Proceedings of the 13th Colloquium for Information Systems Security Education
University of Alaska, Fairbanks
Seattle, WA June 1 - 3, 2009

The Design and Lessons of the ASCENT Security Teaching Lab

Jun-Won Ho, Nayantara Mallesh, and Matthew Wright
[jun-won.ho, nayantara.mallesh]@mavs.uta.edu, mwright@cse.uta.edu

Abstract – The ASCENT security teaching lab provides both graduate students and undergraduate students at the University of Texas at Arlington with an opportunity to get hands-on education in both attack and defense. We developed the lab over a two-year period and have learned valuable lessons from its development and use by students. In this article, we examine the design of the lab infrastructure and the use of laptops, the design of exercises, the role of virtualization, and the use of capture the flag exercises. We also discuss the use of our labs in classes conducted at a large software company in India.

Index terms – Lab exercises, lab design, international experiences

I. Introduction

ASCENT – the Alliance for Secure Computing Education in North Texas – is a joint NSF-funded project between the University of North Texas (UNT), the University of Texas at Arlington (UTA), and the University of North Carolina at Greensboro (UNCG). One of the main goals of ASCENT is to provide laboratory equipment for the participating schools and develop exercises for students in security.

As one of the participating schools, UTA has procured equipment and put together a lab – the ASCENT security teaching lab (or ASCENT lab) – for use in three courses. We designed an architecture for the lab, including virtualization based on Xen, a NFS file server, laptops, workstations, and networking. For students to make use of the lab, we have created a set of hands-on exercises that are appropriate for seniors and graduate students in computer science. The exercises allow students to develop and test both attack and defense both for hosts and on the network. We have put these exercises into a workbook that is available at our website: http://isec.uta.edu/ascent/

iSec: The Information Security Lab
University of Texas at Arlington

This material is based upon work supported by the National Science Foundation under Grant No. 0621280. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

The lab has now been used, at various stages of development, for three courses held a total of five times. Additionally, we brought our materials to India and built a similar lab in a classroom in a software company, with interesting results. Through these classes, we have gained valuable experience in designing and building both the lab and the exercises. In this paper, we describe our lessons. In particular, we focus on the difference between our hopes and expectations and our actual experiences. While many of our goals have been achieved, we have also faced some challenges.

We begin in Section II with an overview of the lab design and the exercises we have developed. In Section III, we discuss our experiences with multi-institution capture the flag exercises, including their valuable role in helping us create lab exercises. Section IV describes the implementation of the lab in a industry classroom setting in India, which helped us to learn more about the strengths and weaknesses of our lab design. We discuss in Section V the design choices for both the lab and the exercises and lessons learned. Section VI discusses ethical considerations, Section VII reviews related work, and Section VIII concludes.

II. ASCENT Security Teaching Lab

In this section, we first present how we established and operated ASCENT security teaching lab and then describe the content of the lab exercises.

[1] *Lab Setup and Operation*

The ASCENT lab consists of 5 Dell desktop computers and 17 Lenovo laptops, three switches, two Cisco XX routers, and VPN boxes. The lab machines are, for most exercises, disconnected from the Internet. While we don't believe that any of our exercises could be a genuine threat to other networks, this helps ensure that no incidents escape the lab environment. We utilized the Network File System (NFS) for lab exercise management. Specifically, we configured one desktop machine as an NFS server and the other desktops and laptops as NFS clients. All lab exercise materials are placed in a designated directory on the NFS server and this designated directory is mounted

by all client machines. Thus, lab materials on the server can be accessed by any client machine.

We gain the following benefits from employing NFS. First, we do not need to assign each student a designated client machine because all lab hand-ins are synchronized on the server regardless of which client machines are used for lab tasks. This is particularly useful for students to do their lab exercises over multiple sessions, as they do not need to remember which client machines they were using before and they can join different lab sessions when necessary. Second, we do not need to access each client machine for lab grading, nor have the students download and submit their work, but only need to access a single server. This helps reduce grading time substantially.

In the Fall 2007 semester, 45 students registered in the lab sections. Since we only had 13 client machines at that time, we employed Xen virtualization systems to enable all students to use the labs at the same time. Xen enables multiple operating systems to run on a single computer. We installed multiple Fedora 7 operating systems on each computer with the aid of Xen. We successfully operated the lab sections, thanks to virtualization, even though we had fewer computers than the number of students.

Since we split the class into undergraduate and graduate sections for Fall 2008, and added more client machines, we had fewer students at one time and we did not need to employ Xen.

[2] *Lab Exercises*

We developed lab exercises for students to gain hands-on experiences in the system and network security fields. Before doing lab exercises, students were required to do pre-labs. The main purpose for pre-labs is to help students become familiar with lab contents and not waste as much time getting up to speed in the lab.

Now we present a brief summary of each lab exercise, evaluate how students performed, and discuss how we taught defenses for attack-based exercises.

- Buffer Overflow Attack

The objective of this lab is to find buffer overflow vulnerabilities in a simple FTP server program and exploit these vulnerabilities. The first part of the lab has the students simply crash the FTP server, while the second part of the lab has students execute shell code. The FTP server helps the students by printing out stack addresses and content, similar to the output shown in Figure 1.

```
[dliu@omega stack]$ ./a.out
BFFFE560:3020100
BFFFE564:7060504
BFFFE568:B75F0908
BFFFE56C:0
BFFFE570:3020100
BFFFE574:8049504
BFFFE578:BFFFE588
BFFFE57C:8048265
BFFFE580:0
BFFFE584:0
BFFFE588:BFFFE5A8
BFFFE58C:80483F6
BFFFE590:1
BFFFE594:2
BFFFE598:3
BFFFE59C:859D78
BFFFE5A0:67E020
BFFFE5A4:80483FC
BFFFE5A8:BFFFE608
BFFFE5AC:73979A
```

Figure 1: Example stack output

Since Fedora 7 uses stack randomization, the exploit would not succeed without intervention. We used this command (as root) to turn randomization off:

```
> sysctl -w kernel.randomize_va_space=0
```

Most students successfully crashed the FTP server program in both the Fall 2007 and Fall 2008 semesters. However, 20% of students failed to spawn a shell in the Fall 2007 semester. They got stuck in the conversion of the *exec* system call to the assembly executable codes. Also, some students had trouble with putting the executable code into the right place in the stack of the vulnerable FTP server program. The lab teaching assistant helped them learn how to insert the executable codes into the stack by using NOPs.

In the Fall 2008 semester, we gave the assembly executable code to students. Given the executable shell code, most students were able to successfully place the code into the stack of FTP server program and spawn a shell.

The lab exercise is quite challenging, requiring many hours of work for some students, even though we cover the topic extensively in class and provide a number of important components in the pre-lab.

For defenses, we spend most of one lecture describing some of the newer defenses that have been incorporated into modern operating systems, such as Stack Guard and stack and heap randomization. These defenses would be much harder to appreciate and understand without a detailed understanding of buffer overflow attacks. We

believe that a lab on defense would not be as insightful, but we are planning to have the students verify that their exploit fails when stack randomization is turned on.

- Command Injection Attack

The objective of this lab is to find a command injection vulnerability in a simple file server and exploit it to launch a command injection attack. Many students had difficulty in finding the command injection vulnerability. This lab turned out to be more difficult for the students than we thought. The lab teaching assistant had to help students learn the basic concept of a command injection attack we covered in class and locate the part related to vulnerability in the file server program.

This suggests that the lab was somewhat successful in helping us fill a gap in the students' knowledge. They learned a key course concept only by experience in the lab. At the same time, we would hope that students come prepared for the lab, which did not happen. We discuss possible solutions for this in Section V.

For defense, we show students the basics of building secure software. However, we do not go into great depth, as we offer a separate course just on building secure software. Admittedly, this lab is not aligned very clearly to the course objectives. However, it is important as preparation for the CTF exercise.

- Virus Creation

The objective of this lab is to learn how to create virus. We built the lab based on the *Jingle Bell* virus by Amit Singh. The virus source code is available at http://www.kernelthread.com/publications/security/vunix.html. The virus is written in C and simply prepends itself to any program files given as command line inputs. We developed a pre-lab that has students work with file input and output to develop the basic components of the lab. Specifically, we first have the students prepend a file to another. Second, we have the students modify a program to read in only half of a file, rather then the whole file, and output to a new executable file. This prepares students for the system programming required in the lab.

The lab itself consists of two parts. The first part is to prepend the viral code to the executable code. The second part is to infect other executables with infected code. Although we gave step-by-step instructions to students, some students did not succeed in creating the virus. They understood the basic concept of how to create virus, but failed to implement the virus program correctly. From this, we realized that there could be subtle issues in implementing virus program. We tried to fix a couple of subtle issues that students had experienced.

For defense, we discuss in lecture how virus signature detection works. It would be an interesting extension of the lab to have students build a signature detector that can find their virus.

- Securing a Linux Host

The objective of this lab is to learn important techniques used in securing a Linux host system. All students understood the main requirements of this lab and gained experience in how to use *setuid* and *setgid* and the basic principles of password management in Linux.

- Creating an Application-Level Rootkit

The objective of this lab is to create backdoors in the victim server and hide it from the victim server. Students can easily create backdoors since they only need to know how to use the *dup* system call. They are required to modify the *ps* program in order to hide backdoor programs from system. Students who had difficulty in modifying *ps* program code usually were not familiar with handling linked lists, which is a core part of the modification.

Students commented that this lab, while interesting, did not require them to understand fully what was happening. For defense, we have the students detect the modified *ps* program by using the tripwire detection tool. We also combine this lab with the previous lab to give a link between what system security does to prevent attacks.

- Intrusion Detection and Access Control List

The objective of this lab is to become familiar with intrusion detection tools and how to configure an access control list. Students learned how to use snort to detect intrusion attempts, TCP wrapper to set the access control list, and Nmap to scan the system ports. This lab is very easy for students and does not challenge them.

- Nessus and Metasploit

The objective of this lab is to learn how to use the *Nessus* and *Metasploit* penetration testing tools. Specifically, students are required to identify the vulnerable server running on the victim machine by using Nessus and then gain full control of the victim machine by exploiting the vulnerable server with Metasploit. All students handled these tools well and successfully gained full control of the victim machine. A buffer overflow vulnerability of the *Icecast* program was exploited by Metasploit.

This exercise is very easy for students, and they do not list it among the interesting parts of the class in post-class

surveys. For defense, we combine this lab with the previous one.

- Cracking WEP

The objective of this lab is to crack WEP by using freely-available cracking tool. All students successfully obtained the shared secret key between the access point and the wireless station. Since a dictionary attack technique is used in cracking tool, the cracking time was determined by the amount of traffic between wireless station and the access point. Namely, the more traffic is generated, the faster WEP cracking is done. We used multiple ping commands to generate a large amount of traffic between the wireless station and access point.

This exercise is very easy for the students and we used it only once. We are currently considering the development of a more complex version of the lab in which students, primarily graduate students taking an advanced security class, would develop and test part of the cryptanalytic attack against the WEP protocol, based on the work of Stubblefield et al. [1]. Alternatively, students could build an attack system based on a password-cracking dictionary attack. For defense, we could have advanced students implement and test the WPA protocol.

III. Capture The Flag

We organized a Capture The Flag (CTF) competition together with UNCG and UNT for the past three years. One team per school participated in the CTF. As shown in Figure 1, the three teams are connected through a Virtual Private Network (VPN). Each team runs vulnerable web, authentication, and file servers on a virtual machine. We used services developed by Vigna et al. for the well-known UCSB CTF contest (see http://www.cs.ucsb.edu/~vigna/CTF/). Secret data flags are planted in the services. Each team is required to keep services running, find the vulnerabilities and fix their service, and attack other teams' servers to get flags. In the Fall 2007 competition, UNT obtained the largest number of flags among the three teams. In Fall 2008 competition, UNT could not participate in the contest but UNCG obtained more flags than UTA.

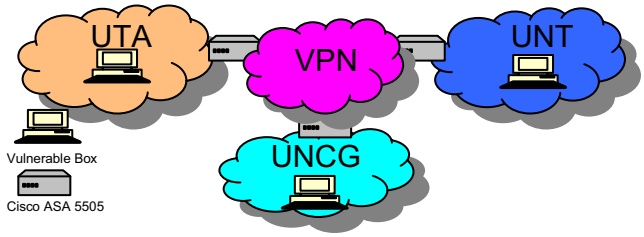

Figure 2: CTF contest configuration diagram.

In both competitions, the UTA team only obtained the flags related to the command injection attack that is taught in our lab exercise. Since only six hours are given to each team, the UTA team had difficulty in capturing the flags by exploiting unknown vulnerabilities within the limited time. At the same time, they failed to consider direct attacks, such as password guessing. This allowed the other teams to access their flags.

For the 2008 contest, UTA students were given specific preparation in the form of an open-ended lab exercise to identify and report on software vulnerabilities in the servers. This had limited success; students learned something about the possible vulnerabilities but still had trouble with exploitation. The exercise showed that students need specific training with each vulnerability to be able to do well in the CTF. Again, the students were most successful with command injection attack due to their specific experience with that attack type.

In theory, the CTF is a great tool for teaching students hands on lessons about security. Our experience, however, has been mixed at best. Since we are teaching students about a variety of security topics, finding software vulnerabilities can only make up a small part of the class. Finding a vulnerability is a matter of either prior experience or especially intelligent students playing with the software. The former requires training, such as with one lab for each type of vulnerability, while the latter requires both very bright students, time (more than the contest allows) and some serendipity.

Furthermore, the CTF format itself demands training for the students. They need to understand how to function as a team, how to divide tasks, and what tasks need to be done. For example, only through reading about other CTF teams on the web did we discover that teams should utilize a "human IDS" to watch the other team's activities on their network. And only through the experience of having the system broken into did we realize that system security mattered as much as software security.

With enough time, instructors could craft CTF exercises that specifically covered what was done in other labs. This would allow them to have the benefits of a CTF, namely the excitement of competition and an open-ended experience, without the frustration that can come from a having a completely open-ended experience with strict time limits.

Alternatively, the instructor could redesign the course to prepare students for the CTF. This would be a more dramatic change and might substitute excitement for learning objectives. Nevertheless, the challenge of going against other teams could be very motivating to students for their practical study. We might consider offering this as an option for motivated students.

IV. ASCENT IN INDIA

In the summer of 2008, we were invited to conduct a three-week course at Infosys Technologies, Mysore, India. As part of the course, we conducted both theory and lab sessions. The lab sessions were based on the exercises discussed in Section II. Due to time limitations we were able to offer only a subset of the lab exercises usually done at UTA. At Infosys, we covered buffer overflows, virus creation, application-level rootkits, and Metasploit.

The course itself was only 14 work days long. It included a morning lecture session, followed by individual effort on lab exercises. Generally, this meant that students would put into practice what they learned right away.

About 14 Infosys employees registered for the course. Of these, a few were fluent with programming in Unix and C. However, the instructions and structure of the labs allowed the students to understand the idea behind the lab rather than focus solely on the technical details. With some individual assistance from the TA, most students were able to successfully complete the labs. It seemed to help that students had access to computers during lecture and could perform basic exercises on the spot.

As done for the Fall 2008 students at UTA, for the Buffer Overflow lab, we provided the executable to the Infosys students. All students could successfully exploit the vulnerable program and crash it. However, fewer students managed to spawn a shell. The time given to the Infoysys students was less than the usual time given during a regular semester at UTA. It appears that given more time, all students would have been able to successfully spawn a shell and complete the lab objectives.

For the virus creation lab, we fixed the code containing the subtle issues mentioned in the previous section. As a result, most of the class was able to finish the lab successfully at Infosys. During the second and third weeks of the course, the students got better at handling the lab assignments. Though a scaled down version of the labs was offered to these students, the learning rate got better for the later labs. The success rate for the rootkit and Metasploit lab exercises were similar to the students at UTA.

It helped that the theory behind the labs was covered shortly before the students were given the lab exercises. Knowing the theory allowed the students to have an overall idea of the goal of the exploit and a general idea of how it was going to be done. Thereafter, the students could concentrate on how to implement the code for each of the exploits.

Feedback from students showed that the students better appreciated the material covered in theory after they had been through the lab sessions. It gave them an opportunity to see first hand how simple exploits work.

V. DISCUSSION

In this section, we describe lessons learned from the use of our lab and the lab exercises described in Section II.

First, we found that the right level of challenge is important to get students to engage with the exercises and learn from them. For example, we found that students who completed the buffer overflow lab understood the concepts involved in the attack very well. Students who struggled with the lab typically struggled with the theoretical concepts. Easy labs do not give students much excitement upon completion and do not engage their minds in absorbing the material. We are actively seeking ways to make labs that have adequately challenging steps, while keeping the challenge in line with students' abilities and what has been covered in lecture.

Similarly, single-step activities are very limiting in the kind of material the students can learn. For example, creating a simple rootkit, given the main pieces of the program, does not give a complete picture of what a rootkit is and what it can do. In future labs, we intend to include multiple steps that each contribute to the student's understanding. For example, we can give the students the original *ps* program and have the students design the rootkit components, determine how to modify *ps*, and then perform the modification.

Another lesson we have learned is that students benefit from extensive pre-lab assignments. Since students enter the class with varying skills and abilities in areas like systems programming, it is useful to put many tasks into a separate pre-lab exercise. For safety and consistent results, the students cannot complete the full lab task on their own – e.g. developing a virus at home. However, the complex sub-tasks involved can be separated into example programs that the students can do on their own. This helps to minimize the issue of students taking much different times to complete the lab work, especially if the TA must always be in the lab for the students to work.

Furthermore, we found that pre-labs should be as extensive as possible to ensure that students prepare for the lab session. When students only need to answer high-level theory questions, they often fail to see how the answer applies in practice. Pre-labs with programming tasks can make students prepare more carefully.

Our successful course at Infosys showed us that it's helpful to tie the lecture directly into the lab exercises as much as practicable. By directly showing students some

of the steps that they will perform in the lab and having students work though basics with the instructor in class, the students are much more prepared for the lab exercise. Further, this helps the instructor become fully aware of the level of the students.

We believe that it is useful to put both attack and defense into a single exercise. For example, we put attacks using Nessus and Metasploit into a combined exercise with Snort and firewall configuration. This allows students to perform an attack and then see their attack fail when a defense is in place. This helps with ethical concerns, as every attacking activity is tightly connected with a defensive activity. Further, it requires both the instructor and the student to weave together the technical knowledge for both attack and defense. Finally, as a practical matter, this adds steps to lab exercises that would otherwise be too short or too easy. For example, using Tripwire for host IDS is too simple for a whole lab, but it can be nicely connected with the rootkit lab to show how a stealthy rootkit can be detected when Tripwire is running. We will seek more ways to incorporate both attack and defense into each lab experience.

VI. Ethical Concerns

We now briefly discuss how ethical issues impact our instructional approach.

Livermore conducted a survey of faculty regarding ethical considerations in the instruction of penetration testing X.[2]. He found that most instructors believed that penetration testing should be taught and should be taught in an environment isolated from the rest of the network. We agreed and consequently designed an isolated lab, even though we don't believe that any of our attacks would be practical against production systems.

Livermore also found that most instructors believed that students should sign a lab agreement [2], which we also do. The wording of the lab agreement is quite stern; students who even briefly skim the statement would notice the dire consequences promised for rule-breaking, such as *retroactive* grade changes. However, Livermore found that most instructors believe that students should take an ethics course, which we do not require. We do briefly cover ethics in lecture in the first week of class. As an additional measure, we have students engage in an online discussion about various ethical questions related to course content.

We currently do not feel that further measures are required. Our perspective is that the level of the course is not sufficient to enable students to exploit many vulnerabilities in production software, especially given advances in security like stack and heap address randomization in operating systems. At the same time, students enter the course with already enough knowledge to operate as *script-kiddies*, who download and run exploits as is.

Logan and Clarkson decry hands-on instruction in attack methods without greater evidence that such instruction creates better security professionals [3]. They propose that instructors should first consider the skills that are most valuable in the enterprise. For our classes, this argument seems besides the point. Our primary goal is not to create security professionals or network administrators but to teach students of computer science both the principles of security and how attacks and defenses operate. Modern defenses often require substantial detail to understand. Preventing buffer overflows, for example, entails modifying operating systems in ways that are hard to understand only through theory. Consider stack randomization: when students have executed a buffer overflow exploit to gain root access, it is much easier for them to see how randomizing stack addresses makes it harder to find the shell code. Importantly, most students seem to recognize that such defenses make new vulnerabilities difficult to find and exploit without a much higher level of training than we provide in the lab.

Logan and Clarkson also point out that students learning attack skills, especially in live exercises, can become focused on "the 'thrill' of the hunt" [3]. This is an interesting point. Vigna, though taking a largely opposing view on teaching attack skills, also notes that students find attacking to be more fun [4]. This can be explained easily enough: attacking is more like open-ended puzzle-solving, while defense is typically more routine activity. However, does it mean that students will learn that hacking is more exciting and interesting to pursue than defense? One possible answer to this problem is to promote the students' futures in penetration testing. More realistically, the students can be shown how learning about vulnerabilities can be critical to understanding attacks and defenses in the systems they build.

VII. Related Work

We first describe related work in laboratory and exercise design, and then focus on discussions of the CTF and its role in security education.

[1] *Laboratory and Exercise Design*

A number of others have reported on their experiences designing and implementing laboratories and exercises for security course work. We now describe some of this prior work.

Tikekar and Bacon describe their laboratory setup and a large variety of projects used in their courses [5]. While they provide an excellent set of ideas for projects, many

of which we also use, they do not use NFS or virtualization and do not discuss principles of exercise design in much detail. Carlson provides a nice introduction to initiating security courses, also with a list of exercises and report of how successful each one was [6].

Schembari reports on experiential exercises designed for a course in cryptography [7]. He concludes that these experiences improve the learning and enjoyment of the course. Kessler and Hoag describe how to create forensics exercises without the use of a lab environment [8]. However, our offensive exercises require an isolated environment for safety.

Vigna describes several network configurations and comments on their respective advantages and disadvantages [4]. In particular, he finds that an image server that can reinstall the OS in a few minutes is a useful tool, as is a fail-safe mode for checking on the availability of victim machines and reinstalling and rebooting when needed. None of our exercises required these features, partially due to virtualization and partially due to having mainly closed-form exercises.

Border reports on the technical details of setting up a remote virtualization-based lab for classes including security [9]. His approach is based on VMWare. We are not convinced that virtualization is sufficient to make the lab exercises we offer safe and more acceptable to third parties, such as the university's network administrators.

Padman and Memon describe an interesting alternative to the isolated lab environment: a virtual laboratory that can be shared between multiple institutions [10]. The virtual laboratory is accessible through a browser interface and is remotely configurable. The browser interface would likely engender greater feelings of safety by third parties than virtualization alone (regardless of actual risk). This approach is compelling for sharing lab resources, but is not appropriate for our present situation.

Abler et al. describe the use of realistic network topologies for the security lab at the Georgia Institute of Technology [11]. The lab is unique in emulating a small part of the Internet, allowing for more realistic scenarios. This is an excellent approach, but it requires approximately 22 routers and most of our lab exercises have little direct benefit from the realistic networks. Nevertheless, their approach would create perhaps the most exciting and engaging environment for learning network security.

[2] *Security Competitions*

CTF exercises and competitions are well-known, and their use has been described for a variety of courses. The CTF idea was likely popularized by an event at the annual DEFCON hacker convention, which started in 1993 (see http://www.defcon.org/ for more information). The NSA holds a Cyber Defense Exercise (CDX) each year for senior computer science students in the US military academies.

Vigna describes the network setup for a *blue team/red team* exercise, in which the blue team defends and the red team attacks, and for a CTF exercise [4]. He describes how CTF exercises create a sense of team pride that goes beyond obtaining a grade; we report similar anecdotal findings. One of the most original aspects of the work is a *Treasure Hunt*, in which teams construct a multi-step realistic attack and race the other teams (rather than clash with each other). Students seemed to appreciate this format, but Vigna also reports that it requires roughly twice the effort to set up.

Wagner and Wudi describe the design and implementation of a "cyberwar" laboratory exercise, similar to the CTF exercise we used. Their exercise, however, focused more on system security and had an extended time frame (48 hours) [12]. While they propose a number of improvements, developing these exercises to the extent proposed requires substantial time and effort beyond what most educators have to give for one exercise.

White and Dodge report on an alternative model for competition, the National Collegiate Cyber Defense Competition (NCCDC) [13]. Conklin describes an earlier version of competition [14]. In the NCCDC, the focus for students is defense technologies, e.g. keeping services running and patching vulnerable systems. While this is also an important model for competition, we believe that students gain more technical skills from learning both attack and defense. For upper-level CSE students, we believe that learning both is more appropriate.

Eagle and Clark argue that the CTF exercise is important to fill a gap in the instruction of vulnerability discovery [15]. In particular, they divide approaches to security into protectionist and constructionist camps, the former focused on existing systems and known vulnerabilities and the latter focused on building secure, high-assurance systems. CTF exercises provide students with experiences needed to practice and understand constructionist thinking (partially through what Eagle and Clark call *destructionist* thinking, i.e. attacking the system). Since our students are mostly aiming to become software engineers, designers, and testers, we believe that the constructionist approach is more appropriate to their education. While students are not given exercises in building secure software in the ASCENT lab, we have designed a separate course just to cover secure programming practices.

VIII. Conclusions

In this paper, we described the ASCENT lab, a flexible instructional laboratory environment, and a set of exercises for use in the lab. The lab can be isolated, such as when students develop and test malware, or connected to other labs, such as for the CTF. The machines in the lab can run virtual machines, allowing for several students to have administrator privileges while taking turns on the same machine. We have also run the lab without virtual machines when they weren't needed. The exercises have met with varying levels of success; we discussed some lessons learned and ways we plan to improve. We transported our lab exercises to another environment, namely a corporate classroom in India, where the lab was quite successful. Ethical concerns are also important for any security instruction and we discussed some of the ethical considerations that went into our design.

IX. Acknowledgements

Thanks to Steve Tate for identifying key pieces of laboratory equipment and giving us a basic lab design to build upon. Donggang Liu co-taught the primary lab course and contributed most of the buffer overflow lab, including the vulnerable server. Amit Singh kindly provided the Jingle Bell code from his website. Giovanni Vigna graciously provided the prior years' CTF images on the UCSB CTF website. Thanks to Steve Tate, Ebru Celikel, Vandana Gunupudi, and Roopa Vishwanathan for their efforts running the CTF. Thanks also to Infosys Technologies Limited, and in particular Dr. V. P. Kochikar, Dr. P. Suresh P., and Alsaad Ishaq, who hosted us at the Infosys campus in Mysore.

X. References

[1] A. Stubblefield, J. Ioannidis, and A. D. Rubin, "A key recovery attack on the 802.11b wired equivalent privacy protocol (WEP)," *ACM Transactions on Information and System Security (TISSEC)*, Volume 7, Issue 2, pp. 319-332, May 2004.

[2] J. Livermore, "What Are Faculty Attitudes Toward Teaching Ethical Hacking and Penetration Testing?" Proc. Colloquium for Information Systems Security Education (CISSE '07), June 2007.

[3] P. Y. Logan and A. Clarkson, "Teaching Students to Hack: Curriculum Issues in Information Security," Proc. SIGCSE Technical Symposium on Computer Science Education (SIGCSE), 2005.

[4] G. Vigna, "Teaching Hands-On Network Security: Testbeds and Live Exercises," Journal of Information Warfare, Vol. 2, Issue 3, Aug. 2003.

[5] R. Tikekar and T. Bacon, "The Challenges of Designing Lab Exercises for a Curriculum in Computer Security," The Journal of Computing in Small Colleges, Vol. 18, Issue 5, May 2003.

[6] D. Carlson, "Teaching Computer Security," SIGCSE Bulletin, Vol. 36, Issue 2, 2004.

[7] N. P. Schembari, "'Hands-On Crypto': Experiential Learning in Cryptography," Proc. Colloquium for Information Systems Security Education (CISSE '07), June 2007.

[8] G. C. Kessler and J. Hoag, "The Power of Simple Hands-On Cyberforensics Exercises: A Guide for Faculty," Proc. Colloquium for Information Systems Security Education (CISSE '08), June 2008.

[9] C. Border, "The development and deployment of a multi-user, remote access virtualization system for networking, security, and system administration classes," Proc. SIGCSE Technical Symposium on Computer Science Education (SIGCSE), Mar. 2007.

[10] V. Padman and N. Memon, "Design of A Virtual Laboratory for Information Assurance Education and Research," Proc. IEEE Workshop on Information Assurance and Security, 2005.

[11] R. T. Abler, D. Contis, J. B. Grizzard, and H. L. Owen, "Georgia tech information security center hands-on network security laboratory," IEEE Trans. on Education, Vol. 49, Issue 1, Feb. 2006.

[12] P. J. Wagner and J. M. Wudi, "Designing and Implementing a Cyberwar Laboratory Exercise for a Computer Security Course," *ACM SIGCSE Bulletin*, Vol. 36, Issue 1, pp. 402 – 406 (2004).

[13] G. B. White and R. C. Dodge, "The National Collegiate Cyber Defense Competition: What are the next steps?" Proc. Colloquium for Information Systems Security Education (CISSE '07), June 2007.

[14] A. Conklin, "Cyber Defense Competitions and Information Security Education: An Active Learning Solution for a Capstone Course," Proc. Hawaii Intl. Conference on System Sciences (HICSS), 2006.

[15] C. Eagle and J. L. Clark, "Capture-the-Flag: Learning Computer Security Under Fire," Proc. Sixth Workshop on Education in Computer Security (WECS), July 2004.