green team members could be performed by

automated scripts. For example, going to a website or logging into an FTP server.

Grading anomalies became difficult as well, as only the basic anomalies could be checked by most of the green team members. The rest had to be checked by the skilled members. This limited the number of anomalies that could be performed. In addition, it made it almost impossible to impose time restrictions on anomalies, as they could not be graded right away.

The documentation that the blue team gave the green team was printed. This meant that the documentation had to be constantly tracked. Also, updates to it had to be scribbled in by blue team members. Commonly this meant that blue teams lost points since they never updated vague points of their documentation.

1.2.3 Problems with the competition network

The competition network simulates the real internet. Instead of using private IP ranges that can be used by anyone connected to the internet via Network Address Translation (NAT), the competition network uses public IP addresses that are not allocated to ISU and are used by other organizations on the internet. Thus the competition network is separated from the internet. A pair of HTTP proxy servers, known as they keyhole, is used to allow systems on the competition network to access the real internet. These proxy servers connect to each other via a shared class C private subnet. This is needed to download software or update software. Commonly blue teams set up their systems by connecting them to the real internet. Once they are set up, they switch them to the competition network. This creates extra work and problems for blue teams. Part of the reason blue teams set up on the real internet is so they can access their systems directly on the network. This allows files to be transferred to/from the systems, as well as directly testing network services. Once the systems are on the competition network, they no longer have direct network access to them until they are also on the competition network. Thus, many teams leave their systems off the competition network during remote set up and move them to the competition network the day before the competition.

The competition network is supposed to model the real internet. However, the network configuration used by ISEFlow is hierarchical and fairly flat. In addition, making the network less flat would be difficult due to a limitation in ISEFlow that only allows each instance of

ISEFlow to have one outside router. ISEBox is made up of 4 or 5 instances of ISEFlow, so this limits how the competition network can be designed.

Modifying the competition network layout is difficult. The configuration file used by ISEFlow is hard to maintain by hand leading to mistakes due to human error. In addition, few people understand how it works, so it cannot be changed easily from competition to competition to support new scenarios.

CHAPTER 2. ENHANCING ISU CDCS

I explain proposed and implemented enhancements to IScorE, scoring in general, the scenario and flags, the green team, ISEFlow, and remote access to the competition network by blue teams.

2.1 IScorE

IScorE is a scoring system used during ISU CDCs. It was originally written by a senior design team and was used for one CDC. It served as a proof-of-concept and allowed us, the white team (ISEAGE staff) and me, to see how such a system could be used.

The first version took care of most scoring needs by allowing information to be entered digitally. For example, usability reports were automatically tallied. An automated service scanner was included as well. While the ability to have all the scores automatically recorded and summed up was helpful, the system they designed was not flexible. It was custom written without using a web application development framework. This meant that much of the functionality provided by typical web application development framework had to be implemented in a custom way, increasing the amount of time it would take someone to learn how the code works. In addition, the database tables were hard coded to store specific data. This meant that any changes to how the scoring was done would require both code and database changes.

Every CDC has different elements to it, so IScorE needs to be able to adapt to that. I proposed to the ISEAGE staff that we use a hierarchy of classes to define score elements. For example, a master class of documentation would have white team docs and green team docs as child classes. Score entries would then reference the child class. This allows for several tables to store many different types of scores. This design is similar to the one that the web based

online course system Moodle uses to store data. In addition, a general way of building forms for both content submission (from blue teams) and scoring should be used. For example, the green team leader could modify the usability check form without having to modify the web code or database by hand. Besides textual data, the forms should also allow for files to be attached. Max Peterson, an undergraduate ISU student and ISEAGE employee, implemented the current version of IScorE using this design.

IScorE greatly improves communication between the various teams by allowing information to be exchanged digitally. I specified that all green team members should have unique accounts and that each of the team members names be stored with their filled out usability checks. Thus, when blue teams had questions about their scores, the green team leader was able to look up who performed the check, and then talk to the green team member that preformed it.

In addition, IScorE helps the green team leader by allowing anomalies to be entered into IScorE ahead of time. The anomalies are then released to teams based on a given time. Anomalies can also be closed at a later time, allowing for anomalies with time limits to be specified. The name of the green team member that scores an anomaly is also attached to the score.

A modified version of the service uptime scanner that came with the first version of IScorE is used in the second version of IScorE. The original service uptime scanner scanned a predetermined set of hosts for each blue team and recorded the results in a database. This had three short comings: 1. The host names, port numbers, and URLs used to check the services required code changes. They were also the same for each team, preventing variations in blue team network setups. 2. Adding or removing services from the competition required code changes. It is common for each competition director to choose different services to be required. Thus, the services that need to be checked change at every competition. 3. The service checks were done in serial. So, team 1 would be checked, then team 2, and so on. While this may work ok if all the services for all teams are up and working properly, it does not work as well when services are down or slow. A timeout is configured to allow each service time to respond in case the server is under load. If the timeout is 15 seconds and there are 6 services per team, it is possible that the service scanner would need 90 seconds to check a team that is completely

down. If there are 20 teams, then it could take up to 28.5 minutes until the service scanner scans team 20. Service scans should be performed on a regular interval and should have an upper bound on the amount of time they take. For example, a scan every 5 minutes would be reasonable to track uptime during a competition. Thus, the scan needs to run in under 5 minutes. To accomplish this, I modified the service scanner to spawn a thread per team. Thus the upper bound for the time to scan is determined by the timeout value and the number of required services. Note that little processing time is required by the service scanner to perform a check, so the amount of time a check takes is mostly determined by how long it takes the service to respond to requests from the scanner.

Besides adding threading to the service scanner, I modified the scanner to pull a list of services from the IScorE database along with unique parameters for each blue team. This list includes the service type and the necessary information to perform a check. For example, an HTTP service would include a URL and a port. An SSH check would include a host name or IP, the port, and a username and password. The results of each service check are then stored in the database. This allows for services to be added and removed through the IScorE web interface without requiring code changes in the service scanner. In addition, each blue team has a unique set of service check information, allowing differences between the various blue teams' setups.

2.2 Scoring

The scoring used at the 2011 Community College CDC is given in Table 2.1 below.

Table 2.1 2011 Community College CDC Blue Team Score Allocation

Percent	**Item**
5.71	intrusion reports
5.71	white team documentation
5.71	green team documentation
14.29	red team evaluation
45.72	flags
11.43	service uptime
11.43	green team anomalies

The green team usability checks were used as a multiplier against the flag score. Each hour the green team was supposed to perform a usability check. The percentage of points awarded to the blue team was multiplied by 1/8 and added to the green team usability multiplier. This was to be done for eight hours. At the end of the competition the value of the green team usability multiplier was between 0 and 1. This was multiplied by the flag score in order to determine the final flag score. The purpose behind this system was to balance the usability of the blue teams' systems with protection from the red team. For example, blue teams could take their systems off-line, thus not losing any flags. Their usability multiplier would then be 0, resulting in no points for flags.

While this system sounds reasonable, in practice it did not work well. Attempting to do a green team usability check every hour is generally not reasonable, as there is not enough man power on the green team. In addition, the uptime service scanner performs much of the same function as the green team usability check. The green team usability checks as still necessary though, as they make sure the systems work properly from a human perspective. The green team members are able to perform more in depth checks than the automated scanner as well. However, performing several checks during the competition is enough to gauge the overall functionality of the blue teams' systems. If the green team spends too much time on usability checks, there isn't any man power left to grade anomalies. Since the service scanner is used to assign an uptime score to the blue team and keeps track of general service availability, the green team's time is better spent working on anomalies rather than completing more than a few checks during the competition.

I recommended we remove the multiplier and allocate the points in such a way as to make the red team and green team scores equally important. Thus, if the green team is ignored, the blue teams suffer huge point losses. If the red team is ignored, the same will be true. The final score breakdown used at the 2012 NCDC is given in Table 2.2 below.

The overall score breakdown by team is given in Table 2.3 above. Thus any blue team that wanted to place well in the competition would need to focus heavily on both the green team and the red team.

Table 2.2 2012 National CDC Blue Team Score Allocation

Percent	Item
5	intrusion reports
5	white team documentation
5	green team documentation
10	red team evaluation
30	flags
15	service uptime
15	green team anomalies
15	green team usability

Table 2.3 2012 National CDC Blue Team Score Allocation by Team

Percent	Item
15	automated systems
10	white team
35	green team
40	red team

2.3 Scenario and flags

To better simulate the real world IT environment, I wrote the scenario for the 2012 NCDC as if it was being written by the CEO of a small company. In order to set up a realistic scenario, I made up a hacking group that was responsible for compromising the company's systems. This group would also continue attacking the systems when they are brought back online. I introduced several other company employees throughout the scenario to give the blue teams an idea of who they were. Joe was the old IT admin that was fired. He serves as a possible disgruntled employee. Chris is the CEO's son and works with some of the systems. Chris often makes recommendations to Michael, the CEO. This fact is used later in the anomalies to request services and changes to the systems that may not be wise.

The other employees can be found in the username and password list along with each employee's job title. This allows employees to be given access to services based on their job roles. For example, the marketing team may need access to the company's blog, but not the janitor. It also allows for requests for new services and systems to be sent from these employees for plausible reasons. For example, the marketing director may ask for mailing list software to

be set up in order to send out a monthly newsletter.

Providing a realistic scenario based around a made up company with made up employees sets the stage for requiring blue team members to analyze their systems and classify the importance of their data. Historically flags were given to each team to place on their servers. These flags were put in places that required root or administrator access. If an attacker gains root access to a system, any information on the system is probably lost. However, it is common for systems to be compromised without the attacker gaining root access. For example, a web server could be compromised to host malware or a SQL injection could allow a database to be dumped. In addition, if the attacker learns the credentials of an important person in a company, such as the CEO or CFO, he or she would have access to sensitive information. In order to better simulate this type of compromise, I introduced the concept of hidden flags.

Typically flags are PGP encrypted files that hold no actual data that is pertinent to the company being simulated. A hidden flag is data on a system that acts as a flag. The blue teams can see and access that data, but they do not know that it will be treated as a flag. The number of hidden flags is known to the blue teams, but not their actual location.

For the 2012 NCDC, I placed two hidden flags on one system for each blue team. The first was a file in the CEO's directory that contained the following made up information

> To get info to the board:
> https://secure.directorsdesk.com/login.aspx
> username: michael.teeth.rule
> password:

Each team was given a unique password. Directors Desk is a web site used by company board members to communicate and schedule meetings. Thus highly sensitive information would probably be accessible by logging into Michael's Directors Desk account. There were several ways for the red team to gain access to this file if the blue teams were negligent. First, the directory permissions allowed anyone logged into the system to access it. Second, a backup script was written to back up the system. The backup location was accessible by anyone on the system. Third, either Michael's account or the root account could be used to access the data. So, even if the blue teams managed to keep the red team from gaining root access, that hidden

flag could still be compromised if they didn't fix the various directory permission issues.

The second hidden flag was a row in a credit card database. An example is given in Table 2.4 below.

Table 2.4 Example Database Row

Field Name	**Data**
team	1
member_id	889-29-4879
name	Kevin Koolwin
address	One Microsoft Way Redmond, XP, 88918
phone	082-398-1124
cc	8879192837879870
cc_expire	12/01/20
cc_ccv	8847
dues_paid	1

A simple web script was used to retrieve information from the database. By default it was written to be insecure. The red team could change parameters sent to the web script to access any data from any database. Blue teams needed to modify the script to only return the necessary information. In addition, a database administration web application was left wide open that blue teams would need to secure. So, if the blue teams were negligent, the red team could capture the hidden flag without needing to gain administrative system access.

Using hidden flags changed the workflow on how blue team systems were prepared. In the past, a master image was created for each system provided by the white team. This was done by installing an operating system and some programs onto a computer or a virtual machine. A copy of the hard disk information is then cloned onto other systems or virtual machines. The copy that is cloned onto other systems is known as a system image or a disk image. These images were then given to each blue team. Each system had to have a set of users on them to simulate real company employees. The account passwords were all the same for each blue team, and blue teams were not supposed to change them with the exception of the administrator password. Two changes to this workflow had to be made: 1. Some user accounts needed unique passwords so that the compromise of one blue team's username and password list would not lead to other blue teams losing their hidden flags. 2. A master image could no longer be given

directly to each team without first modifying it. We first deployed the master image to each team. We then booted the system, logged in, and assigned new passwords to two users. The blue teams were also provided with these passwords so they could use them on the systems they set up from scratch. We then modified the file in Michael's directory with a unique password. We also modified a specific row in the credit card database, changing the data in most of the fields. We then executed commands to cover our tracks and shut the systems down.

In an anonymous survey sent to the NCDC members after the event, 20 out of 21 participants thought hidden flags should be used in future events. Note that I was not able to filter out 2 test survey responses as my original survey data was lost in a compromise of the survey hosting site. I have summarized survey results with totals, but not individual responses.

When asked what they thought about the hidden flags, most members responded positively or indifferently. However, one member expressed the concern that in a real business the data to be protected would be known and not hidden. While that assumption is somewhat true, it is common in the real world for IT workers to determine the value of data based on its content and use. For example, the database of credit card numbers was a sensitive database, so it would warrant protection. The data wasn't hidden from the blue team members, but merely was not classified as a flag. Changing the term hidden flags to user data flags or some other term may help to clear up confusion about their intent. The purpose was not to intentionally hide sensitive data in a place that the blue teams would not expect, for example placing credit card numbers in /usr/bin, a directory used by Linux systems to hold programs. The purpose was to make the blue teams look at the data they were protecting in common locations, such as user home directories and relational databases used for web application, and decide how to protect it. In addition, the scenario provided listed what the role of each employee account was for, so the blue teams could ascertain that the CEO might have sensitive data in his home directory to protect.

Our experience with providing unique info to teams led us to adding the requirement to IScorE that it should be able to distribute unique data to each team.

Providing a slightly modified master image to each team does have a disadvantage. It would be possible for two teams to collude to determine what information is unique. For example,

the diff command, common on Linux and other Unix like systems, can compare all files and directories between two directories. The only way to work around this would be to provide completely unique data to each team. However, this is probably not realistic given the amount of effort that it would take to generate such data. A rule could be added to the competition that would forbid such collisions between teams.

Overall using hidden flags requires more effort from the competition organizers. The benefit is that the blue teams must analyze the data present on their systems and ensure that it is properly secured. This is a more complex task than simply ensuring that the red team cannot gain administrative access to their systems.

In order to help prevent teams from becoming discouraged and leaving the competition, I introduced the ability for teams to earn back lost flag points. Each flag was worth 50 points. If teams could describe how the flag was either stolen or planted, and recommend a way to mitigate the vulnerability, they could earn back up to 25 points per lost or planted flag. This gave the blue teams something to do once their flags were compromised. In addition, it rewarded teams for at least understanding the vulnerabilities present in their systems, even if they didn't know how to mitigate them. For teams that lost a flag due to a minor mistake, it allowed them to demonstrate that they did indeed know what they were doing, and to be rewarded for that knowledge. Overall the ability to earn back points places emphasis on the need for students to identify and learn how their systems were compromised by themselves, before the red team debriefing. Since this concept was used at the NCDC, most teams were highly motivated, so we weren't able to see how it affected teams that would have otherwise left early.

2.4 Green team

The number of anomalies and usability checks that the green team can run during a CDC is directly proportional to the number of technically skilled green team members. Unfortunately, it is common for there to only be a few such members. Typically people that have participated in a CDC are able to perform better than those who are new. This is especially true of past blue team members. I was the green team leader at the CDC for the IT-Olympics. I had enough skilled members to be able to perform both usability tests and attempt some new time

limit based anomalies. I also changed how teams perform usability tests.

In the past, at each usability check, a green team member would check the usability of a different blue team. This had two major problems: 1. Green team members had to spend time trying to figure out each blue team's environment by reading the provided documentation. 2. Green team members would commonly either skip reading the documentation or go through it too quickly resulting in a poor score for the blue team. The idea behind having each green team member grade a different blue team each time was to help keep the grading fair, as one green team member may be more generous with points than another.

I assigned several blue teams to each green team member for the duration of the event. This allowed each green team member to become familiar with the blue teams' systems they were assigned to. If a blue team contested their usability grade, they would work with the assigned green team member. If the green team member made a mistake, or if the blue team documentation was incorrect, the green team member noted this for the next usability check. Points were retroactively awarded if the green team member was at fault. If the blue team documentation was at fault, no points were awarded. However, the blue team was allowed to fix the problem resulting in more points for them during the next usability check.

In order to address the fair grading issue, I instructed the green team members to grade objectively. For example, in the past, a green team member may login to a Windows RDP server and run OpenOffice Writer. If the green team member didn't like the RDP environment, such as if the right mouse click was disabled on the Windows desktop, the member might take away points. Instead I structured the usability reports to allow only points for objectivity. In other words, either they could run OpenOffice Writer or they couldn't. There wasn't any room for the green team member to decide how usable the program was, just that it was at a bare minimum usable. The same sort of objectivity was used for other services. For example, for e-mail the green team member needed to be able to send an e-mail and receive it.

By assigning blue teams to green team members for the duration of the event and adjusting the usability reports to be more objective, green team usability checks were performed more quickly and consistently. The ability for the blue teams to check their scores via IScorE allowed the blue teams to work with the green team to ensure they were awarded all the points they

deserved. The extra time the green team members had was used to grade anomalies.

The quality of the blue team documentation given to the green team was graded at the end of the competition. Green team members were able to give their input based on how useful and complete they thought the documentation was. This allowed the green team members to comment on a blue team's environment once instead of at every usability check. In addition, the red team evaluates each blue team's environment and assigns them a score. This score also takes into account the usability of blue team systems. For example, if a blue team locks down a system to the point where it is unusable, the red team will not award them as many points.

I introduced time limited anomalies at the IT-Olympics as well. For example, blue teams would only have 15 minutes to delete or disable an account once notified, or a consultant account would need to be created within 30 minutes. These anomalies mirrored the real life requirements of needing to complete specific IT tasks within a tight time window. For example, it is common for companies to have IT policies that require user accounts to be terminated immediately after an employee is terminated or quits. An IT emergency might occur with the need to bring in an outside consultant. Commonly these consultants need accounts created for them so they can interact with the IT systems. In both of these situations, time is critical. If the action is not taken quickly, the results could be dire. For example, a terminated employee could steal company information or attack company systems. If the consultant doesn't have an account, he or she may not be able to fix the IT systems.

Having time limited anomalies does pose a logistical challenge for the green team. In order to process the anomalies as they are submitted, they either need to stop what they are doing or they need to be idle. In general, performing usability tests takes the most time. If a green team member has four blue teams to check, it might take him or her up to an hour to complete the checks. In order to be as fair as possible, the checks should be completed at approximately the same time. This is because as the competition goes on, the blue team's systems are commonly taken down by the red team. The difference of a half an hour could be significant to a blue team's usability score. Therefore it is not desirable to stop green team members in the middle of performing usability checks. The same is true of other anomalies. If blue teams need to set up a forum on their web server, and that web server is taken down, they will not get points for

the anomaly. So it is possible that the blue team may complete the anomaly, but 30 minutes later their web site goes down before the anomaly can be graded. Although blue teams have requested rescoring of anomalies after bringing services back up, they commonly do not manage to bring their services back up.

Overall it is important to process anomalies and usability checks as quickly as possible. Commonly these two checks can be done simultaneously. For example, if the blue teams were requested to set up a forum on their web server, this could be checked while the green team member is checking the blue team's web server. If the green team falls behind on usability checks and anomaly grading, performing a time limited anomaly would prevent the green team from catching up. Thus the time limited anomalies have to be released by the green team leader when there are low points in the green team workload. The green team leader announces the anomaly to the green team and instructs the members to watch for submissions from the blue teams. Once a submission is received, the green team immediately checks it. If the submissions are not checked immediately, then the blue teams could submit the anomaly and then perform the work once the time limit has expired. So, while time limited anomalies can add some realism to the simulated IT experience, they can also cause logistical problems for the green team.

One important aspect that I introduced to the green team usability checks was the persistent data check. Now that teams are using virtualization systems, it would be easy for them to snapshot their systems. If the system is attacked and defaced by the red team, the blue team can simply revert their system back to the state of the snapshot. This sort of action would not be ok in the real world, as user data would be lost. In order to detect this type of action by a blue team, I instructed green team members to add data to certain systems during the first usability check. In subsequent checks, the green team would check to make sure that this data still exists. If it does not, the blue teams lose usability points.

In addition to the data persistence check, at the NCDC a file restore anomaly was added as well. As part of the scenario, blue teams were instructed to perform hourly file level backups of user data. Green team members again created data during their usability checks. The data is then later deleted by the green team after several hours. The blue teams are then given an

anomaly requesting this data to be restored. Both data persistence and working backups are important parts of any IT system.

For the CDCs I also built and secured a green team e-mail server. This was a Scientific Linux 6 system configured with Postfix, Dovecot, and Roundcube for SMTP, IMAP, and web based e-mail access respectively. Each blue team was given an account on the system. This allowed the blue teams to test their e-mail services to make sure they could send and receive mail from their systems to the green team e-mail server. Before IScorE was used, I also used this server to contact the teams regarding green team anomalies and other pertinent green team information. This server both assisted the blue teams in ensuring their e-mail services were working properly, and gave the green team a reliable way to test them.

2.5 ISEFlow

The operation of ISEBox is easy for anyone with IP networking background to understand from an abstract level. ISEBox is made up of several virtual machines. Each machine is called a board. Each board runs one instance of the ISEFlow program which performs IP routing. Each board contains a set of virtual IP routers with statically defined routes. IP traffic can be routed between the boards, within the boards, and to IP networks outside of ISEBox. The virtual routers can be ping'ed (ICMP ECHO). The virtual routers that connect to outside networks can send and receive Address Resolution Protocol (ARP) requests and responses. This allows the routers to associate an ethernet MAC address with an IP address on the outside network. It also allows other devices on the outside network to do the same with the virtual router's IP and MAC address. Besides pinging and ARP the virtual routers don't support any other network services. Each router has a set of interfaces that is used to receive IP packets from other routers. Each board can have one outside router that is able to route packets in and out through a physical ethernet interface. This limitation makes it hard to create networks that model the internet with a small number of boards.

To overcome that limitation, I modified the ISEFlow code to allow each board to have more than one outside router. The code itself was mostly ready for this. The modifications can be summed up in several major steps: 1. Allow more than one outside router to be configured per

board. 2. Remove the assumption that router 0 is always the outside router. 3. Determine the correct outside router and pass this information to all necessary function calls.

Configuring ISEBox is harder than simply understanding its operation. The routing tables for each router require the destination network, the next hop IP, and the next hop board-/router/interface number to all be defined by hand. For example:

Dest IP,	mask,	type,	next IP,	next board,	next router,	next interface
10.0.4.0,	24,	internal,	10.0.1.2,	0,	1,	0

Entering all this information by hand is error prone. It also makes changing the network configuration hard, as any affected routing table entries must be updated by hand throughout the entire file. While working on the modifications I made to ISEFlow, I commonly made small mistakes when modifing the network configuration. At one point I even created a routing loop. I constantly had to question whether my code modifications were causing bugs or whether config modifications were causing bugs.

In order to make it easier to modify existing ISEBox configurations, and to allow for simpler creation of larger ISEBox configurations (more than 10 routers), I wrote simple-isemaker in Python 2.6. simple-isemaker takes two files as input and outputs a valid ISEBox config file. The first file is a list of networks in the format:

name, network number, bit mask

Where name is text, network number is an IP network address with no host bits set, and bit mask is the bit mask applied to the IP network address to define an IP subnet. For example: a,10.0.1.0,24

The second file is a list of routers in the format:

router, board, outside, neighbor/network pairs

Where router is a positive integer, board is a positive integer, outside is either 0 or 1 for false or true, and neighbor/network pairs are a list of the router's neighboring routers and the networks that interconnect them. For example: 5,2,0,0,c,2,f,6,h,9,g

The program needs to determine how to route an IP packet from any router to any network subnet in the ISEBox network in order to build each router's routing table. To do so, the read in configuration data is used to create a directed graph. The routers are vertices or nodes.

The adjacency list of neighbors for each router is used to create directed edges. The network subnets are properties of the nodes. A modified breadth-first search is run once for each router as a root node. Since breadth-first search determines a path that is of shortest distance from the root node to each node in the graph[2], this can be used to find a route of shortest distance to any network in the graph.

Pseudo code for the following modified breadth-first search algorithm can be found in the appendix. The first step is to mark all nodes and networks as not visited. Next, the networks that are properties of the root node are marked as visited. An empty list is then allocated for each neighbor node to keep track of reachable networks that are closest or as close to that node as any of the other neighbor nodes of the root node. Next, each neighbor node is appended to a FIFO (first-in, first-out) queue along with the neighbor's identity. This identity is tracked as the graph is traversed.

The program then loops by dequeueing a node from the queue until the queue is empty. With the dequeued node is the identity of the node that is of distance 1 from the root node. This node is an ancestor of the dequeue node. So it is the node that the root node should send any packets to that need to reach unvisited networks on unvisited neighbor nodes of the dequeued node. I shall refer to this node as the first hop ancestor node. Each of the dequeued node's neighbors are appended to the queue if they have not been visited before. The first hop ancestor node is append along with the neighbor nodes. The appended neighbors are marked as visited. Each of the networks for each neighbor node is checked to see if they have been visited. If they have not been visited, then the network is added to the list of networks for the first hop ancestor node. The network is then marked as visited.

Once the loop exits, the modified breadth-first search is complete. Each neighbor node of the root node now has a set of networks that are closest to it, or of equal distance to it, than any other neighbor of the root node. The sets are disjoint, so the root node can use this information to build a routing table. The root node can then use this table to route a packet to a neighbor that is closest to the destination network.

To build a routing table for each router, this modified breadth-first search is run on each router. Once this is complete, the necessary steps to generate a usable isemaker config begin.

Each router is numbered uniquely in the simple config file for the entire ISEBox network, but each board needs to have a uniquely numbered index of routers starting at 0. Each router is assigned a unique index number on the board it belongs to. The routers that act as outside routers are assigned the lowest index numbers, followed by the inside and internal routers.

The interfaces for each router need to be indexed as well. An interface is needed for each network that the router connects to. The outside interfaces have the lowest index numbers. Each interface needs to be assigned an IP from the network it connects to. This is accomplished by assigning an IP from the pool of IPs available in the network. If the interface is an outside network, then the last IP in the network before the broadcast address is used. For example, for the outside network 10.1.2.0/24, the address 10.1.2.254 would be allocated. For the outside interfaces, an ethernet MAC address must be assigned. This is accomplished using a MAC address generator that is provided with a seed. The MAC addresses are generated in sequential order from the given seed.

Once the interfaces for each router have been configured, routing tables usable by ISEBox need to be configured for each router. For each neighbor node, the type of connection to the neighbor needs to be determined. This is done by checking to see if the neighbor is on the same board. If it is, it is an internal connection. If not, it is an inside connection. Next, the interface that the neighbor router uses to communicate with the current router is looked up using the network identity given with the neighbor in the network configuration file. For the neighbor, the set of networks closest to it are enumerated. A routing table entry is made for each network using the network number and bit mask along with the interface information previously determined.

The ISEMaker configuration file is now complete. Since simple-isemaker is designed for common network scenarios, it makes assumptions about how the network is structured and how much information the user needs to provide. In order to write simple config files for simple-isemaker, the user is not required to allocate IP addresses for each interface used by each router. MAC addresses need not be specified for outside interfaces either. This may not be desirable in all situations, but in order to support specifying this type of information, the size and complexity of the simple-isemaker config files would increase.

In addition, several assumptions are made about the network structure. First, it is assumed that any outside network is present on only one router. Second, it is assumed that a router can communicate with its neighbors symmetrically over the same network. So, if router 1 connects to router 2 via network A, it is assumed that router 2 connects to router 1 via network A as well. Third, network IP ranges do not overlap. These assumptions are checked during the configuration file generation. If any are violated, the program quits with an error.

The program makes use of the IP Address Manipulation Library for the Python Standard Library. This library was recently developed and made a part of the Python standards[3]. Although it is not part of Python 2.6 yet, the reference implementation is available. I used this library to store, manipulate, and validate IP addresses and network specifications. Thus, any error in specifying the network number for a network in the networks config file will be caught and reported to the user. In addition, I used this library to find the last usable IP address in a network before the broadcast address and to enumerate through IP addresses in a network. It is also used to perform the checks to ensure that networks do not overlap.

Overall, simple-isemaker allows for two simple configuration files to be used to generate the more complex ISEBox configuration file. This will hopefully allow anyone who is familiar with IP network design to create an ISEBox network configuration quickly and easily. All of the configuration input is validated and checked for errors which reduces problems caused by human error. In addition, many of the opportunities for human error in an ISEBox file are removed since much of the generated config file is made automatically. For example, indexing routers, interfaces, and referencing those indexes in routing tables. This will also make modifying the network configuration easier, as fewer edits will need to be made by hand. In addition, due to the error checking, many simple errors made while modifying the network configuration will be caught by the program.

2.6 Remote access to the competition network

It would be useful for teams to be able to access their systems on the competition network. Currently, the team must either use the virtualization system management client or be connected directly to the competition network. It would be desirable to allow teams to connect

directly to their systems to manage them and to test them. I've found three solutions to allow teams to connect to the competition network remotely from the real internet. 1. Use proxy servers 2. Use a VPN 3. Set up a Twice NAT for services

2.6.1 Proxy servers

They keyhole is used to connect the ISEBox network to the real internet. It is made up of two HTTP proxy servers. Client requests from the competition network are sent to keyhole 1. Keyhole 1 then forwards the request to keyhole 2 over a private IP range. Keyhole 2 then performs any necessary DNS lookups and sends the request to the intended destination. Any replies are sent back to keyhole 1, and then to the client. The Squid HTTP proxy server is used. It supports HTTP proxying and the HTTP CONNECT method proxying. With HTTP proxying, the data sent between the client and the server are in clear text. The data can be modified by the proxy server. Only the HTTP protocol can be used through the proxies in this manner. HTTP CONNECT is provided to support HTTP over SSL, which prevents man-in-the-middle attacks, so the proxy server cannot modify the data. The proxy server must pass the data between the client and the server unmodified. Besides being used for HTTPS, many client applications can use the HTTP CONNECT method to access services through a proxy. Two examples are SSH and FTP. In order to allow remote access to the competition network, the keyhole could be designed to work in both directions. A username and password could be required to access the proxy server on keyhole 2 in order to send requests to the competition network. Unfortunately, Microsoft's remote desktop protocol (RDP) does not support connections through an HTTP proxy using the HTTP connect method. However, this would allow access for services such as e-mail, FTP, SSH, and web.

2.6.2 VPN

A Virtual Private Network (VPN) can be used to allow teams to connect to the competition network remotely by carefully choosing the competition IP ranges and modifying the routing table on the client system. In order for teams to access the competition network, routes for each blue team network could be added to the client's routing table. These routes would direct

traffic destined for those networks through a VPN connection that would terminate within the competition network. To allow the teams to continue to use their internet connection normally, only the networks that are assigned to the blue teams should be routed through the VPN.

Although the real internet and the competition network IP ranges overlap, typically internet users do not need to connect to other internet users. So, in order to prevent the overlap from causing a problem, IP address ranges could be chosen from ISPs around the world that provide internet service to consumers. Thus statically routing traffic from these ranges on the real internet to the competition network shouldn't interrupt the user's normal internet use.

PPTP VPN clients are built into most consumer operating systems. However, they do not provide a good way to specify a large number of static routes to be pushed to the client. OpenVPN does provide such functionality, but it would need to be installed by users on their computers. This may be problematic for users that do not control the computer they use. For example, when a student is using a computer in a school computer lab. However, most students participating in a CDC do have their own computer, so this would be a reasonable approach.

An OpenVPN client configuration file and a self signed certificate would need to be given to each team member. The configuration file contains all the information needed to connect to the VPN. Once the client connects, the static routes are pushed to the client. The VPN server gives out private IPs via DHCP. The VPN clients are then NAT'ed behind an IP on the VPN server. The VPN server would be on the real internet, but would also be connected to the ISEBox network. In order to route traffic to the ISEBox network, a static route would need to be created for each network that should be reachable through the VPN. These static routes would all point to the same router that would be present on the ISEBox network.

For teams that set up a firewall to protect their networks, they could also set up a VPN to allow access to their internal network. I was able to successfully use a client on the real internet to connect to the ISEBox network via a VPN. I then created another VPN connection through the first in order to connect to a network protected by a firewall.

Note that although I specified that a NAT should be used for the VPN clients when accessing the competition network, it is not necessary. The VPN clients could be assigned IPs on the competition network itself. However, this would allow any system within the network to access

the VPN client's system. This is probably not desirable.

Although this setup would require educating the blue team members and would require more setup and maintenance for the white team, it would provide an effective way to allow teams to directly access systems on the competition network. The one major problem would be selecting IP ranges for the competition network and the private IP range for the VPN clients such that no blue team member would have problems with legitimate internet traffic being redirected through the VPN. Given the large number of IP addresses available that do not host internet services, I think this problem can be overcome with careful planning.

2.6.3 Twice NAT

Typically NAT is used to allow the use of private IP addresses to connect to the internet. A client with a private IP sends a request that is routed through a device that has both a private and public IP. The device rewrites the IP header information to make the request appear to come from its public IP. The request is then sent to the destination. Any replies are sent back to the client system after changing the IP from the device's public IP to that of the client.

To use NAT to allow for remote access to services on the competition network, the NAT must be performed in both directions. According to RFC 2663, this is known as Twice NAT[4]. First, a port is opened on a system connected to the internet. Client requests are sent to the system on this port. This system then forwards the request to an IP and port on the competition network which is the true destination of the client request. It changes the source of the IP packet from the client's IP address to its own IP address that is used to send traffic to a competition network router. It then changes the destination IP address and port number to that of the service on the internal network. Once a reply destined for the client is received, it reverses this process.

I successfully tested this setup using the FreeBSD Packet Filter (PF) to perform the NATs. The configuration file I used can be found in the appendix. In order for blue teams to use this setup to access their systems, a NAT would need to be created for each one of their services. In addition, since these services would be exposed to the internet, the blue team should probably be required to provide their client IP addresses that would be accessing the

NAT. This would prevent malicious people on the internet from finding the vulnerable CDC systems and exploiting them. The setup would either have to be created by hand, or optimally, through a system where blue team members could request access and the NAT rule would be created automatically.

The Twice NAT setup could also be used to create jump points within the competition network. For example, a Windows system running RDP or a Linux system running SSH could be placed on the competition network. Blue team members would then connect to these systems via a Twice NAT. From there, they would be able to jump to other systems via RDP or SSH. In addition, SSH can be used to forward ports between the client and the server. The blue team members could use this functionality to directly connect to their services.

CHAPTER 3. ISEBox on KVM

In order to provide a CDC in a box to other institutions, it is necessary to pick and assemble the software that will form the base of CDC in a box. I chose to use Linux and the Kernel-based Virtual Machine (KVM) virtualization environment to form this base.

RedHat Entperise Linux (RHEL) 6 supports KVM as the primary virtualization environment. Although a license must be purchased to use RHEL, there are free versions of RHEL that are built using the source code that is given out by RedHat to build RHEL. RedHat is required to give out this source code as their software is licensed under the GNU Public License (GPL). Two such free Linux distributions are CentOS and Scientific Linux. Therefore users of the CDC in a box can either purchase support for the deployment of the virtualization infrastructure or they can use a free community supported version. I used CentOS for development and testing.

I was given a document entitled "ISEMaker" along with the ISEFlow code. ISEMaker contained instructions to set up ISEBox on VMware Server. At the time, VMware Server had just been retired by VMware, so a new virtualization environment needed to be chosen. VMWare had several replacement products, such as VMware Workstation, VMware Player, and VMware ESXi. Each of these products had advantages and disadvantages when compared with KVM. VMware workstation requires a license to be purchased. In addition, it is not designed to run virtual machines for long periods of time acting as servers; it is primary used for development. VMware Player is similar to VMware workstation, but it is free. It has less features that workstation. VMware ESXi is VMware's flagship virtualization environment. Unlike VMware workstation and player, which are installed on top of an existing operating system such as Windows, ESXi runs directly on top of the system hardware. This means that a system has to be dedicated to running just ESXi. In addition, this system could not be used by the ESXi operator to maintain the ESXi environment. A separate client package must be

installed in Windows to perform the administrative operations. Due to ESXi running directly on hardware, the range of hardware it supports is limited, so not all existing systems will be compatible with ESXi. The basic version of ESXi is free. However, to gain more advanced capabilities such as the ability to script the ESXi environment, a license must be purchased from VMware[5].

KVM runs on top of the Linux kernel. This means that any hardware that Linux supports, KVM will support. The only exception is processors. Processors must have virtualization instructions built into them in order to utilize KVM. These extensions are Intel VT-x or AMD-V. Both are found in most modern processors. Any server class system will have them, and most consumer class systems have them as well. Some of Intel's cheapest processors do not have the Intel VT-x extension. These are found commonly in budget laptops.

KVM is easy to install on a RHEL or CentOS system. Once installed, there is a GUI based administrative interface called Virtual Machine Manager (VMM). There is also a command line utility called virsh. The VMM is an X Windows based application and can be run remotely via X Forwarding. Since virsh is a command line application, it can be used over SSH. This allows for the environment to be remotely administered. In addition, virsh can be used in shell scripts in order to automate the configuration and operation of KVM.

I replaced most of the documentation in the ISEMaker document to be KVM specific. I also added details about how to set up each system needed for ISEBox. To update the document I added details on how to install RHEL or CentOS and prepare the KVM environment. Next I prepared documentation and scripts to set up ISEBox. ISEBox is made up of the following systems: 5 Boards, 3 FreeBSD test systems, 2 FreeBSD systems for the keyhole, 1 FreeBSD system to control the boards named snowflake, 1 pfSense firewall, and 3 Windows systems. The documentation details the steps to set up each system. Previously only the set up of snowflake and boards 1-5 were somewhat explained. Scripts and XML configuration files are included to configure most of the KVM setup automatically.

The configuration for KVM is stored by KVM in XML files. These files can be exported to save the configuration information. For example, each virtual network has an XML configuration file, and each virtual machine has an XML configuration file. The exported files can be

modified and used to configure or reconfigure KVM.

To begin the set up process, the user must create a storage pool to store operating system installation disk images. The user must also create the bridged interface that will be used to connect ISEBox to the internet. Once these steps are complete, a network configuration shell script is run to configure the virtual networks. This script uses network XML configuration files to create each network work, to start it, and to set it to autostart upon system bootup.

Next, the user can modify the virtual hard disk storage pool location if the default location used by KVM is not acceptable. The user then runs a script to auto create the virtual disks used by the initial VMs that need to be configured. Next, the user runs a script to create the initial VMs. At this point, the user must configure snowflake by hand. This involves following the given instructions to set up FreeBSD and build the ISEFlow code. Next, the user follows the instructions to set up board 1. The user then shuts down board 1 and uses a script to clone board 1 to boards 2-5. The user then must boot up each board and follow the given instructions to make each board unique. At this point ISEFlow can be configured and started.

Keyhole 1 and 2 are then created. The keyhole systems run the Squid HTTP proxy software to allow the systems on the ISEBox network to access web sites on the real internet. Keyhole 2 is set up first by the user. Next, keyhole 2 is cloned to create keyhole 1. The user must then make minor modifications to the cloned keyhole 1. The NAT-FW system is set up following the given instructions at this time. Finally, the Windows and FreeBSD test systems can be created however the user wishes. These can be cloned as well once an initial test system is created.

CHAPTER 4. SUMMARY AND FUTURE WORK

4.1 Summary

I have described various enhancements that can be applied to the CDCs hosted by ISU to improve their operation. IScorE's integration into the CDCs is a major improvement in score accuracy, communication between the blue teams and the white/red/green teams, and time spent tabulating scores. The scoring itself has been refactored to balance the blue teams' requirements to both secure their systems while maintaining usable systems. Hidden flags have been introduced which better approximate the type of data normally found on typical IT systems. Blue teams must analyze the data on their systems in order to protect the hidden flags. The ability to earn back points from lost or planted flags gives the blue teams the role of a typical IT team that has had a system compromised. They must learn what happened, how it happened, how to fix it, and how to prevent it from happening again. Hopefully this will lead students to learn how their systems were compromised instead of waiting for the red team debriefing or simply giving up and going home. Modifications have been made to ISEFlow to allow it to better model the structure of the internet with only a few boards. In addition, simple-isemaker allows anyone with IP networking background to be able to configure a network for ISEBox. Each of the remote access solutions presented would allow blue teams to have direct access to their systems on the competition network, allowing them to set up and test their systems more efficiently. The ISEBox on KVM set up documentation and scripts provide the necessary information for anyone to set up their own ISEBox even if they are not familiar with virtualization technology or ISEFlow. This work forms the basis for future work on the CDC in a box project.

4.2 Future work

In order to work on the requirements for the CDC in a box, I needed to participate in multiple CDCs as a blue team member and as a white and green team member. These experiences taught me that running a successful CDC requires many experienced people all doing their respective jobs. I focused much of my research on how the green team should function in order to positively affect the competition. After enhancing the green team and other elements of the competition, I think it is ready to be shown to other institutions. The following steps should be completed in order to complete the CDC in a box project. 1. A simple-isemaker config should be written to model the real world wide internet topology. This could be accomplished with around 40 routers. I began work on modifying ISEFlow and creating simple-isemaker with the goal of being able to make such a network configuration. 2. The KVM based ISEBox and newly made ISEBox network configuration should be tested and then used at an ISU CDC. 3. One or more of the remote access options presented should be implemented and used for an ISU CDC. 4. If all the testing goes well, the support systems needed to run the CDC should be implemented on top of KVM along with the ISEBox systems. 5. The amount of configurability that a potential user of the CDC in a box would need should be determined. Based on that, the ISEBox systems and support systems should be templated to allow the necessary configuration changes to be made via shell scripts. Optionally, another type of interface could be used to make the changes such as a web interface. 6. A downloadable copy of the ISEBox and support systems would need to be provided, as well as the necessary documentation needed to setup KVM. Scripts should be provided to install and configure the systems that make up the CDC in a box. The existing ISEMaker guide and support scripts would form the basis for this. 7. Guides need to be written by experienced participants of the CDCs. A guide would be needed for each of the white, green, red, and blue teams. In addition, a general overview document would need to be written. 8. Example scenarios and anomalies should be collected and made available. 9. Additional ideas for scenarios and anomalies should be provided.

Once these steps are completed, the CDC in a box would be ready for use by other higher learning institutions.

APPENDIX

An example configuration file that creates a Twice NAT on a FreeBSD system is provided as well as psuedo code for the modified breadth-first search algorithm.

FreeBSD example PF configuration file for Twice NAT

The following file (/etc/pf.conf) is taken from a working FreeBSD based Twice NAT setup. Comments are provided to explain each line of the file. Comment lines begin with the # character. Documentation for the PF can be found online[6].

```
# hosts on this interface don't have a route to the int_if
ext_if = "em0"
# hosts on this interface use this as their default gw
int_if = "em1"
# hosts on int_if
lan_net = "199.100.16.0/24"

# don't filter the loopback interface
set skip on lo0

# all connections to the IP on the ext_if to be redirected (port forwarded) to
# the IP on the internal network (not this machine) on from port 1022 to 1099
rdr pass on $ext_if proto tcp from any to 192.168.24.100 port 1022 ->  \
                                                    199.100.16.95 port 1099

# for testing purposes, if allowed, the internal interface hosts can access
# the external interface hosts via nat
nat on $ext_if from $int_if:network to any -> ($ext_if)

# NAT rule from the external interface to the internal interface to the IP
# running the desired service, the port is forwarded by the rdr rule, this rule
# provides the NAT, this machine will appear to be the source IP, even though
# it is not
nat on $int_if from $ext_if:network to 199.100.16.95 port 1099 -> ($int_if)

# default rules to block all traffic, note that the rdr rules with pass are
```

```
# not affected
block in all
block out all

# only allow traffic destined for IPs on this host, that should prevent
# traffic from being routed through this system
pass in on $int_if from any to 199.100.16.100
pass in on $ext_if from any to 192.168.24.100

# allow any traffic out, as the in rules limit what will be routed
pass out on $int_if to $lan_net
pass out on $ext_if to any
```

Modified breadth-first search psuedo code

The following pseudo code is written using the Python programming style and language syntax.

```
for node in routers:
    node.visited = False
for network in networks:
    network.visited = False
for rootNode in routers:
    rootNode.visited = True
    for network in routers.network:
        network.visited = True
    for neighbor in rootNode.neighbors:
        neighbor.visited = True
        for network in neighbor.networks:
            if network.visited == False
                rootNode.nextHop[neighbor].add(network)
                network.visited = True
        queue.append((neighbor, neighbor))
    while length(queue) != 0:
        (node, ancestor) = queue.dequeue()
```

```
for neighbor in node.neighbors:
    if neighbor.visited == False:
        queue.append((neighbor, ancestor))
        neighbor.visited = True
        for network in neighbor.networks:
            if network.visited == False
                rootNode.nextHop[neighbor].add(network)
                network.visited = True
```

BIBLIOGRAPHY

[1] D. Jacobson and N. Evans, "Cyber defense competition," in *Proceedings of the 2006 American Society for Engineering Education*, Chicago, Jun. 18–21, 2006.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms.* The MIT Press, 209, p. 594.

[3] P. Moody, "IP address manipulation library for the python standard library," PEP (Python Enhancement Proposals) 3144, Feb. 2012. [Online]. Available: http://www.python.org/dev/peps/pep-3144/

[4] P. Srisuresh and M. Holdrege, "IP network address translator (NAT) terminology and considerations," RFC 2663, Aug. 1999. [Online]. Available: http://tools.ietf.org/html/rfc2663

[5] VMware, Inc. (2012, Jul.) FAQs for VMware vSphere Hypervisor, free server consolidation (based on VMware ESXi). [Online]. Available: http://www.vmware.com/products/vsphere-hypervisor/faq.html

[6] J. Ferrell, *Chapter 31 Firewalls*, FreeBSD Handbook, The FreeBSD Documentation Project, 2012. [Online]. Available: http://www.freebsd.org/doc/handbook/firewalls-pf.html