REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* APR 2012	**2. REPORT TYPE** Conference Paper Post Print	**3. DATES COVERED** *(From - To)* JAN 2010 – JUL 2011

4. TITLE AND SUBTITLE
DefEX: HANDS-ON CYBER DEFENSE EXERCISES FOR UNDERGRADUATE STUDENTS

5a. CONTRACT NUMBER N/A
5b. GRANT NUMBER N/A
5c. PROGRAM ELEMENT NUMBER N/A

6. AUTHOR(S)
Sonja Glumich, Brian Kropa

5d. PROJECT NUMBER GAIH
5e. TASK NUMBER CY
5f. WORK UNIT NUMBER BR

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/Information Directorate
Rome Research Site/RIGA
525 Brooks Road
Rome NY 13441-4505

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/Information Directorate
Rome Research Site/RIGA
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
N/A

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-RI-RS-TP-2012-007

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA #: 88ABW-2011-2709
DATE CLEARED: 12 MAY 2011

13. SUPPLEMENTARY NOTES
Publication in WorldCom 2011, Security and Management Session Proceedings, 18-21 July 2011, Las Vegas, NV. This is a work of the United States Government and is not subject to copyright protection in the United States.

14. ABSTRACT
DefEX incorporates a set of hands-on cyber security exercises aimed at developing problem-solving proficiency, teamwork, and cyber defense skills in undergraduate students. The exercises include Code-Level and System-Level Hardening, Static and Dynamic Reverse Engineering, Detect and Defeat, Digital Forensics, and the Wireless Access Point Treasure Hunt. Providing a diverse group of students with a common set of foundational knowledge and finding the balance between enabling participation of novice students and generating problems complex enough to challenge experienced students posed the major curriculum design risks. Instructors reduced the risks by administering a technical survey, requiring students to complete a set of fundamental exercises, and assigning balanced student teams. As a result, student teams successfully completed all of the exercises.

15. SUBJECT TERMS
Reverse Engineering, System Level Harding, Cyber Defense Exercise, Cyber Security Curriculum, Cyber Security Education.

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** JAMES S. PERRETTA
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	8	**19b. TELEPHONE NUMBER (*Include area code*)** N/A

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

DefEX: Hands-On Cyber Defense Exercises for Undergraduate Students

Sonja M. Glumich and Brian A. Kropa
Cyber Sciences Branch, Air Force Research Laboratory, Rome, NY, USA

Abstract - *DefEX incorporates a set of hands-on cyber security exercises aimed at developing problem-solving proficiency, teamwork, and cyber defense skills in undergraduate students. The exercises include Code-Level and System-Level Hardening, Static and Dynamic Reverse Engineering, Detect and Defeat, Digital Forensics, and the Wireless Access Point Treasure Hunt. Providing a diverse group of students with a common set of foundational knowledge and finding the balance between enabling participation of novice students and generating problems complex enough to challenge experienced students posed the major curriculum design risks. Instructors reduced the risks by administering a technical survey, requiring students to complete a set of fundamental exercises, and assigning balanced student teams. As a result, student teams successfully completed all of the exercises.*

Keywords: Cyber Defense Exercise, Cyber Security Curriculum, Cyber Security Education

1 Introduction

DefEX consists of hands-on cyber security exercises designed to promote problem-solving proficiency, teamwork, and cyber defense skills in undergraduate students. Students worked in teams of three to complete the following exercises: Code-Level and System-Level Hardening, Dynamic and Static Reverse Engineering, Detect and Defeat, Digital Forensics, and the Wireless Access Point Treasure Hunt.

DefEX falls under the full spectrum of cyber defense activities including preventing attacks, detecting attacks, surviving attacks, and recovering from attacks. The Code and System-Level Hardening exercises address proactively preventing attacks before they occur, the Detect and Defeat exercise involves discovering and countering attacks, the Forensics exercise involves enabling recovery efforts by analyzing the aftereffects of attacks, and the Reverse Engineering exercises address analyzing malicious code to aid in surviving and recovering from attacks.

Student participants were rising seniors studying Computer Science, Computer Engineering, Electrical Engineering, Mathematics, and Physics at academic institutions across the country. The major curriculum design challenge involved enabling participation of novice students and generating problems complex enough to challenge experienced students. Students all possessed one semester of computer programming in a high level language, a semester of discrete math, and a year of calculus. While enforcing the set of minimum academic requirements helped, instructors took additional steps to account for the varied backgrounds of students.

Accommodations included creating a technical survey and a set of fundamental exercises and assigning balanced student teams. Issuing the technical survey to students before the exercises to assess prior experience in areas such as networking, computer security, and digital logic helped instructors tailor background materials for the exercises. Requiring students to complete a set of fundamental exercises before the DefEX exercises ensured a common familiarity with VMware server, the Linux operating system, networking protocols, and the client-server model. Assigning students into teams to maximize the diversity of college majors created balanced teams performing on near-equal footing. For example, a team might consist of a mathematics student, a computer science student, and a computer engineering student.

2 Code-Level and System-Level Hardening

2.1 Background

Code-level hardening involves activities undertaken by software developers or testers to produce secure source code. System-level hardening includes actions carried out by users or operators to securely configure an existing system. The software development lifecycle (SDLC) serves as a beneficial contextual framework for teaching secure application design, implementation, testing, and hardening. The instructor presented the SDLC in seven phases: 1) Specify (Requirements), 2) Architect, 3) Design, 4) Implement, 5) Test, 6) Deploy, and 7) Maintain. Software developers and testers engage in code-level hardening mainly during the Implement and Test phases, and operators conduct system-level hardening during the Deploy and Maintain phases.

One point reinforced by examining security activities in the context of the SDLC was that code-level and system-level hardening activities only eradicate a limited number of vulnerabilities introduced in hardware or in the early SDLC phases. Code-level hardening has a limited ability to remove vulnerabilities introduced during the Specify, Architect, or

Design phases. System-level hardening is additionally restricted in fixing Implement and Test phase vulnerabilities. Conversely, developers and testers have a limited amount of influence in helping users to securely configure a system in the Deploy and Maintain phases. Designing a user friendly system defaulting to strict security settings helps, but doesn't guarantee secure configuration.

This examination contradicts the validity of the practice of blaming the user for current cyber security woes. Most users lack the ability to remediate architectural, design, or code-level vulnerabilities such as those existing in proprietary operating system binaries. It also belies the notion that if cyber operators are diligent enough while monitoring packets and intrusion detection system (IDS) alerts they can successfully thwart all attacks during the Deploy and Maintenance phases. Although there exist many knowledgeable, committed cyber operators, by the time they deploy a majority of systems, the cyber security battle was already lost earlier in the SDLC.

2.2 Exercise Goals

The aim of the Code-Level and System-Level Hardening exercises was for students to grasp the complexities and manpower involved in developing a secure system. In the context of the SDLC, students should have comprehended that requirements engineers, architects, designers, developers, testers, operators, and users all play essential roles in producing and maintaining secure systems.

2.3 Code-Level Hardening Description

After a short lecture introducing foundational concepts including the SDLC, system-level hardening, and code-level hardening, students examined the vulnerable application "Madam Zora" (Zora). The instructor designed Zora, a custom web application, specifically for the Code-Level and System-Level Hardening exercises.

Students tested four vulnerabilities exhibited by the web application: 1) Cross-Site Scripting (XSS), 2) Structured Query Language (SQL) Injection, 3) Command Injection, and 4) File Upload. Next, the students patched the associated flawed Perl and PHP Hypertext Preprocessor (PHP) code. Finally, students retested the vulnerabilities to ensure the coding changes fixed the vulnerabilities. The secure programming practice take away from this exercise was to filter all user-influenced input and output for web applications. To accommodate students of varying programming expertise, the instructor led the testing, patching, and retesting of the four vulnerabilities, conducting frequent checks for understanding and inviting questions.

2.3.1 Cross-Site Scripting

XSS occurs when users manipulate web application input to execute client-side commands on a system. A well-known test for XSS entails inputting the JavaScript *alert* command into a web application text field. Students tested Zora text fields by entering the following command:

```
<script language='javascript'>alert('Zora!');</script>
```

If an alert box containing *Zora!* appeared, students established the XSS vulnerability of the underlying script. The Zora XSS vulnerability existed in a PHP file that echoed unfiltered user input back to the screen. To eliminate the vulnerability, students filtered the input using the PHP *htmlentities* function and retested the code. The *htmlentities* function translates certain ambiguous characters into their corresponding character entity references. For example the '<' character becomes '*&lt;*'. This prevents inputted JavaScript commands from being evaluated. The vulnerable Zora code outputting unfiltered user input is shown below:

```
$unfiltered['input'] = ($_GET['user_input']);
<?php echo $unfiltered['input']?>
```

The fixed Zora code outputting input filtered with *htmlentities* is as follows:

```
$unfiltered['input'] = ($_GET['user_input']);
$filtered['input']=htmlentities($unfiltered['input']);
<?php echo $filtered['input']?>
```

2.3.2 Command Injection

Command injection occurs when users manipulate input to execute terminal commands. The Zora command injection vulnerability stemmed from a line in a Perl file using the *system* function in conjunction with user input saved into the *$username* variable:

```
system "cat ./${username}";
```

Students executed desired commands with the privileges of the Apache2 web server process by inputting a semicolon terminator followed by the command of choice into the username field on a login web page. For example, if students inputted a semicolon followed by the *print working directory* command (*;pwd*), the subsequent screen listed the command output. The output in this case was the current working directory (*/home/zora*).

Students implemented multiple strategies to fix the vulnerability. Some students filtered the input saved into the *$username* variable to remove characters typically not found in usernames such as semicolons and slashes. Other students eliminated the *$username* variable from the *system* command while adding code to preserve the original functionality of the web server.

2.3.3 SQL Injection

SQL injection vulnerabilities occur when attackers craft input data to cause SQL statements to be executed in ways unanticipated by the original programmer. A common method of testing for the vulnerability involves inputting a value that causes *WHERE* statements to evaluate to true.

In the Madam Zora web application, a vulnerable PHP script included the following statement:

```
$sql = "SELECT fortune FROM fortune_table
        WHERE spirit_name='{$unfiltered['name']}'";
```

If a user inputted a value causing the overall statement to always evaluate to true, such as:

```
' or 'x'='x
```

the executed command would be:

```
$sql = "SELECT fortune FROM fortune_table
        WHERE spirit_name='' or 'x'='x'";
```

Since *'x'='x'* is always true, the SQL server displays all fortune records in *fortune_table*. Students filtered the input with the built-in PHP *mysql_real_escape_string* function, which strips out special characters such as quotes. The fixed Zora PHP script included the following code:

```
$filtered['name']=mysql_real_escape_string
        ($unfiltered['name']);
```

Students eliminated the vulnerability by inserting the filtered value *$filtered['name']* in place of the unfiltered value *$unfiltered['name']* in the *SELECT* statement.

2.3.4 File Upload

The file upload vulnerability occurs when user input influences the creation, naming, and content of files. In the exercise, students used a web form containing multiple fields to upload malicious code and save it to a file name of their choice. A Zora Perl file contained the vulnerable code:

```
open (FILE, "> ./${zora}") or $er = 1;
        if ($er == 0) { print(FILE "$fortune");}
```

where *$zora* and *$fortune* are set by user input fields. The Perl file saved the content of the *$fortune* variable as a file with the name of *$zora*. Due to the large amount of text users could save into the *$fortune* variable, it was possible to upload the source code for a small backdoor to the Zora server. Although the web server saved the file without execute permission, students leveraged the command injection flaw (discussed in section 2.3.2) to issue commands to change permissions on, compile, and run the uploaded backdoor.

2.4 System-Level Hardening Description

During the System-Level Hardening exercise, students identified and patched 22 system-level vulnerabilities exhibited by the Madam Zora VMware image. The Zora image distributed to students ran standard File Transfer Protocol (FTP), Telnet, Hypertext Transfer Protocol (HTTP), Internet Printing Protocol (IPP), and MySQL servers and non-standard persistent Netcat backdoors on seven ports in the 4000-6000 range. Students possessed a list of operational requirements to provide while patching the system: 1) Apache2 server running on standard port 80, 2) MySQL database server running on standard port 3306, 3) Netcat and Cron installed and operational, 4) Working sudo account named *zora*, 5) Secure mechanism to command and control the system, and 6) Secure mechanism to copy files to and from the system.

Students received a blank checklist for the 22 vulnerabilities. Vulnerabilities included persistent backdoors, file permission and account misconfigurations, and the use of outmoded, unencrypted services such as Telnet. As students identified and patched vulnerabilities, the instructor checked the vulnerabilities off of each student list. At the conclusion of the exercise, the instructor discussed all 22 vulnerabilities with the students.

3 Reverse Engineering

3.1 Background

Reverse engineering involves deciphering the construction and function of a system by studying its observable structure, characteristics, and behavior. Performing reverse engineering requires familiarity with computer architecture, operating systems, programming in assembly and high-level languages, and tools such as disassemblers, virtual environments, network monitors, and system monitors.

There exist two major categories of reverse engineering activities, dynamic analysis and static analysis. Dynamic analysis entails performing a behavior-based study of an executing binary. Dynamic analysis activities include executing the analyzed binary in a controlled environment, viewing file, process, and registry key modifications, and detecting opened ports and established network connections. Static analysis involves studying a binary code listing or artifact without executing the binary. Static analysis activities consist of dissecting a written code representation of a binary by identifying functions, parameters, and arguments and tracing its control flow.

Studying reverse engineering in nine hours by students of varying backgrounds posed a significant risk of incurring student alienation or despair. Although all students had experience programming in a high level language, many had limited exposure to assembly language and computer architecture. Conducting an introductory lecture and smaller supporting exercises before the main exercises reduced this risk. Managing student expectations regarding mastery of the material and providing numerous hints, code comments, and intensive instructor assistance also helped.

3.2 Exercise Goals

The instructor stressed to students the impossibility of becoming expert reverse engineers in nine hours. Instead, the instructor presented the main goals as: 1) Reinforcing foundational computer science and engineering topics such as computer architecture, programming, networking, and operating systems, 2) Exposing students to the range and complexity of reverse engineering activities, and 3) Inspiring further study in the field of reverse engineering.

3.3 Reverse Engineering Preparatory Lecture

The three-hour preparatory lecture covered reverse engineering concepts such as static and dynamic analysis of code, the computer stack, registers, core x86 assembly language instructions, malicious code, packers, and a set of reverse engineering tools. The instructor provided students with a Windows VMware image containing tools and sample code for analysis. After introduction to each reverse engineering tool, students accomplished a small, but meaningful task using the tool. For instance, after learning about hex editors, students used the XVI32 hex editor program to view the strings embedded in a binary.

After the lecture, students completed dynamic and static analysis activities. The activities consisted of thirty-minute warm-ups intended to prepare students for the three-hour dynamic and static analysis exercises. For the dynamic analysis activity, students applied tools such as hex editors, packers, file system monitors, process monitors, and network monitors to analyze the behavior of a packed binary that output a string to the terminal when executed. The static analysis activity involved determining the purpose, parameters, and return value of an assembly language function. At the conclusion of each activity, the instructor reviewed the solution with the students.

3.4 Dynamic Exercise Description

During the three-hour Dynamic Reverse Engineering Exercise, students analyzed three malicious code binaries. Establishing selection criteria for the malware to be analyzed proved crucial for exercise success. The instructor downloaded malicious code from *offensivecomputing.net*, which provided a name, classification, hash values, and a brief description of available malware samples. The instructor obtained and tested over thirty samples before making the final selections.

Selection criteria included the following: 1) Malware must consist of three samples clearly representing different malware categories (e.g. worm, bot, rootkit), 2) Malware must be robust and reliable, 3) Due to time constraints, malware must be unpackable without applying complex techniques such as dumping the original code from memory with a debugger, 4) Malware must exhibit overt, complex, and varied behavior, such as file system changes, persistence or defensive actions, data collection, and propagation, and 5) Malware must immediately demonstrate effects.

After applying the selection criteria, the instructor chose three samples including a keylogger, a trojan, and a bot. All samples exhibited the desired levels of robustness and reliability, used no packing technology or proved easy to unpack, and demonstrated immediate and varied effects upon installation.

Because the exercise required executing malware, students performed the following tasks: 1) Divided laptops into research machines and analysis machines, 2) Copied the VMware image containing the malware and reverse engineering tools to the analysis machines, 3) Connected the analysis machines to the provided isolated switches (not connected to the Internet), and 4) Connected the research machines to the network providing Internet connectivity. As all student laptops were identically configured and patched and the malware either didn't spread over networks or took advantage of well-known, patched vulnerabilities, spreading from the VMware images to student analysis machines did not present a concern.

After students setup their analysis and research machines and started the VMware image containing the malware samples and analysis tools, they completed an analysis worksheet for each of the three malware samples. The worksheet included fields such as functions and libraries referenced in the binary, files and registry key changes, command and control method, malware defenses, and remediation recommendation.

3.5 Static Exercise Description

During the three-hour static analysis exercise, students studied a four-page assembly language listing and packet capture of the Structured Query Language (SQL) Slammer worm. Slammer, also known as Sapphire, is a memory-resident Internet worm that uses a buffer overflow exploit to take control of hosts running a vulnerable version of SQL Server. Slammer propagated by using its victim hosts to generate and send single User Datagram Protocol (UDP) packets containing attack code to random Internet Protocol (IP) addresses. The instructor selected Slammer for the static analysis exercise due to its short assembly code listing and relative simplicity.

To account for the varying backgrounds and technical skill-sets of the students, the instructor split the code into logical, manageable segments, and provided students with research questions for each segment. After students analyzed each code segment and answered the associated questions, the class reconvened to discuss the questions and the instructor checked for understanding before proceeding to the next segment.

Due to time constraints, the instructor provided code comments for instructions requiring tracking values on the stack or deciphering with a debugger. In addition, the instructor provided the identity and value of the stack entries referenced by the code. For example, the instructor provided the comment *Push address of sock_addr structure* for the instruction *push eax*. The instructor held a final discussion for lessons learned at the end of the exercise.

4 Detect and Defeat

4.1 Background

The Detect and Defeat exercise introduced students to technologies that detect cyber threats within a network or on a host computer, how to defeat those threats, and how to architect defense-in-depth systems. Covered technologies included firewalls, network intrusion detection systems (NIDS), and host intrusion detection systems (HIDS).

A firewall is a device that filters network traffic at one or more of the seven layers of the Open Systems Interconnection (OSI) networking stack. There are many different types of firewalls including network packet filters, application proxies, and host system firewalls. At each operating level a firewall will permit or deny traffic based on a set of rules or policy.

Intrusion detection systems come in two different varieties, NIDS and HIDS. The common NIDS is a passive device that examines the ingress and egress traffic of a network and flags suspicious or malicious traffic based on a set of signature rules. A HIDS monitors the integrity of important files, binaries and executables on a host machine and alerts on suspicious of malicious actions. Regarding detection techniques, this exercise focused on signature-based IDSs and did not address anomaly-based IDSs.

4.2 Exercise Goals

The goals of the Detect and Defeat exercise were to teach students common network security practices, introduce them to popular network defense tools, and to use these tools to identify and defeat threats within a network. The exercise used tools for the Linux operating system to provide students with additional Linux experience.

4.3 Description

The Detect and Defeat exercise introduced students to firewalls, NIDS, and HIDS for the Linux operating system. For practice in configuring firewalls and network intrusion detection systems, the Linux operating system provides an optimal environment for students to learn how operating systems execute firewall rules and process intrusion detection signatures. The two-part exercise consisted of the following: 1) Introductory tutorials for the Linux application iptables, the Snort IDS, and the host Advanced Intrusion Detection Environment (AIDE), 2) Hands-on practice using the tools in a live environment. In part one of the exercise, the instructor asked students to implement the requirements outlined in each tutorial on a practice Linux virtual machine (VM). For part two, the instructor gave students two VMs to use in a live scenario. The two VMs consisted of an aggressor VM and a defender VM. The defender VM had iptables, Snort, and AIDE installed but not configured. The aggressor VM periodically executed a set of scripts that sent data to open ports on the defender VM, triggered Snort IDS signatures, and created alerts within AIDE. The objective for students was to correctly configure the defender VM to detect and defeat the aggressor VM actions. The students demonstrated to the instructor the following items: 1) Used iptables to filter incoming traffic and allowed traffic on port 80 (HTTP), 443 Secure Sockets Layer (SSL), and 22 Secure Shell (SSH), 2) Configured Snort to recognize and alert against the traffic coming from the aggressor VM, and 3) Established an AIDE hash database for the files in the /etc directory and demonstrated an alert on a file in the /etc directory.

5 Digital Forensics

5.1 Background

Computer forensics is the discipline that combines elements of law and computer science to collect and analyze data from computer systems, networks, wireless communications, and storage devices in a way that is admissible as evidence in a court of law [1]. The traditional forensics methodology consists of acquiring evidence without altering the original media, analyzing the data to produce the necessary evidence, and proving the authenticity of the evidence.

Ten years ago computer forensic practices mainly entailed examining computer hard drives after a crime took place. Recently, live response forensics, network forensics, and rapid evidence gathering of data have been included under the computer forensics field of study. In digital forensics there are two basic data types, persistent data and volatile data. Volatile data is the transient data found on a digital system that exists while the computer is powered on. Persistent data is the information stored on a hard drive. This exercise concentrated primarily on the technical aspects of performing digital forensics and did not include in-depth coverage of the legal aspects of forensics.

5.2 Exercise Goals

The goal of the Digital Forensics exercise was to enable students to analyze digital media using established digital forensic techniques. The goal included having students analyze a piece of digital media and gather digital evidence using established methods that will stand up in a court of law.

5.3 Description

The digital forensics exercise focused on the following scenario:

Agent Johnson of the Office of Special Investigations (OSI) is in the middle of an investigation against several groups connected to the mafia. Agent Johnson just arrived to a murder scene of someone connected to the mafia group. Agent Johnson is in charge of digital evidence collection and has found a running desktop at the scene. He must: 1) Perform a live forensic investigation, 2) Conduct a post-mortem investigation, and 3) Build a report and present the findings to the instructor. Recovered evidence must include an IM conversation between mafia members, photos of a weapons exchange, and the passwords to a user account with important information.

The two-part exercise included a live response investigation and a traditional forensic analysis of the physical hard drive. The exercise setup consisted of a Windows XP SP1 virtual machine representing the "running desktop" and a USB external hard drive representing the physical hard drive of the desktop. Students used a combination of the following forensics tools to accomplish exercise objectives: 1) Helix Forensics Live Linux CD, 2) WinHex, 3) md5sum, 4) Linux 'DD' command, and 5) FTK Imager.

Throughout the exercise the instructor checked to ensure students performed their forensic analysis using sound techniques. In a forensic investigation the integrity of the original evidence is crucial. The instructor required students to preserve the chain of command while interacting with the digital evidence. This included performing the following steps to accomplish the exercise: 1) Used statically linked tools from the Helix Linux CD to record all the running processes, applications, logged in users and all pertinent transient data of the running desktop, 2) Calculated an md5sum hash of the physical hard drive in a read-only manner, 3) Created a forensic image of the physical hard drive, 4) Took an md5sum hash of the forensic image and compared it to the original media, 5) Used a hex editor like WinHex or forensic analysis tool to examine the forensic image, and 6) Thoroughly documented the process and all the evidence findings. At the conclusion of the exercise, each team presented their findings to the instructor.

6 Wireless Access Point Treasure Hunt

6.1 Background

The capstone DefEX was the Wireless Access Point (WAP) Treasure Hunt. Students engaged in wardriving to locate a series of time-constrained challenges distributed across a small city. Wardriving entails utilizing a vehicle and a portable computing device to search a geographic area for a wireless network. Students used network detector software such as NetStumbler and Kismet to locate WAPs of interest. The Treasure Hunt challenges located at the WAPs included completing a cryptography problem and circuit worksheet, discovering a password vulnerability, conducting a forensics analysis on a thumb drive, bypassing authentication on a website, and identifying file, database, and mail server misconfigurations.

6.2 Exercise Goals

The overarching objective of the WAP Treasure Hunt was to test the leadership and decision making skills of the students in a time-critical environment. The Treasure Hunt provided a cumulative team-based challenge for students that reinforced select topics from other exercises.

6.3 Description

In the WAP Treasure Hunt exercise, teams searched for a series of WAPs temporarily dispersed by instructors around a city. At each WAP site, students completed a challenge to receive the map leading to the next WAP site. Students earned points for completing challenges within a given time deadline. If the deadline passed, instructors gave students the next map but did not award any points. The team accumulating the most points won the exercise. The exercise culminated with a final challenge at a bowling alley where students found the "treasure", a package of silver and gold chocolate candies and a surprise pizza and bowling party.

The key organizational activities included constructing the order of site visits, creating the map scrolls, and setting rules of engagement for the exercise.

1) Construct Challenge Site Visit Order: The instructor generated the order of the challenge sites each team visited. Although a unique path for each team was desirable to minimize challenge site congestion, due to the number of teams and sites some redundancy was unavoidable.

2) Create Map Scrolls: For each site, the instructor created a satellite map of the pertinent city area, delineated a reasonably sized search area incorporating the challenge site with a red box, and added red text indicating the SSID for the WAP at the site. The instructor rolled the maps and tied them with different colored ribbons to create scrolls. Each team had a unique ribbon color, which helped site attendants provide teams with the correct scrolls.

3) Set Rules of Engagement: For safety reasons, the instructor assigned each team a driver, who was not allowed to assist students with locating the WAPs or completing the site challenges. Instructors prohibited students from connecting to any WAPs other than the ones involved in the exercise and required them to gain permission from their driver before connecting to any WAP. Instructors also disallowed using any equipment other than student laptops, such as external antennae.

The challenges included completing a cryptography problem and circuit worksheet, examining WAP password authentication, conducting a forensics analysis on a thumb drive, investigating website authentication, and identifying file, database, and mail server misconfigurations.

1) Initial Challenge: Student teams gathered at an initial site for an exercise briefing, to review the rules of engagement, and to complete the first challenge, a cryptography problem and a circuit diagram worksheet. After reviewing the exercise purpose and rules, each team received six map scrolls tied with six different colored ribbons. The cryptography and circuit diagram challenge answer resulted in one of the six colors. If students solved the problem correctly and chose the right scroll, they proceeded to the next challenge. If the team calculated an incorrect color and chose a wrong scroll, the map led to a time penalty site where an instructor required students to answer a set of cyber operations questions before giving them the map to the next challenge.

2) WAP Configuration: When conducting vulnerability assessments, students often overlook supporting networking infrastructure such as switches, routers, and wireless equipment. The WAP configuration challenge tested this common oversight while distracting students with a decoy Nepenthes honeypot. In this challenge, the password-protected WAP web interface had an easily guessed username and password. The main WAP configuration screen listed a passphrase that students could exchange for the next map.

3) Forensics: Each student team analyzed a USB thumb drive using sound digital forensic techniques. Instructors set up the challenge site as an "investigation scene", placing a thumb drive containing the evidence and a decoy laptop workstation at the scene. Instructors formatted the thumb drive with a persistent version of Ubuntu Linux that used an ext3 file system and embedded the passphrase for the next map in the slack space of the file system. To find the passphrase, each team created a forensic image of the thumb drive and examined the image in a hex editor or a piece of forensic analysis software.

4) Website Authentication Challenge: The Website Authentication Challenge required students to complete a six-level password challenge. Inputting any values for the username and password on the first level advanced students to the second level. The passwords for levels two through four resided in a comment, an image tag, and a hidden form field in the html source code respectively. Inputting any values into the username and password fields on the fifth level provided a list of nine file names from file1.txt through file9.txt. Inputting each file name into a field titled *Magic Phrase* gave students Morse code representations for letters of the password and a Morse code key. The sixth level gave the hint *l6pass.txt*. Students could obtain the password by inputting *http://<ip address>/l6pass.txt* into the browser window. Completing the sixth level gave students a passphrase to exchange for the next map.

5) FTP, MySQL, and Simple Mail Transfer Protocol (SMTP) Configuration: Students explored multiple machines running FTP, MySQL, and SMTP servers to derive a passphrase and earned the next map. Students retrieved a MySQL username and password from an FTP server with anonymous login enabled. Students used the username and password to log into a MySQL server and retrieve the first part of the map passphrase from a table. The instructor hid the second part of the map passphrase in the banner of a SMTP server. Students created a Transport Control Protocol (TCP) connection with the SMTP server to retrieve the banner.

7 Conclusion

Successful completion of DefEX required Computer Science, Computer Engineering, Electrical Engineering, Physics, and Math undergraduates to exercise a broad range of skills. Technical skills included interpreting Assembly language, analyzing and patching Perl and PHP scripts, finding and eliminating persistent backdoors, configuring intrusion detection systems and firewalls to repel known attacks, and conducting live and post-mortem digital forensics analyses after an attack. Exercises also required leadership, teamwork, executing under pressure, and problem-solving skills.

Providing a diverse group of students with a common set of foundational knowledge and finding the balance between enabling participation of novice students and generating problems complex enough to challenge experienced students posed the major curriculum design risks. Instructors reduced the risks by administering a technical survey, requiring students to complete a set of fundamental exercises, and assigning balanced student teams. As a result, student teams successfully completed all of the exercises.

8 Acknowledgements

The authors wish to thank Dr. Kamal Jabbour, ST, Air Force Senior Scientist for Information Assurance, Regina Recco, Dr. Sarah Muccio, Lt Col (ret) Ken Chaisson, and Thomas Vestal for their diligent support during the development and execution of these exercises.

9 References

[1] US-CERT, "Computer Forensics," [Online]. Available: http://www.us-cert.gov/reading_room/forensics.pdf.