Defense Against the Dark Arts: An Approach to Introductory Computer Security Education

Sam Boger '12 Brown University
Advisor: Roberto Tamassia

May 3, 2012

Abstract

More engineers should receive basic education about computer security. The prevlence of cyber crime demands fundamental change to the development and maintenance of software. Currently, many universities do not provide undergraduate security courses, only only offer these courses as a specialty. I propose a new focus for computer security education based on reaching a wider audience and prioritizing breadth over depth. I supplement this proposal with implementations of four security projects for the undergraduate computer security course at Brown University.

Chapter 1

Introduction

1.1 Motivation

Introducing security to computer science students requires basic knowledge of many fields. The topic cannot be taught in a vacuum, since many security exploits emerge from oversights or fundamental problems with other computer infrastructure. Therefore an effective security course must have students with some background in these essential computer science topics. Instead of requiring this knowledge as a prerequisite, I believe a well-designed course can both teach the background and the security topics.

An introductory course in computer security occupies a unique position in undergraduate computer science education. Most upper level electives in computer science provide an in-depth study of one particular field, such as computer vision or robotics. These courses typically start with the basic theories of the field then turn to more specific or sophisticated techniques. For example, a course on databases may start by learning about basic relational databases before delving into noSQL or fault tolerance. Courses on computer security must cover a lot more different topics at a shallow depth. This means that introductory security courses can be very accessible and wide reaching.

A broad course must still be effective in its goals. This course at Brown University should produce students that can think critically about security and approach coding with awareness to potential problems. A student must understand a potential attackers perspective to write secure code. Code that is written assuming that all clients are legitimate and honest will lead to vulnerabilities. Any use of user input, user data, or dynamic content must assume the worst from the end user. If the programmer has no knowledge of general categories of attacks, it is infeasible to write secure code. I have created a series of projects for the introductory computer science course that teaches important security lessons from an applied perspective. After completing the projects, students have good first hand experience attacking software and web applications. They must also analyze network security and the implications of even briefly compromising a victims account. These topics give them a basis for further study in computer security, but more importantly, an understanding of what vulnerabilities plague current software. This understanding is vital for future engineers so they do not create software that is vulnerable to these known attacks.

1.2 Contributions

For this report, I will provide a summary of the cyber security industry, focusing on the damage from cyber crime and the means corporations take to prevent and minimize this damage. This will serve as background for my desire to bring security education to more software engineers. I will continue with an analysis of computer security education at Brown University and other institutions across the US and Canada. Finally, my main contribution from this report are the implementations of four computer security projects targeted towards undergraduates with few prerequisites. These projects each teach valuable lessons about vulnerabilities and defense techniques. The projects have the benefit of being maintainable, highly accessible, and require few resources to complete. These projects are therefore ideal for distribution outside of Brown's computer science department.

Chapter 2

The State of Computer Security

2.1 Cyber Crime

Security has become a major part of the software industry. The true economic impact of attacks is hard to estimate, since data can only be collected by surveys, which have inaccurate results due to the nature of the industry.[3] Surveys have few responses that report significant cybercrime damages that greatly affect the final outcomes, so these surveys are very likely to be inaccurate if they happen to include one more or one less such response. Estimates range widely, but most sources estimate somewhere around $100b in damages every year from data breaches.[5] Therefore companies have an enormous economic incentive to avoid these damages.

Some of these vulnerabilities come from large-scale network issues, Internet infrastructure vulnerabilities, or other sources that cannot be easily addressed by individual efforts. Others are attacks on applications, including stealing users credentials for web applications, which may lead to further unauthorized access to other accounts or services. These attacks can be prevented by more careful programming.

2.2 Public Opinion

High profile security breaches cause dramatic changes in the publics perception of software companies. For example, when Sony potentially lost 70 million customers credit card information in 2011, they estimated the damage to their bottom line totaled over $170 million[17]. Undoubtedly some of that loss was in customer service for those affected, customer turnover, and decline in new customers. While other services may have had similar security flaws, only a big public disclosure will have those detrimental effects on a companys profits.

Quick development and deployment of patches can help mitigate damages from cyber attacks. The damage to public relations, however, is immediately felt even if the attack does not cause much real damage. Prevention of cyber attacks is the only way to help an organization remain in high esteem in terms of security. A good record on numerous products and applications can be instantly ruined by a single publicized breach.

Unfortunately public perception of security does not always match reality. For example,

if users experience unauthorized access of their account they claim, my account has been hacked! when they likely mean that their password was stolen, or they did not log out of their account on a public computer. The company owning the website may have been perfectly secure, and the attacker gained access to the password through another site where the user used the same password, or by eavesdropping on unencrypted traffic.[10] Similarly, a corporation that loses customer data may not disclose the attack at all, since announcing that their service was vulnerable to attacks may indicate general security weakness to cyber criminals, potentially resulting in further attacks.

2.3 The Corporate Viewpoint

Complex software requires integration of many distinct software packages written by many different authors. It is unrealistic to assume that a product will only use code written by that companys employees. Therefore there will always be inherent risk with deploying software, since the security of the code cannot be internally verified. Once the engineering department has finalized code for an application, the company may pay for security audits by third parties as discussed earlier. In addition, some companies have tried to avoid public disclosure or sale of vulnerabilities to undesirable parties by offering bounties on vulnerabilities in critical software. Google has done this, offering thousands of dollars for severe vulnerabilities in their major products.[19]

Producers of commercial software must be confident in the security of their product. Use of third party software can account for significant vulnerabilities in otherwise secure applications. For example, its estimated that around 80% of vulnerabilities in Microsoft software originated in third party code.[16] While it is sometimes possible for companies to patch third party software, the reputation of both parties, if not their money, is at stake. For example, Apple refused to enable Flash on their mobile devices in part because of the many vulnerabilities found in Flash over the past few years, and the inability of Flash to fix these vulnerabilities in a timely manner.[18] Adobe eventually discontinued their development of Flash for mobile.[12] Adobe did not cite Apple not supporting their technology, or the resulting negative press, in their decision, but it seems likely that Apples decision affected the discontinuation.

The presence of these post-engineering security measures indicates that companies do not believe that their engineers can write secure software. As a result, the entire company assumes lots of risk and spends many resources on security concerns. The alternative to focusing on these issue after producing the product is security by design. Google Chrome used this philosophy, as can be seen on their website

> Security is a team responsibility. Theres a common misconception that security can be handled as a feature or add-on component. The fact is that security of any complex piece of software is a cross-cutting concern, determined by millions of seemingly innocuous decisions being made by developers every day. Thats why its essential for every team member to be aware of secure development practices, and work with their security team throughout the lifecycle of the project. This general awareness helps supplement our normal security review process of auditing, regression testing, and fuzzing.[13]

By realizing that security is a vital part of a web browser, Google created arguably the most secure browser available. It is worth noting, however, that vulnerabilities in Chrome have still been discovered.[11]

2.4 Vulnerability Discovery

All major corporations dedicate resources to ensuring the security of their services. Some companies ensure that all code undergoes a security review before release. Security teams may also be tasked with setting up automated tools to detect attacks, mitigate damage once attacks are detected, and quickly patch vulnerable services. Separate personnel may be devoted to handling public relations relating to successful or unsuccessful attacks. These in-house solutions can be supplemented by third party security reviews, which can be conducted under several different paradigms. The third party may have access to the source code of the application, may have no access to the application, or may have privileged access to the application. In all cases the third party is expected to discover and disclose vulnerabilities to the authors.

Anyone not hired to do an audit that discovers a vulnerability has several options for what to do with that information. The discoverer may never disclose the information, personally exploit the vulnerability, or sell the information to a criminal or criminal organization. Instead, he or she may employ full disclosure, where all information regarding vulnerabilities is given to the public as soon as it is discovered. Alternatively, he or she may choose responsible disclosure, where the exploit is disclosed only to the developer of the software who keeps the information secret long enough to patch the vulnerability before publicly releasing the bug and the fix.

Some companies buy or discover vulnerabilities in software of major corporations to sell them the details of the exploit before disclosing it to the public. These companies, called security information providers (SIPs), provide an appealing option for independent researchers who want compensation for finding such vulnerabilities. They have the option of selling this information to attackers illegally or selling the information to these legitimate SIPs, who usually offer less compensation, but at no legal risk. These companies have been estimated to take 10%-15% of the vulnerability market.[4] Without these SIPs, more attacks would reach the hands of cyber criminals. Furthermore, this service provides an incentive for law-abiding independent security analysts.

2.5 The Asymmetry of Cyber Security

Many aspects of cyber security are disproportionate in nature. A very small coding error may result in enormous damage to a corporations revenue, reputation, or even infrastructure. Additionally, a very small and/or poorly funded attacking party may cause this high profile, widely spread, and economically devastating damage. These natural asymmetries mean that companies cannot be assured that spending more time or money on security will necessarily result in significantly less risk or damage, since any oversight could cause as much damage as a major flaw.

The complexity of software results in the presence of security vulnerabilities. There

are simply too many lines of code to check, too many interfaces into the application, too many features, and too many edge cases to catch all potentially dangerous interactions. A thorough review of even relatively small applications is difficult, since there are tens of thousands of lines of code already written, not to mention the third party libraries used by this code. Spending significant resources to review 98% of this code might still leave one key vulnerability undiscovered, which would mean that all of the resources spent had no effect on preventing damage. A more thorough security review makes no guarantee of results until a complete and accurate review is performed. Even then, the security of the application must be proved before the producer can be absolutely sure of the security of their application. Such a proof is infeasible or impossible due to the number of programming languages, differences in configurations and system specifications, and numerous other details that cannot be accounted for.

Similarly, many important exploits do not require sophisticated attackers. Smaller applications, or applications handling less sensitive data than say, a bank, might feel more secure if the application could defend against all simple attacks, while perhaps being vulnerable to large scale efforts. Unfortunately, the nature of common vulnerabilities (like cross site scripting, SQL injections, buffer overflows) means that any attacker with basic resources could implement the attack. Open source software such as Tor (to help the attacker stay anonymous) and Metasploit (to discover vulnerabilities and implement attacks) empower individual attackers with powerful tools. Compromising user accounts on innocuous sites may still be valuable due to the prevalence of repeated passwords across sites. A study by a security firm found that nearly three quarters of their users duplicate their banking password on other sites.[15] Therefore it is unreasonable for even small applications to assume that no adequately sophisticated attacker will care enough, or have time, to attack a service. It is difficult to fly under the radar of attackers.

2.6 Relying on Users

It is not reasonable for everyday users to employ best practices when surfing the web. For example, even the presence of the lock icon in Google Chrome, which indicates that the connection is encrypted with a public key using a valid certificate, is not something many users know about.[2] Even though cryptographically secure systems have been established and are being used properly by browsers, end users have no idea what the results of these measures mean, or how to check the results. Clearly the end user cannot be expected to make well-informed security decisions.

Security applications that trust the end user as the ultimate authority go unused, or cause users to turn off all protection to prevent annoyance. A telling example is Windows Vista, which prompted many users to look up guides on turning off all security measures so their actions were not interrupted with security warnings. Similarly, the Firefox add-on NoScript fell victim to the same problem. Almost any modern website requires Javascript to run properly, so a fully enabled NoScript breaks almost every webpage. Users must add specific exceptions, which they have no idea whether these exceptions are legitimate or not, until their website works. This can result in uninstalling NoScript, or whitelisting entire websites at a time, rendering the protection worthless. Pushing the burden of making security decisions to the end user does not work at large scales. Manual inspection is a poor method of

finding malicious code which can be quite cleverly disguised to be unrecognizable. For example, no user would expect "PHNjcmlwdD5hbGVydCgndGVzdDMnKTwvc2NyaXB0Pg"[9] to be executable JavaScript, but a browser will. Therefore security tools must make decisions for the user, potentially allowing user override in case of false positives.

2.7 The Broad Attack Surface

Any part of an application can lead to a severe vulnerability for the rest of the application. Many exploits use unused or insignificant features to gain access to more important application functionality. For example, recent attacks on the internal networks of automobiles leveraged insecurities in mobile Bluetooth pairing to gain access to the brakes of the automobile.[7] An arbitrary kernel read vulnerabilities in Linux have been implemented by an integer overflow vulnerability in one particular disk driver.[8] Even code that does not access sensitive values needs significant security consideration.

One method for reducing the risk for vulnerabilities is to simply create a more simple application. A smaller attack surface necessarily means that the attackers job is more difficult. Using fewer common resources means it is less likely that an attack discovered on another application will compromise your application as well. Fewer lines of code mean that an effective security audit is easier to perform. This approach is considered by grsecurity, which prevents many kernel modules from ever being loaded, thereby making any discovered exploits in these rarely used modules irrelevant.[14]

2.8 A New Focus

Instead of focusing on infeasible code reviews, trusting end-users with complex security issues, or trying to buy out the vulnerability market, I propose that companies should focus on hiring more security-minded engineers and educating the ones they already have.

Whether the industry likes it or not, it is possible that the government will expect software vendors to take more of the security responsibility. Following a report of the top 25 vulnerabilities in software, Alan Paller, the director of research for SANS, stated, The purpose is to shift the responsibility for fixing errors to the vendor so that the vendor fixes it before it is delivered. SANS was tasked to create a list of these errors partly by the NSA and DHS.

Such a shift is a long-term solution to an ever-present problem, but one that could be very effective. When writing code, it is very important to engage in best practices. There are numerous institutions that have detailed proper security practices, including a 50-page document put out by SANS.[1] While keeping up to date with current best practices will always be a step in the right direction, it seems infeasible to keep all of these details in mind while trying to write effective code in a production environment, which often includes time pressure. Moreover, there are so many different documents and practices that it becomes hard to even decide on the best best practices to begin with.

Instead, I believe that engineers simply need to better understand the thought process, tools, and methods of attackers when coding. For instance, instead of trusting the best practices for escaping user input, the programmer should consider possible input in the

particular context of the application to determine if any user input could have undesirable effects. For example, only the developer knows what encodings could potentially be used in the current context, so they know whether only considering UTF-8 inputs is a good assumption or not. Traditionally, penetration testing has been described as a mixture of art and science. Similarly, coding securely must be considered an art, trusting the skills and experience of the programmer. For this to work, programmers should have experience being on the black hat side of the scenario. They must realize that all code presents a potential for vulnerability, especially if it is operating with escalated privileges, or handles user input in any way (even if the user does not usually modify the value, like a browser cookie).

Engineers should be constantly penetration testing their own code. Obviously, this does not fix major design flaws or complex problems in integrating code, but it would help to reduce a large class of vulnerability–simple programmer error. For engineers to do this properly, they must be well educated in both security practices and in hacking techniques. A later subsection of this paper will detail a proposed philosophy, and some newly developed projects, for an undergraduate introductory course in computer security.

I do not believe that engineers should hold all responsibility for exploits in their code. The previously mentioned approaches to improving software security should still be maintained and developed. No single approach will be completely effective at securing software. This is simply one approach that I believe needs more attention and research.

2.9 Alternative Attacks

As mentioned earlier, some attacks are non-technical in nature and cannot be prevented with better engineering. The public will always be more technologically ignorant than software engineers and security experts about what to do with their private information. Phising and social engineering attacks will always be present and effective, since they only need uneducated or inattentive victims to succeed.

Software companies should not be responsible for educating the general population about safe browsing techniques or secure passwords. It will help, however, to educate more computer savvy people about basic security measures and technologies. With the ever-increasing connectivity of the world and the social graph, increasing the number of people who know these tips can greatly increase the number of people who learn these second-hand. Therefore a greater exposure to introductory security classes could be an effective mechanism for reducing the success of these indirect attacks on individuals. Perhaps if Syrian President Bashar al-Assad had known a graduate of CS166 at Brown University his password might not have been "12345".[6]

Chapter 3

Modern Computer Security Education

3.1 Security Across the US and Canada

To make a real impact on the general population of software engineers, security principles should be taught at many institutions. Computer security has been studied for decades, but still many undergraduate programs lack practical education in this subject. Most institutions have a cryptography course, and many more have a network security course which overlaps significantly, but far from completely, with the proposed curriculum. The following information was collected from other institutions public facing websites detailing their computer science curriculum. Courses listed at each institution are done on a best-effort basis using the search tools available. This data reveals that many institutions have limited offerings in computer security despite the overwhelming evidence that vulnerabilities are ubiquitous across the software landscape. Some Universities like Cal Tech and the University of North Carolina seem to have no security courses available at all. Even for students with some courses available, all institutions except Brown and the University of Alberta have significant (at least two years) of prerequisites for their security course. This means that only students that specifically choose to take a security course will graduate with exposure to this way of thinking.

Institution	Survey Course	Any Undergraduate	Few Prerequisites	Graduate	CS Cryptography
Brown	1	1	1	1	1
Harvard				1	1
Dartmouth	1	1			1
Yale		1		1	1
Cornell	1	1			1
Columbia		1			1
Princeton	1	1			
MIT				1	1
Stanford	1	1			1
Goergia Tech	1	1		1	1
Cal Tech					
Carnegie Mellon				1	1
UPenn	1	1			
UC Berkley	1	1		1	
UC Davis	1	1		1	
Delaware State		1			
University of Nebraska					
University of Nevada Las Vegas	1	1			
University of North Dakota		1			
University of South Dakota		1			1
Unviersity of Vermont		1			
University of North Carolina					
University of Miami		1			1
University of Georgia	1	1			
University of Illinois	1	1			
McGill University		1			1
University of Toronto					
University of British Columbia					
University of Alberta	1	1	1		1

Figure 3.1: Presence of security courses at selected Universities

3.2 Security at Brown

Brown University has security courses for many different audiences.

There is an international policy course aimed at bridging the gap between technical people and policy makers. This course covers many basic technical topics like network programming as well as more general and political topics like international law governing cybercrime. This course complements the ideas in this paper from a less technical perspective. Law and policy cannot be made intelligently without understanding the technical climate of the Internet and computer security. The recent proposed bills in the US like SOPA and PIPA demonstrate the potential for technical ignorance to create ineffective and potentially damaging policy.

There is a computer science cryptography course that focuses on different methods of cryptography. This course focuses on cryptographic systems and protocols, reductions of security protocols to known hard problems, and other formal measures of security. There is also a course in the math department that focuses on the number theory behind cryptography. Topics include modular arithmetic, symmetric key cryptosystems, public key cryptosystems, lattice based cryptography, and elliptic curve cryptography.

For advanced students, a graduate seminar on computer security was first offered in Spring 2012. This course consists of some advanced lectures, but mostly relies on student presentations of security papers. The course ends with final projects by students (some in groups) that must include some practical demo of the presented topic. Sample projects include vulnerability analysis of a new course management system being used by Brown called Canvas, extending a papers results about acoustic eavesdropping on keyboards, and malware detection.

Finally, the course most relevant to this paper is Introduction to Computer Security, CSCI1660. This course is a survey course intended for undergraduates and some graduate students. I have worked on this course for two years as the Head Teaching Assistant managing a small staff of Undergraduate TA and one Graduate TA. The staffs responsibilities include developing and delivering the course projects, writing and editing homework problems, holding regular office hours for students, and grading all assignments.

Students taking the security course have widely varied backgrounds. The course has no prerequisites besides the 1-2 semester introductory sequence in computer science. The course is not required for any concentrator. Almost every other 1000 level course requires some middle level course like software engineering, introduction to systems, or introduction to discrete mathematics. The lack of requirements puts this course in a unique position in the department, as many sophomore students take this course as their first elective. The topics covered by the course vary widely. Here is a list of topics covered. I have appended to each the other 1000 level course that this topic relates to (OS = Operating Systems):

- Basic Cryptography (Crypto)
- Hash Functions (Crypto)
- Password Cracking
- Buffer Overflow (OS)
- DoS Attacks (OS)
- ARP poisoning/spoofing (Networks)
- Wireless networks (Networks)
- DNS and cache poisoning (Networks)

- Setuid and file permissions (OS)
- XSS and CSRF attacks (Web Apps)
- SQL injection (Databses)
- Firewalls (Networks)
- Lock Picking
- Penetration Testing
- Computer Forensics (OS)
- Malware (Machine Learning)
- Data confidentiality (Crypto)
- Spam (Machine Learning)
- Public Key Cryptosystems (Crypto)
- Digital Cash
- Electronic Voting
- Airport Security

This list is quite long and covers many different topics in computer science. For example, to understand ARP spoofing, the basics of networking must be covered. To explain buffer overflows, operating systems must be explored. In fact, the basics of almost every topic covered in other computer science courses at Brown must be explained at some point during this course.

Expertise in any of these topics would aid in understanding the relevant topics and perhaps result in a deeper understanding than is expected from students with less background. All students leave the course with basic understanding of all of these topics, which also helps if they later take any of these courses.

All of these courses related to security attempt to educate students about these issues in order to make the software landscape more secure. Certainly teaching expertise in these topics is important to continuing research and development. I believe that the best way to make an impact quickly, however, is to lower the barrier to the fundamentals of this knowledge. Every computer science student should have access to a security course where the basic attacks are covered and practiced. More people need to be educated about security issues for any effective change to happen. Prerequisites and highly technical courses are great for those who take them, but limit the scope of impact.

Chapter 4

Practical Security

4.1 Tradeoffs

There are many tradeoffs to be made when designing projects for students. I will describe my general approach to these tradeoffs to help the reader understand my design decisions.

Practicality helps motivate students as well as provides them with valuable experience. However, practicality can come at the cost of over specification. In particular, designing projects that depend on certain VM instances may mean that real vulnerabilities can be exploited in a project, but if that means that only students with that particular VM (say, Windows Server 2000) can complete the project, than I believe the lessons should be made more general. Overall, practicality helps projects be interesting, but should not be prioritized over simplicity, clarity, and fewer technological dependencies.

Two of the proposed projects allow for open-ended work by the students. The projects do not specify steps for the student to go through, but instead only state the assumptions and the goals. These projects contrast with many other computer science projects which specify exactly what should be done, and ask the students to simply meet the requirements. The open-ended projects force students to be creative, use critical thinking skills, and develop an appreciation for the discovery of the procedure instead of simply execution. On the other hand, it can be incredibly frustrating and discouraging if students cannot get started or get stuck during these projects. I believe for the goals of this course, which aims to teach students how to think critically about security, such open-ended projects are necessary. The risk of having some students struggle is worth the overall benefit of this approach. Course staff should be available to give appropriate hints to help students that are particularly stuck.

Stencil code limits the amount of work students must complete for each project. Extensive stencil code helps narrow the student's task to the most salient features of the application. However, doing this also limits the student's overall understanding of the whole process. Something can be said for making students implement all parts of any application they produce for a course. For this course, however, we aim to cover many subjects at a shallow depth. Therefore providing stencil code allows students to spend more time on different subjects. For this reason I tried to provide as much stencil code as possible where applicable. This usually makes automated grading easier as well, since all students' code obeys the same input and output formats.

4.2 Practical Issues for Projects

There are practical concerns for these projects that I considered when designing the projects as well. Because of the varying backgrounds and few requirements for the course, I decided to make all of the support code for the project in the same language (Java). Having clear and simple projects decreases the amount of effort required to update, expand, and maintain the projects going into the future. Similarly, having small and modular projects allows for easier distribution of the projects to people outside of Brown.

When writing code in an unfamiliar language, there is a lot of time spent looking up syntax, style, and particular APIs. Students came in to the course with varied experience with particular languages, so there is no way to make the projects use a language that is familiar to everyone. I decided that Java was a good choice for the projects since the department has many Java-centric courses, and I was most familiar with Java so I could write the highest quality support code. The first project requires few lines of code and no complicated coding paradigms, so students have a fairly painless introduction to Java. The second and third projects continue to use Java in parts of the assignment, although techniques using other languages are sometimes encouraged.

As an added advantage, the support code for these projects serves as good examples of writing Java code, and engineered code. Again, since students taking this course have not necessarily taken any software engineering courses, they have not often encountered the source code to programs doing fairly complex tasks like network communication, file system operations, encryption, and user interfaces. The support code handles these in a well-designed fashion, dealing with these tasks modularly.

The support code also needs to be maintained in the future by the course staff, assuming these projects are reused. Since TAs for 1000 level courses are often juniors and seniors, the turnover rate for these courses is quite high. It is rare for the writer of the projects or assignments to still be working on the course. Therefore documentation and clarity of coding must be emphasized. A project with complicated setup or distribution steps can quickly cause problems if versions change, or some assumed conditions no longer apply (like a certain course directory structure). Since code that is easy to maintain is often simpler, smaller, and less intertwined with specific technologies, meeting this goal often helped in making the code more easily distributed.

Working over department administered networked VMs was the specific technology that caused problems for this course in the past. Running multiple VMs on the same machine can be hard for large VMs or less powerful computers. It is unrealistic to expect any outside party attempting these projects to have access to multiple machines and to set up a VM network. Therefore by not trying to keep this problematic technology up to date and functional, it has become feasible to have individuals outside of Brown with fewer resources be able to complete the projects.

4.3 Student Expectations

These projects are implemented with Brown University students in mind. Some students view these projects as quite demanding. Students must be able to work and solve problems independently. Students are expected to use online resources when needed, but are expected

to cite and explain any code or results obtained from sources outside of course material.

From a practical standpoint, students must be able to code in Java. Since the first project has a relatively simple coding portion, it is feasible to learn Java programming while completing these projects. Students should be comfortable with simple unix commands that may be necessary or useful for some of the projects. They should be familiar with basic algorithms and data structures (one or two semesters experience is sufficient). Basic mathematical skills are expected as well, including simple probability, big O notation, and basic modular arithmetic.

Students must have access to a computer with a few GB of free space to install VirtualBox and a Linux VM on their machine. Both of these are free and open source, but they do take up significant space. Otherwise, there are no technical requirements for students.

Chapter 5

Projects

5.1 Firewall

5.1.1 Overview

The first project is the only strictly white hat project of the four. Students are asked to code rules that would work as a dynamic firewall. The desired result is a port knocking firewall that hides the existence of the server by dropping all incoming packets, unless the client sends a packet to a certain series of ports on the server, in which case the server allows a connection from only that client for a short period. This defends against an attacker who port scans the server, since the attacker would see all ports closed, and assume the server was not responding to any requests.

To make this project more manageable and portable, the students do not work with live traffic or with real firewall rules. Instead, they are tasked only with creating the logic for the firewall, deciding whether to accept or drop each incoming packet. The students are also asked to explore Linuxs iptables, which is required to implement a functional firewall. They also must perform some basic operations with netcat, which is useful for testing and sometimes implementing network programs including firewalls. These sections are done on the VirtualBox VM. Since many of these features can be tested with only an Internet connection or with traffic over the loopback interface, no VM network is required.

Finally, students must analyze the effectiveness of their port knocking firewall. In particular, there are several open-ended questions asking about issues like eavesdropping, IP spoofing, and denial of service attacks that require the students to understand the strengths and weaknesses of their approach. They also must consider possible improvements or more creative attacks.

5.1.2 Drawbacks

To make this project feasible without a VM network, the final product is not a functional firewall. It uses an interface that allows input from live traffic, and the packet decisions could be used to modify real firewall rules using iptables, but this is not part of the project. As a result, there is less overall satisfaction with a completed project, and the resulting code is not actually useful.

Additionally, as the analysis questions have the students discover, the firewall is not a particularly good one. It should only be used in conjunction with other network security, as it does not diminish the security of other measures (aside from risk compensation). Improvements to a basic port knocking firewall would go beyond the scope of the course this early in the semester, and would not provide much more educational value, since the concept is the same with more complicated implementation. For example, making the port knocking sequence time-dependent by hashing the known sequence with the current time would help prevent replays, but coding that solution would not provide more insight than simply stating the protocol.

The sections about iptables and netcat are fairly vague, which I believe is an advantage. The current directions ask students to use the tool in one specific way, and then to implement anything else that is interesting. In this first version of the project, students did not do a great job in exploring either of these utilities. The handout should better specify the depth to which they should explore these tools functionality without giving them a specific task.

5.1.3 Benefits

By writing code that only deals with the logic, testing becomes trivially easy. Students are given sample input packets that are stored in a file. They can much more easily test scenarios like concurrent connections, exact timeouts, and long delays when running on static input as opposed to creating live traffic. Otherwise, since clients are identified by IP address and the server must run in a VM, it is difficult to quickly execute meaningful tests. When run on local test files instead, the staff can easily test that the students firewalls meet expectations by running the students code against a test suite with expected output. The students can also write their own tests before even starting the project, so they know what features are left to implement and know when their firewall works as required. This is a perfect example of test-driven development, which is a valuable technique to see in action–writing simple and modular projects leads to unintended lessons in software engineering.

As mentioned in the drawbacks, the port knocking firewall does not provide adequate security. This is not necessarily a problem with the project. Students need to understand that although a security measure provides protection against some attacks, such as port scanning or blind brute forcing, there may be simple and realistic workarounds for an attacker. For instance, an eavesdropper on the client or server side can easily bypass this security by reading a legitimate clients network traffic. This provides a valuable lesson about security: securing against certain attack vectors does not secure against all attackers. Threat models must be properly considered.

5.1.4 Conclusions

The project has a lot of strengths for a first project. It requires relatively little coding, it teaches a relevant subject, and the student must perform some creative analysis. The staff can grade the functionality of the assignment automatically. Although the student codes from a white hat perspective, the analysis questions give insight into how one would break the protections afforded by the firewall. Wireshark, netcat, and iptables fit well into this assignment, so these important networking tools can be introduced or not depending on the desires of the course staff.

Many extensions could easily be added to this project. Students could extend the project to be a functional firewall if they have sufficient infrastructure to test it. Numerous additional security measures could be implemented, including cryptographic improvements on the protocol, time dependent knock sequences, and one-time use knock sequences. Similarly, a number of exploits could be demonstrated, including eavesdropping, denial of service to one client or the whole server, and session stealing.

5.2 FileTransfer

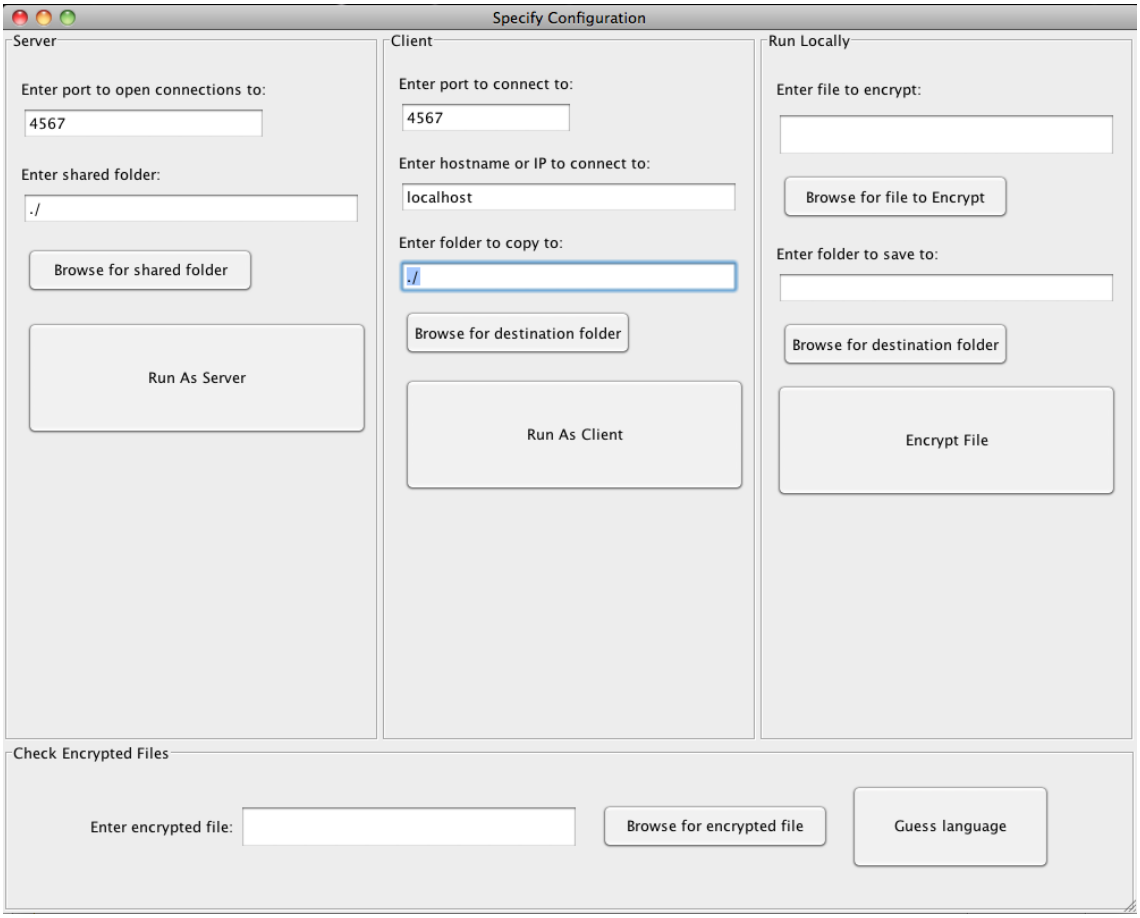

Figure 5.1: The main FileTransfer window

5.2.1 Overview

The second project asks the students to bypass security measures in a sample encrypted file downloader. This project relates to DRM protections, but is really about insecure software patterns and creative hacking. The assignment is quite open-ended. Students are given an application called FileTransfer that downloads files from a server. The downloaded files are

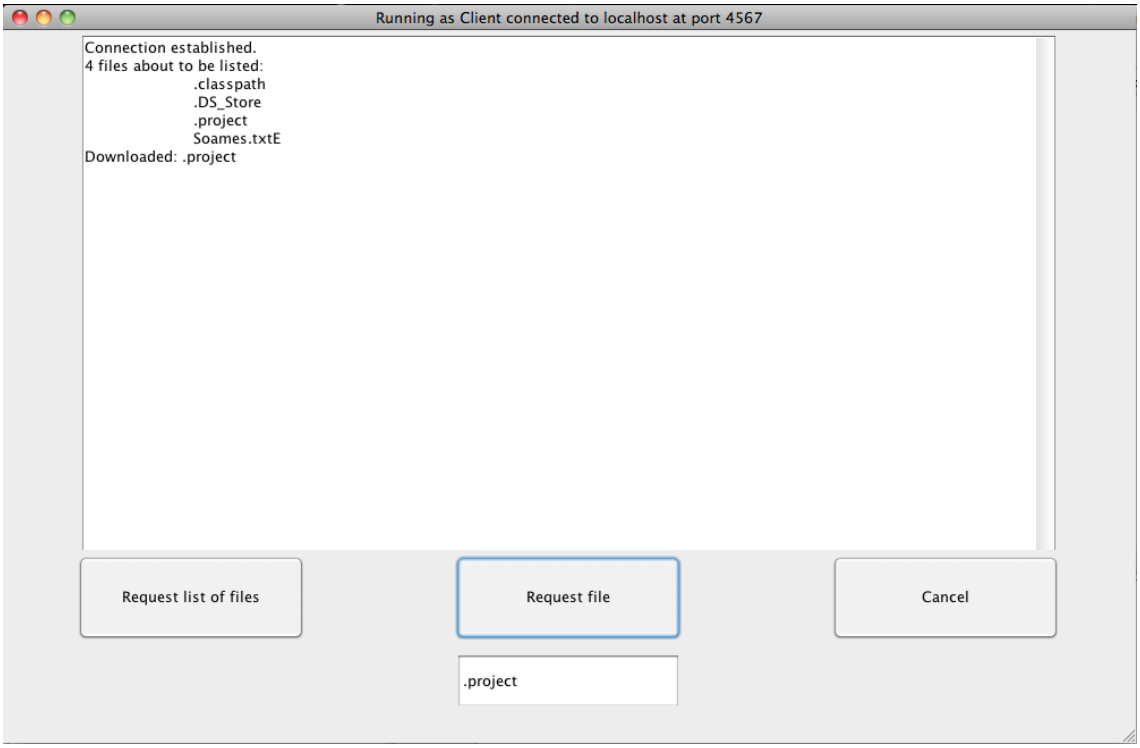

Figure 5.2: The FileTransfer in client mode after downloading a file

encrypted and stored on the client side. The application itself can read the encrypted files and likely determine which language the file is written in. This is a similar situation to a protected music file that can be downloaded from a central server and saved locally but can only be played from the designated music player.

Students are asked to find four distinct methods for getting the unencrypted files from the server. The given application has several severe security flaws that students can exploit. First, they must explore the program to see how it works and identify potential attack vectors. Then they must implement of these attacks. Little other information is given, except some information that is supposed to be public knowledge for many applications, like the method of encryption used. Students are given the code used for encryption/decryption, but not the secret key. At least 7 distinct attacks exist, and at least 5 are easily implementable by a student of any basic security course. I will decline to discuss specific vulnerabilities or attacks as this project may be reused in the future.

5.2.2 Drawbacks

This project is designed around secrecy. Some exploits and methods for inspecting the program are easy to implement but hard to think of. Therefore students who discuss their progress or techniques with other students may give away a large portion of the project. The project is valuable because students have to use their ingenuity to get enough information to write exploits. This also demonstrates the differences between attacking a closed-source

application and an open-source equivalent. It takes significant effort to even discover how the application works without being able to directly examine the source code.

5.2.3 Benefits

Students have to attack an executable Java program. They are not told how it works or how it was written at all. They have to break it to their advantage. The setup for this project is interesting to students. How much information can you get about how the programs running on your computer work? This is a problem rarely discussed in other courses, but very important for engineers.

Students are encouraged to look online for tools that may help them. This serves as a great opportunity for students to see the community around security software for research. There are numerous tools available for security analysis, penetration testing, network analysis, etc. that students may come across for this assignment.

The application that they break has many glaring security holes, but none are put in specifically to be vulnerable. Basically, it was written with no concern for security but no particular vulnerabilities are built in. It serves as an example of an insecure application that looks acceptable at first glance. The application is a great example of nave engineering that ruins the intended purpose of the application.

5.2.4 Conclusions

Since students have to do some much exploring and investigation on their own, the project can be incredibly fun or potentially frustrating for students. Some students have gotten stuck right at the beginning since there is so little explicit instruction. Similarly, once a couple exploits have been found, finding a third or fourth can be problematic for some students. Its hard to strike a balance between wanting students to have to think critically about the problem at hand and simply frustrating them. I think the current version of the project is reasonable for a course that has covered the necessary topics.

The project is a great example of my proposed philosophy for security education. The project gives students examples of the mistakes they might make by letting them discover those mistakes in a real application.

The actual implementation of this project could change depending on the target students. The application could be more obfuscated than compiled Java code to prevent any source code reading. A second phase where students must secure the application is also possible.

5.3 FiveMinuteHacks

5.3.1 Overview

The third project asks students to think about the consequences of unintended access to a users account. In particular, students must think about what an attacker can achieve with brief access to an account in the context of finding a user already logged in but away from the computer. Information stored locally, such as private keys or sensitive unencrypted

documents are obvious targets. These files, however, might be hard to find and copy without arousing suspicion or risking the real user returning.

Therefore the attacker may want to compromise the account for later remote access. With brief access, an attacker may also try to infect the machine with some kind of keylogger or password stealer. For this project, students are asked to write and implant a fake version of the su utility that logs the users password and interaction as the root user. With this kind of access other important information like hashed passwords may also be recovered, with or without the aid of tools like memory dumps or disk inspection. Students must also implement the important methods making up a rainbow table to invert hashes of a known password space to potentially learn the password of the victim.

Finally, as a less technical component of the project, students investigate their own machines to discover what kind of sensitive information would be available to an attacker with limited time at their machine. For example, students investigate their browsers history, cookies, cache, and saved passwords to see what information can be gained without entering the users password. They check for temporary files, the download folder, and any other locations that may contain personal files that are not often cleaned. Students note their findings as part of their report.

5.3.2 Drawbacks

Part of the project involves writing backdoors into the system. Students have to write an executable file in a public location to get a shell with their victims credentials. This only makes sense if both the attacker and victim share an account on a networked or shared filesystem, which may not be applicable in many instances. They are also asked to write a backdoor via a network connection, which is much more realistic.

The rainbow table they write is generic enough to work on large password spaces, but the project requirements do not require them to build a practical table since it would take a long time to generate and require a lot of storage. As it is, the students must only generate a rainbow table that can look up four character lowercase alphabetical passwords, which is quite a small space.

In terms of programming languages, the fake su portion has the drawback that it is easiest to implement using Python. This breaks the previous advantage of the course only requiring the students to use of Java. Fortunately, a proper implementation of this in Python requires very little code and no language specific patterns.

Finally, since this project has students implement some very real attacks, the course staff has to make it clear that these attacks are taught only for educational purposes, and implementing such attacks is not acceptable by university policy or the law.

5.3.3 Benefits

Overall, this seems like the most satisfying project for students. They get to implement real attacks that they could actually launch against peers or strangers if they could get brief access to their machines.

The rainbow table portion provides an opportunity for creativity as well. Students must not only implement a rainbow table, but they must improve it somehow. There are several options available for students, including an option for them to improve it in their own

way. The first option asks students to experimentally measure the theoretical properties of a rainbow table. They must demonstrate the relationship with increasing chain length, number of rows, and the password space on storage space, password coverage and password lookup time. This choice of extension requires good analytical skills as well as relating the practical to the theoretical. Secondly, students may make their tables multithreaded, speeding up table creation and lookup time. This allows students to create much longer chains, since lookup time increases with no storage drawbacks. This option emphasizes the systems and engineering component of security programming.

Thirdly, students may investigate reduction functions. They are provided with a reduction function in the support code, but it is not very efficient (in fact it uses a hash function itself). Students must explain the given reduction function as well as writing their own and showing that it is an improvement. This option relates to the more abstract and mathematical basis of security.

If none of these choices appeal to the student, they are allowed to think of their own improvement as long as it addresses an important component of rainbow tables or security programming in general. There are a number of such improvements, like improving password coverage through some kind of memory or heuristic to prevent overlapping chains. The reduction function could be modified to fit more realistic password spaces, for instance combinations of English letters that can be pronounced (i.e. biasing password coverage towards strings that forbid many consonants in a row). Unfortunately, no students chose to do their own improvement this first year giving the project.

I think the non-technical portion of the assignment is a great exercise for everybody to do, not just security students. Most people assume that their personal data is secure to the outside world. They rarely consider how secure their information is to anybody who can snoop around their computer for a couple of minutes. Sit in a computer lab or library long enough and you will certainly see opportunities to do exactly that. Many people do not realize that their saved passwords in Chrome and Firefox can be viewed in plaintext without entering a password. Fewer still realize that just deleting a browsers history does not delete all the cookies associated with the website, so such an attacker can see their browsing habits easily and quickly. Still more users would be surprised on the presence of their credit card numbers, social security numbers, or plaintext passwords in files on their file system.

This portion is not supposed to scare students. The bigger issue here is what level of access people want to secure their data against. These questions are really trying to get the students to think of threat models. A system is only secure against a threat, and some people do not wish to secure themselves against an attacker with physical access. It is a reasonable risk to take, but it should be one taken intentionally rather than accidentally.

5.3.4 Conclusions

This project is the only project completely conceived from scratch for this year. I think it really addresses the main issues I want to convey with the projects for the course. Students must confront their personal security policies. They get practical experience with what they can do once a machine has been compromised. Often, security exercises are designed to be proof of concept, where once any access can be achieved the attack is deemed successful.

I believe students need to understand that once an attacker gets access to create a single file, or run a single command, as a user, that the users account may be compromised permanently. By opening up a networked backdoor, the machine could be vulnerable to an attacker until the process is noticed and stopped manually, which would rarely happen.

Finally, from a pedagogical standpoint, the rainbow table portion teaches many valuable lessons. The students see another well-engineered project in Java where code is broken up so they only have to implement a few key methods to have a functioning rainbow table. The extensions allow students to investigate any one of a number of relevant roots of computer security, which can either be a strength or interest of the student. It even leaves room for creativity in all of the extensions, since it is never specified exactly how to make the improvement, only what must be improved.

5.4 FirstClass

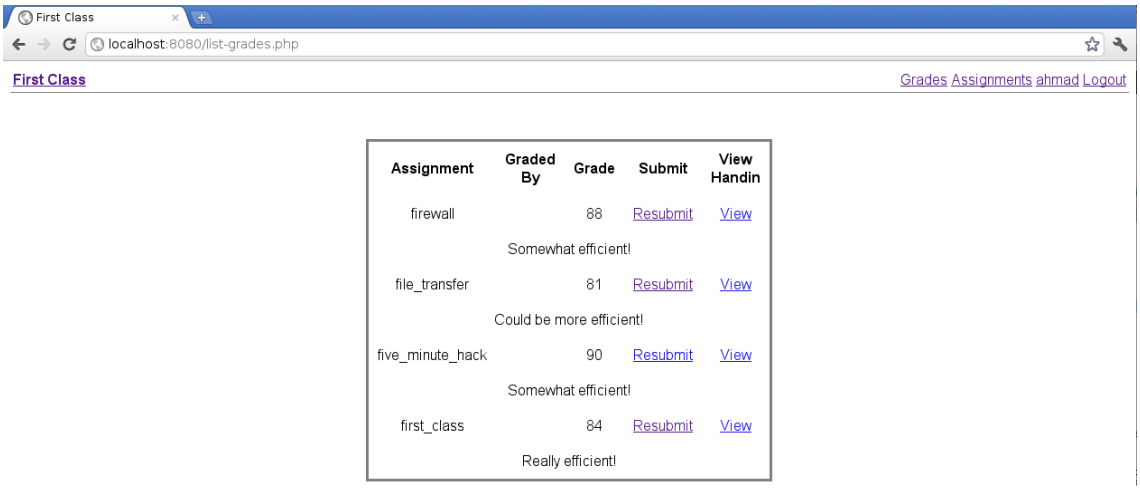

Figure 5.3: Viewing grades in FirstClass

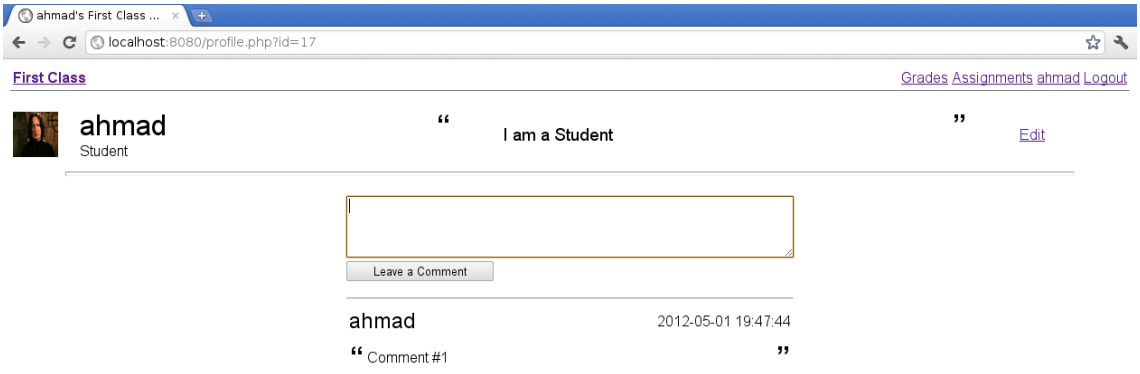

Figure 5.4: A profile page in FirstClass

5.4.1 Overview

Any survey course in computer security course has to cover web vulnerabilities. The last project does just that in the most realistic scenario possible. Students are given access to a basic account on a course management web application and must act with unintended privileges. The management site is written with simplicity in mind, since students have little experience with web development or web languages like HTML, JavaScript, or PHP. Therefore the site has little formatting or UI design, but focuses on the core functionality.

The assignment is intended to give little information away that could not be gathered about a real application. Because all the students for this course have access to the universitys networked file system, the course staff designed a system that lets students create lightweight copies of the web application such that they can run and modify their own versions locally without being able to read the source files of the server. This allows students do damage to the server since they can always get a fresh copy of the undamaged server on demand. If no such infrastructure is available, an honest student can still do the project without reading the servers source, or even with full access to this information. The difficulty of the project changes and it becomes less realistic if the students are allowed or encouraged to read the source, but the exploits still need to be discovered and implemented.

To explore the application and implement the exploits, students are given a handful of unprivileged student accounts that can simply browse other users public information, make discussion posts, post homework solutions, view their own grades, and update their own profile. Administrators can edit and view quite a bit more, but students are not given access to any administrator accounts. Instead, they must find exploits to gain access to administrator accounts or otherwise perform forbidden actions.

The handout details certain assumptions the students can make while developing their exploits. The site has to be in use by students and administrators, they assume that design decisions are made as expected (i.e. no weird hash functions are chosen instead of the most common ones), and they are told that admins can be tricked in clicking on reasonable looking links.

Similarly to FileTransfer, many exploits are intentionally left in the application. At least seven exploits that students could discover exist, and they must find five to get full credit for the assignment. They must also include a brief description of a fix for each exploit, or an assertion that the exploit cannot be fixed.

5.4.2 Drawbacks

The specific exploits left in are quite easy to fix. Therefore, it is unlikely that a widely used web application would have these specific exploits. However, with only small modifications, some such exploits do exist on very popular web applications. These modifications tend to be specific to web technologies like html encoding or browser specifications, which make the modifications difficult to expect students to discover. Instead, as this is an introductory course, the exploits must be simple examples of broad families. This is a drawback, but also makes the assignment more feasible and less frustrating for students. Furthermore, students with varying backgrounds all have the opportunity to discover these exploits.

The lack of direction in the project could lead to some frustration. Some of the exploits inevitably deal with specific web exploits that students have never practiced. A simpler

approach where the vulnerability and target is specified but the students have to implement the exploits would be easier, but would not fit with the course objectives. Discovery of exploits is an important step towards understanding the source of vulnerabilities. Students should be forced to think like an attacker without knowing exactly what steps should be taken.

Not all users who try this project can hide the source code with clever permissions and setuid scripts. This goes against the course goal to make these projects well suited to wide spread distribution, but the project does not need this feature.

5.4.3 Benefits

This project lets students practice attacks that hit the most easily accessible targets: web applications. Since web applications intentionally allow user interaction and user generated content, they are particularly vulnerable to attack. Students will gain an understanding of how unexpected or unusual user input can have detrimental effects on a system. Access controls are implemented, but done so poorly enough that the user can easily escape these restrictions. Access controls and models for roles is a whole topic within security that students are exposed to in this context. This does not happen if engineers think like malicious users as they design a system and write code.

Students are able to see how design decisions that at first glance seem necessary and obvious can end up with the worst vulnerabilities. Allowing image hosting, file upload, and discussion posts are important features but also serve as fantastic attack surfaces. These vulnerabilities can be found in many real world applications as well, but usually with insufficient protection instead of this application that has no protection.

This project has a very realistic setup. Any web user has access to hundreds of thousands of web applications in browsers and mobile platforms. All of these applications have broad attack surfaces and certainly contain vulnerabilities. This project provides a simple example of a vulnerable application and forces students to actually implement attacks instead of just discovering them. In typical security research, many vulnerabilities are demonstrated by getting the browser to execute the JavaScript alert(1), which is not as conrete as actually stealing a users cookie or acting as an administrator.

5.4.4 Conclusions

This project has many benefits over the other projects. Web vulnerabilities are the easiest to test, demonstrate, and discover in the real world. Students write functional exploits. They have to use their creativity and ingenuity to discover exploits. The project provides little unreasonable knowledge or support that a real attacker would not have about a web application. The project is easily defined and left up to the students.

It serves as an excellent final project. For a course with more time, or for independent students, it may be worthwhile to have the students secure the code as a follow-up assignment. This would require more knowledge of web technologies, but is definitely feasible with the help of online tutorials and code snippets. For some courses, it may be possible to have students attack each others secured web servers as a fun exercise.

The provided application has basic functionality and fairly basic exploits. It would be possible to extend the server with more protections to create a more challenging project.

For example, a better cookie system could lead to more interesting attacks involving cookie modification or stealing. Offering some XSS protection would force students to do more work to have the server execute JavaScript.

Chapter 6

Conclusions

6.1 Results

It is difficult to measure the success of these projects in a quantitative way. We do collect mid-semester surveys from students. These survey results indicate nearly universal approval of the first two projects. An equal number of students complained that Firewall and FileTransfer were too easy and too hard, which indicates the difficulty level is roughly appropriate for students. The grade distributions for these projects are in figures 6.1 and 6.2. The new projects all have wider grade distribution which helps to differentiate students. Note that the direct comparison is difficult since the projects cover similar material, but have different requirements and different rubrics for grading, as well as different students and mostly new course staff.

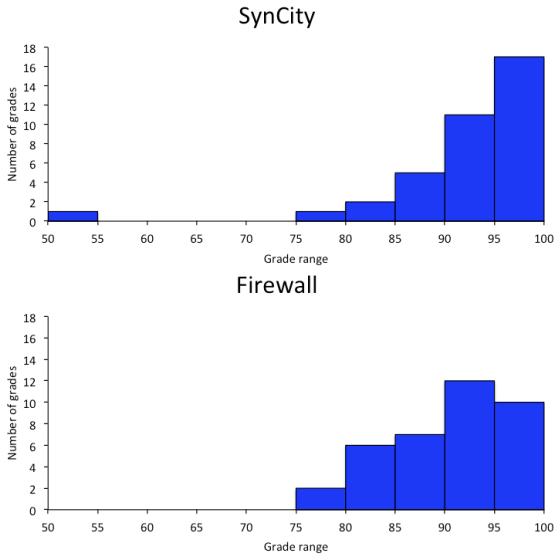

Figure 6.1: Grade distributions for Firewall and its predecessor SynCity

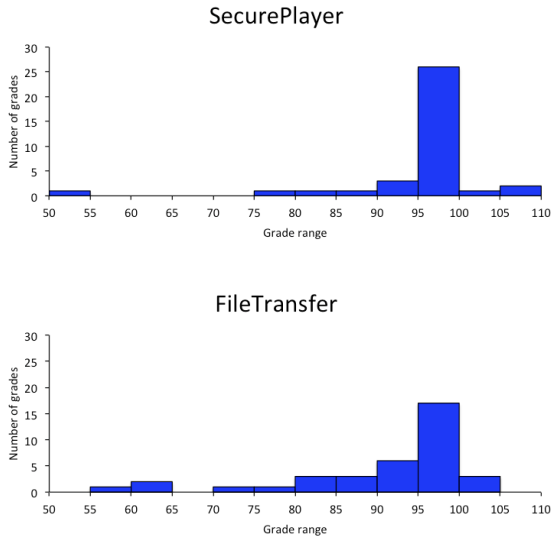

Figure 6.2: Grade distributions for FileTransfer and its predecessor SecurePlayer

6.2 Conclusions

Cyber security education is lacking. Security experts have a good grasp on what systems are secure, what protocols need development or change, and other large scale problems with our information infrastructure. These problems require fundamental changes to these old protocols and existing technology.

On the other hand, there are many vulnerabilities that exist purely from programmer ignorance or error. As a part of an eduction institution, we can make the largest impact in this realm. Programmers have a complex job to make their products scalable, functional, useful, and effective. Unfortunately, an incredibly effective product with security flaws may still be successful. When companies prioritize product quality before security, vulnerabilities are bound to appear. Moreover, their selection of engineers will not focus on their security credentials.

This report proposes a new perspective on producing code. Code must be secure. The risks for companies are large but will only continue to grow if security holes remain in applications. Writing secure code is complicated, but not impossible. Making wide scale changes or audits of existing code is impractical. Instead, we have to work on producing higher quality code in the future, and over time these preventable vulnerabilities will decline.

The projects proposed seem effective in reaching these goals. The largest change from previous versions of these projects is their lack of dependencies. By making these projects depend on less resources specific to a well funded organization like Brown University's computer science department, it makes these lessons accessible to more students. Open courseware has been on the rise in the last few years, and development of course materials should facilitate this growth.

Security educators need to think more about their audience and their overall goals. Secuirty as a specialty needs to remain, but not be the only option to learn about this

topic. Security protocols usable as a black box like developing cryptographic protocols and designing secure networks should not be forced upon students not interested in these topics. The practice of exploiting and preventing cross-site scripting and SQL injection, however, is applicable to anyone writing code on the web.

Engineers have responsibility for the products they produce. Educators have a responsibility for the engineers they produce.

6.3 Acknowledgments

I would like to acknowledge Roberto Tamassia and Bernardo Palazzi, the professors of the undergraduate and graduate security course at Brown University. I would like to further acknowledge Professor Tamassia for being a very helpful thesis advisor as well as having the foresight to create this course when few other similar courses existed. I give my thanks to John Savage as well for serving on my thesis committee.

I would also like to thank the previous course staffs of the security course for developing the original versions of many of these projects, in particular Saurya Velagapudi. I could never have completed this work without the course staff that has worked with me over these past two years, in particular the graduate TA James Kelley, and undergraduate TAs Dave Kilian and Douglas McErlean who wrote the source for FirstClass, and John Boreiko for testing these projects. I believe the strength of Brown's computer science program lies in the TAs, and these individuals are all proof of that.

Bibliography

[1] Belk, Mark et. al. *A Guide to the Most Effective Secure Development Practices in Use Today* SAFECode, February 8, 2011.

[2] Dhamija, Rachna Tygar, J.D. Hearts, Marti *Why Phising Works* Conference on Human Factors in Computing Systms, April, 2006.

[3] Florencio, Dinei Herley, Cormac *Sex, Lies and Cyber-crime Surveys* Microsoft Research Blog, June 2011.

[4] Frei, Stefan et. al. *Modeling the Security Ecosystem- The Dynamics of (In)Security* WEIS 2009, June 24, 2009.

[5] Goldman, David *The Cost of Cybercrime* CNN, July 22, 2011.

[6] Kaelin, Lee *Anonymous hacks Syrian President's email, reveals weak password* techspot.com, February 8, 2012.

[7] Koscher, Karl et. al. *Experimental Security Analysis of a Modern Automobile* IEEE Symposium on Security and Privacy, 2010.

[8] Rosenberg, Dan *Introduction to Kernel Exploitation* Virtual Security Research, August, 2010.

[9] RSnake *XSS (Cross Site Scripting) Cheat Sheet* ha.ckers.org.

[10] Schoemaker, Jeremy *How I hacked your Facebook account* ShoeMoney blog, March 8, 2011.

[11] Zetter, Kim *Chrome Owned by Exploits in Hacker Contests, But Google's $1M Purse Still Safe* wired.com, March, 8, 2012.

[12] *Flash to Focus on PC Browsing and Mobile Apps; Adobe to More Aggressively Contribute to HTML5* adobe.com, November, 2011.

[13] *Google Chrome and Browser Security* google.com, 2010.

[14] grsecurity technical specifications, http://en.wikibooks.org/wiki/Grsecurity.

[15] *Reused Login Credentials* Trusteer Security Advisory, February 2, 2010.

[16] *Secunia Yearly Report* secunia.com, February 14, 2012.

[17] *Sony Corporation FY2010 Consolidated Results* May 23, 2011.

[18] *Thoughts on Flash* apple.com, April, 2010.

[19] *Vulnerability Reward Program* google.com, November, 2010.