# TEACHING SECURITY LABS WITH WEB APPLICATIONS, BUFFER OVERFLOWS AND FIREWALL CONFIGURATIONS[*]

*Richard Weiss*
*The Evergreen State College*
*2700 Evergreen Parkway, NW*
*Olympia, WA 98505*
*weissr@evergreen.edu*

*Jens Mache*
*Lewis & Clark College*
*Portland, OR 97219*
*jmache@lclark.edu*

## ABSTRACT

Teaching information security is becoming an important part of the computer science curriculum. Students learn best when they have hands-on experience. This paper presents examples of laboratory exercises which we used in an undergraduate class and were evaluated by the students. We present the results of that survey and the features of those exercises with strengths and weaknesses from the instructor's perspective, and whether they cover the fundamental security principles. Since many schools do not have a security class, we also examine how these exercises could be included in standard core courses such as networks, computer architecture, database systems, or software engineering.

## INTRODUCTION

There is a growing demand for professionals who are trained in computer security [2, 3, 7, 12]. This can range from programmers to consultants and IT professionals. Information security is not currently part of the core computer science curriculum, and it seems likely that the way to meet the demand is to integrate it into existing courses, as well as specialized classes in information security. Solving the problem of effectively integrating information security into the undergraduate curriculum requires more hands-on experience. On the one hand, there are several texts which have been published, e.g. *Counter Hack Reloaded* [10], *Hacking- The Art of Exploitation* [4], and several others [6, 9,11]. There is also a myriad of online resources which one could use. We have adapted tutorials and other information available from online and private sources to create a collection, which we hope captures some of the important ideas. We have tested our lab exercises at Lewis and Clark College (L&C) and The Evergreen State College. We have three reasons for presenting our experience. We would like to encourage the following: the inclusion of some hands-on security in all core classes, the creation of easy-to-use and easy-to-modify curricular resources for faculty, and the creation of exercises which would complement current and future textbooks.

Although "legacy" vulnerabilities will continue to be around for a long time, the technology of malware changes rapidly, and the software that a student learns about as a freshman may be obsolete four years later. In order to understand what exercises will have lasting value for our students, we have attempted to formulate some principles of information security education based on conversations in two BoF's at SIGCSE 2011. They include teaching students to:

1. validate input, avoid extra features, test API's used,
2. understand how their code works
3. debug complex systems
4. test boundary cases and failure modes
5. follow good design and engineering practices
6. understand ethical hacking and malware
7. make security features easy to use, so there is no incentive for disabling them.

As argued by George Ledin [7], computer science education is not keeping up with malware development in particular, which has grown in sophistication and prevalence, and what is needed is hands-on experience. While vulnerable applications such as telnet and ftp have been replaced by more secure versions, some of the basic vulnerabilities such as buffer overflow are still relevant today. The reason is that security needs to be built in from the beginning of a software project, and often that is neglected in favor of cost and performance. This is why security education should be integrated into the curriculum. One of the obstacles to achieving this is that most computer science faculty do not have expertise or experience in this area so are not familiar with the basic techniques [12]. In particular, any curricular resources need to be easy-to-use and self-contained. This paper describes several exercises that we have tried with undergraduate students and what the results were.

Our approach was to select topics we thought were important and to find laboratory exercises which would illustrate them. The topics we picked were buffer overflows, Web vulnerabilities, configuring a firewall to protect a network and using netcat to set up backdoors and relays. Below, we discuss the content of each of four exercises, their strengths and weaknesses, and how the students perceived them. We also examined the exercises in terms of the types of vulnerabilities that are most common and their coverage of the principles of security education listed above. At the end, we sketch a botnet exercise which builds on the buffer overflow and netcat exercises.

## METHODOLOGY

We tested four exercises describe below. They were assigned to 14 undergraduate students in a class on security at Lewis and Clark College (L&C). The buffer overflow was also tested independently by a couple of undergraduate students at The Evergreen State College. The students at L&C were given background material and lectures before being asked to complete the exercises. At the end of the semester, they were given a survey to capture their reactions to the exercises. Thirteen students rated the exercises in seven dimensions: level of interest, amount of time spent, whether time spent was worthwhile, whether it increased their understanding, whether they were sufficiently prepared, the level of difficulty, whether it would motivate them to further study, see Fig. 1. First, we describe the exercises.

### Buffer Overflow.

We tested Wenliang Du's lab [1] as a model exercise for exploiting a buffer overflow. This was done in an undergraduate class and required that students have root privileges in their development environment and was done with Linux. Buffer overflow is still on the list of most common vulnerabilities. For example at a SICCSE 2011 birds of a feather session on computer security, this was the first thing mentioned that students should know about. A buffer overflow occurs when data is copied from one area of memory to another usually on the program stack and more data is copied than there is space allocated, so other information is overwritten. When malware does this, the purpose is to change the control flow of the program, for example overwriting the return address of a function call. This can occur in C, when the functions **gets** or **strcpy** are used, but there are examples in all standard languages. There have been several defenses against this, such as address randomization, compiler-inserted code and non-execution protection of the stack. These are disabled in the exercise, otherwise the exploit becomes significantly more difficult.

### Exploiting Vulnerabilities in Web Applications.

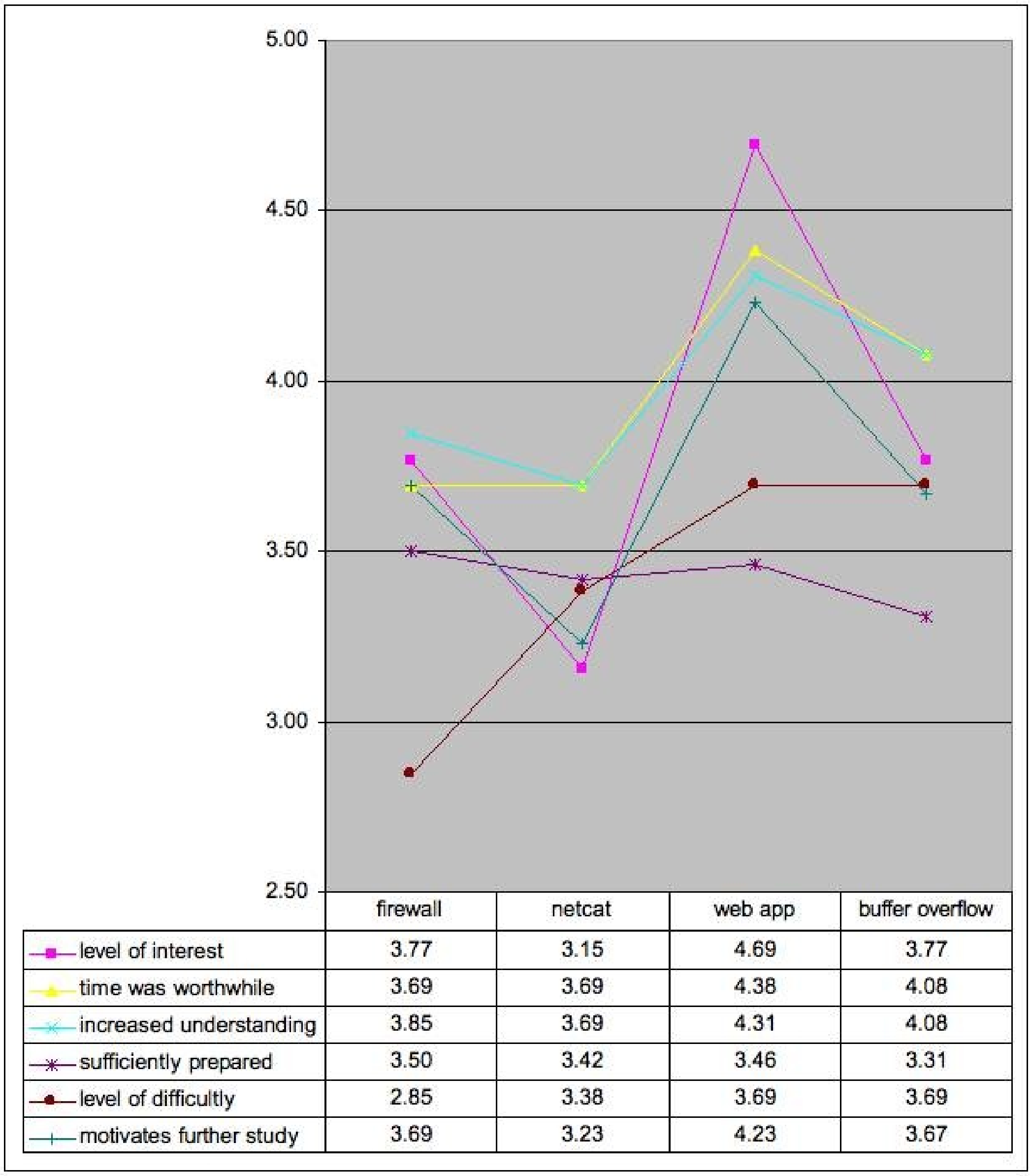
This exercise was based on a competitive capture the flag (CTF) exercise called Security First, developed at UNCC [14]. The premise is that the students are attacking a bank website and are competing to be the first to login as the bank's financial manager and transfer funds between two specified accounts. The game includes cracking hashed passwords using md5crack.

### Firewall Configuration.

This was also presented as a competitive game, where each student was simultaneously writing firewall configuration rules and testing the rules of other players in order to win points. The game was developed by Ken Williams [13]. The students also read [9] for background, and they had to understand the logical structure of how the rules interact.

**Netcat Exercise.**

Netcat, often referred to as the "Swiss army knife for TCP/IP", is used by system administrators and attackers to interact with systems across a network. Netcat is well-described in [10]. In the lab, students put netcat to many uses, including port scanning, making connections to open ports, moving files, as well as building back-doors and relays.



| | firewall | netcat | web app | buffer overflow |
|---|---|---|---|---|
| level of interest | 3.77 | 3.15 | 4.69 | 3.77 |
| time was worthwhile | 3.69 | 3.69 | 4.38 | 4.08 |
| increased understanding | 3.85 | 3.69 | 4.31 | 4.08 |
| sufficiently prepared | 3.50 | 3.42 | 3.46 | 3.31 |
| level of difficultly | 2.85 | 3.38 | 3.69 | 3.69 |
| motivates further study | 3.69 | 3.23 | 4.23 | 3.67 |

Figure 1. Student survey results for the four exercises, using a Lickert scale from 1 to 5.

## DISCUSSION OF RESULTS

Fig. 1 shows how thirteen undergraduate students rated the four lab exercises. During the semester, students first worked on the firewall configuration, then on the netcat exercise, then on the web application, finally on the buffer overflow lab.

A seventh question was about time spent. For the four exercises, the averages were 1.42, 2.00, 2.42 and 2.00, where 1 represented <= 2 hours, 2 represented <= 4 hours, 3 represented <= 6 hours, 4 represented <= 8 hours and 5 represented > 8 hours.

### Buffer Overflow.

One of the strengths of the buffer overflow exercise is that students will most likely come away with an appreciation for the first six principles of information security above. Another strength is that it remains among the top 10 software vulnerabilities encountered. However, in order to understand how buffer overflows work and how to avoid or detect and eliminate them, one needs to understand some basic computer architecture and low-level assembly language and hardware issues. Potentially, this makes the exercise difficult for students who have not had that background. On the other hand, it can be an interesting exercise where the goal is to teach about assembly language, function calls, and the program stack. This could be useful in two ways. It can motivate students to take classes on computer architecture/organization and operating systems, or it could be used in these classes to increase student engagement.

Just reading the lab description provided by [1] was not sufficient for our students because they did not have the context which was assumed. This exercise illustrated the difference between a reading assignment and a hands-on laboratory. Even though the ideas are easy to explain, and Erickson [4] does a good job, many students still had some difficulty in carrying out the steps. The exercise covers the first principles of teaching students to validate input and test API's used. The students viewed this exercise as the most difficult of all of the exercises, and as expected, it increased their understanding significantly.

Some major advances in hardware and software which have made buffer overflows more difficult to exploit are: the ability to turn on address randomization and to turn off executability of stack memory. The impact of the former is that it might require more trials to produce a successful exploit in which the address of the malicious code is predicted correctly.

### Exploiting Vulnerabilities in Web Applications (Security First Bank).

This type of exercise is relevant to many different classes in the computer science curriculum, including Networks, Database Systems, and Software Engineering. One of the strengths of this exercise is that it covers many different techniques and concepts in information security and potentially engages students at multiple levels of difficulty. It covered all but number 3 from the OWASP (Open Web Application Security Project) top ten vulnerabilities:

1. Cross Site Scripting (when a program does not validate or encode input data)
2. Injection Flaws (particularly SQL injection)

3. Malicious File Execution
4. Insecure Direct Object Reference (this is where a parameter in a URL can be manipulated to access files, records or other type of information)
5. Cross Site Request Forgery
6. Information Leakage and Improper Error Handling (error messages that contain source code and data)
7. Broken Authentication and Session Management (e.g. session tokens)
8. Insecure Cryptographic Storage
9. Insecure Communication (failing to use cryptographic tools to protect sensitive information sent over a network)
10. Failure to Restrict URL Access.

In addition to password cracking, the techniques involved SQL injection, information leakage through improper error handling, filter evasion, insecure cryptographic storage, and session hijacking to change account settings and use the "forgot my password" option. One of the weaknesses is that the exercise does not explicitly address the principles or provide training in defense. However, this could be improved through accompanying assignments, requiring students to reflect on the design decisions that led to vulnerabilities and how they could turn their exploits into testing scenarios. The exercise integrates well with readings in [9].

This exercise can extend over two weeks and utilizes graduated difficulty. This means that the more skill and understanding a student has, the farther she will get in the assignment. There is a potential difficulty in that the exercise goals have dependencies and the student cannot tackle some parts of the exercise without first completing others. However, the fact that students rated this exercise highly means that this was not actually a problem. There are many facets to this exercise and it would be possible to create scaled-down versions for more introductory classes possibly by including hints. One of the weaknesses is that the VM image file is 1.8 GB compressed, so students cannot easily download it from home. The graph in figure 1 shows that the Web App was rated the highest in all categories (level of difficulty was a tie).

**Firewall Configuration.**

One of the strengths of this exercise is that it specifically focuses on defense skills, although it only covers one of the top 10 vulnerabilities, i.e. failure to restrict URL access. There was a significant amount of work to set up the exercise, and could use some refinement. For example, it uses CISCO syntax, whereas some would prefer IPtables syntax. It also uses a lot of machines to run the game server, and a separate computer for each competitor and the admin GUI. One could potentially develop this for smartphones. The students liked the competitive aspect. This exercise teaches many of the basic concepts in networking, including how IP addresses are assigned. It focuses on network layers 3 and 4.

**Netcat and Botnets.**

This exercise was mostly exploring the uses for netcat and was derived from [10]. One of its strengths is that can be used for treasure hunts and with botnets. A botnet is a collection of computers, which are controlled by a smaller set of computers, where the members of the botnet request instructions from the master computers. With a netcat listener on the target, an attacker can connect to it. Moreover, netcat can "shovel shells" to evade firewalls. It is very easy for instructors to set up on Linux. We thought about developing a sequence of exercises to extend the buffer overflow exercise and to increase the power of the exploit.

There are issues of safety with this exercise in the sense that a student lab should not adversely affect the school network, nor could it impinge on the privacy of other students. Potentially, this would make running a botnet difficult. Gephardt and Kuperman [5] have addressed the question of how to provide the hardware and software environment for doing this at small colleges, given that there are limited resources which can be devoted to a single class.

## CONCLUSIONS

The students ranked the Web Applications lab exercise the highest in terms of interest and where they learned the most. It also had the greatest coverage of topics and was easy to grade. Students get immediate and incremental feedback and it can be run as a competition. It was worthwhile for the students, not too hard to set up. The Buffer Overflow lab exercise needed more background, but it has potential as part of systems/assembly language class or to motivate students to take more systems classes.

While the exercises we tried are not perfect, they represent more or less the types of hands-on labs that we think are needed. The Web applications exercise seems to come the closest to achieving the goals we stated above: it could be used in several core classes such as networks, databases and operating systems, it is relatively easy to use, and it can be used with many textbooks.

We still need to think about what we can teach and how to provide laboratory exercises for students to use in this environment. For many experiments, students need to have privileged access to the computers in the laboratory (host or virtual machine guest) and they sometimes need to be isolated from each other and the rest of the world; however, the exercises we describe here have minimal requirements.

## REFERENCES

[1] Du, W., www.cis.syr.edu/~wedu/seed/Labs/Vulnerability/Buffer_Overflow/, retrieved April 22, 2011.

[2] Bratus, S., Shubina, A., Locasto, M., Teaching the principles of the Hacker Curriculum to undergraduates, *SIGCSE 2010*, 122-126, 2010.

[3] Cooper, S., Nickell, C., Piotrowski, V., Oldfield, B., Abdallah, A., Bishop, M., Caelli, B., Dark, M., Hawthorne, E., Hoffman, L., Pérez, L., Pfleeger, C., Raines,

R., Schou, C., Brynielsson, J., An exploration of the current state of information assurance education, *SIGCSE Bulletin*, 41 (4), 109-125, 2009.

[4] Erickson, J., *Hacking: The Art of Exploitation*, San Francisco, CA: No Starch Press, 2008.

[5] Gephardt, Kuperman, Design of a virtual computer lab environment for hands-on computer security exercises, *Journal of Computing Sciences in Colleges*, 26 (1), 2010.

[6] Kaufman, C., Perlman, R., Speciner, M., *Network Security: Private Communication in a Public World*, Upper Saddle River, NJ: Prentice Hall, 2007.

[7] Ledin, G., The growing harm of not teaching malware, *CACM* , 48 (1), 144-146, 2005.

[8] Logan, P. Y., Clarkson, A., Teaching students to hack: Curriculum issues in computer security, *SIGCSE*, 157-161, 2005.

[9] Simpson, M. T., Backman, K., Corley, J. E., *Hands-On Ethical Hacking and Network Defense*, Boston, MA: Cengage Learning, 2011.

[10] Skoudis, E., Liston, T., *Counter Hack Reloaded*, Upper Saddle River, NJ: Prentice Hall, 2006.

[11] Smith, S., Marchesini, J., *The Craft of System Security*, Boston, MA: Pearson Education, 2008.

[12] Turner, C. F., Taylor, B., Kaza, S., Security in computer literacy - A model for design, dissemination, and assessment, *SIGCSE'11*, 2011.

[13] Williams, K., http://williams.comp.ncat.edu/firesim, retrieved January 15, 2011.

[14] Yu, H., Williams, K., Xu, J., Yuan, X., Chu, B., Kang, B., Kombol, T., Interactive simulation tools for information assurance education, *Proceedings of the Second Annual Conference on Education in Information Security* (ACEIS), 2009.