Integrating Web Application Security into the IT Curriculum

James Walden
Dept. of Computer Science
Northern Kentucky University
Highland Heights, KY
waldenj@nku.edu

ABSTRACT

Attackers are increasingly targeting web applications. Buffer overflows had been the most common vulnerability type since CERT began collecting statistics, but web application vulnerabilities like cross-site scripting have dominated vulnerability reports since 2005. Despite billions of dollars spent on network security, the amount lost to computer crime, much of it the result of the insecurity of web applications, grows every year. In part, this problems results from the fact that perimeter security techniques like firewalls do little to protect web applications.

In order for students to be prepared for the current threat environment, we need to integrate web application security into the IT curriculum. Both information security and web programming classes need to cover this topic. This paper describes techniques, tools, and labs for integrating web application security into both types of classes. Some techniques, such as penetration testing using web proxies, are applicable to both types of classes. Other techniques, such as secure programming guidelines, are primarily useful in web programming classes, while some tools, like web application firewalls, are more important in information security classes.

We use the open source web application security teaching tool WebGoat for introductory labs that teach the students about the nature of specific vulnerabilities like SQL injection. These labs also introduce students to open source web testing proxies, such as Burp Suite, which they use more deeply in later labs that focus on penetration testing of a complete web application. Students in security classes also learn how to use web vulnerability scanners and web application firewalls, while web programming classes focus on learning how to write code without common vulnerabilities.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; C.2.0 [**Computer-Communication Networks**]: General—*security and protection*

General Terms

Web application security

Keywords

web application security, web security education

SIGITE'08, October 16–18, 2008, Cincinnati, Ohio, USA.

1. INTRODUCTION

As security at the network perimeter has improved, attackers have transferred their efforts to the easier target of web applications. Web applications are at the core of our electronic banking and commerce systems. Web applications, like Google Docs, are also beginning to replace traditional desktop applications. While the importance of web application security is growing, the quality of web security is declining.

The 191% increase in the number of security vulnerabilities from 2004 to 2007[2] is largely attributable to vulnerabilities in web applications. Vulnerabilities found primarily in web applications, such as SQL injection, make up four of the five most common types of vulnerabilities. Cross-site scripting became the most common vulnerability in 2004, displacing buffer overflows for the first time[15].

Network security techniques like firewalls and cryptography are of limited use for defending web applications. Organizations have to allow network access to web applications so that customers can use them to bank, shop, or otherwise interact with the organization. The administrator cannot block network connections at the firewall, and encrypting the connection to the web application with SSL does nothing to prevent an attacker from launching a cross-site scripting or SQL injection attack.

Web applications need to be defended at the application layer, either by designing and coding an application that has no vulnerabilities or by protecting the program with a tool such as a web application firewall. However, few web applications are designed with security in mind, and it requires extensive customization to configure an application firewall to effectively protect an application.

In addition to being less well-protected, web applications typically have assets that attackers desire. While credit card numbers, bank accounts, and other personally identifiable information (PII) are the most common targets, web sites are increasingly being used to distribute malware to unsuspecting users. Compromised sites appear unchanged to their

users, but serve malware using embedded iframes, which are normally invisible to the user. Websense reported that 51% of malicious sites found in the second half of 2007 were legitimate sites that had been compromised by attackers[18].

Most security curricula focus on network layer and operating system security, rather application security. There are few textbooks that discuss secure programming techniques, and the code shown in many programming textbooks contains common vulnerabilities like cross-site scripting and SQL injection. Given the dangerous environment in which web applications operate, students need to have a solid understanding of the threats to those applications and how to mitigate them as software developers and as system administrators.

Web application security needs to be addressed from both the perspective of the web developer and of the web administrator. In the IT2005[1] curriculum, web security should be covered from the developer's perspective in topic IPT5, software security practices. It can be covered from the administrator's perspective under three topics: WS5, Web Security, IAS6, Security Domains, or IAS11, Vulnerabilities.

This paper outlines the topics that need to be covered from both perspectives and describes laboratory assignments that will help students learn how to find and remediate security vulnerabilities in web applications. The remainder of this paper is organized as follows. Sections 2 and 3 present the goals and topics to be covered. Laboratory exercises are discussed in section 4. Section 5 describes challenges in integrating web security and feedback received from students, while section 6 expands on future directions for integrating web application security into the curriculum.

2. GOALS

The primary goal for a unit on web application security course is for students to be able to deploy web applications that function correctly even when under attack. In a web programming course, this goal will result in a focus on writing code without vulnerabilities, while in an information security course, the focus will be on configuring the application and its environment to prevent the exploitation of any vulnerabilities in the code. Both approaches are necessary, as no nontrivial application will be devoid of vulnerabilities and no amount of configuration will render a vulnerable application immune to all attacks.

Upon completion of a unit in web application security, students should be able to

1. Identify and explain common vulnerabilities found in web applications.
2. Explain the security implications of client-side technologies, like ActiveX and Javascript.
3. Detect security vulnerabilities in web applications using appropriate tools.
4. Design and implement web applications that do not contain common vulnerabilities.
5. Deploy and configure a web application in a secure manner.

The fourth goal is only appropriate for a web programming course, while the fifth goal is important to both information security and web programming courses. If web security topics are covered in both types of courses, the last goal should be covered in the administration course.

3. CONTENT

Determining the right mix of content can be challenging and depends on the background of the students and what other topics the course covers. Course materials can be selected from a variety of materials, including professional books like the *Web Application Hacker's Handbook*[3] or the Open Web Application Security Project (OWASP) Guide[17]. The OWASP Top 10[10] web application vulnerabilities is a good place to start deciding what topics to discuss.

Before learning about web application security, students need to have a background in web technologies, including both client-side and server-side functionality. They need an in-depth understanding of web input, including how HTTP works and the many different types of encodings, such as URL encoding, HTML encoding, Base64, and other techniques. They also need to understand how easily client-side controls like hidden form fields and input validation can be bypassed using a proxy.

A detailed topics list follows, including topics appropriate for both web programming and information security courses. A course in web application security would cover all of these topics, while an information security course would skip the secure programming topic.

1. Web Application Input
   a. HTTP Requests
   b. URLs
   c. Cookies
   d. Form Parameters
   e. Encodings
2. Client-side Technologies
   a. Javascript Security
   b. Java Security Model
   c. ActiveX Trust Model
   d. Bypassing Client Security
3. Input-based Attacks
   a. Path Traversal
   b. Web Testing Proxies
   c. Vulnerability Scanners
   d. Input Validation
   e. Blacklist Filters
   f. Whitelist Filters
4. Injection Attacks
   a. Interpreters and Injection
   b. SQL Injection
   c. Shell Injection
   d. File Inclusion
   e. XML Injection

f. Separating Code and Data

5. Cross-Site Attacks
    a. Stored Cross-Site Scripting
    b. Reflected Cross-Site Scripting
    c. HTTP Header Injection
    d. Cross-Site Request Forgery
    e. Output Encoding
6. Authentication
    a. Basic, Digest, and Form Authentication
    b. Passwords
    c. SSL
    d. Session Management
7. Secure Programming
    a. Secure Design Principles
    b. Avoiding Common Vulnerabilities
    c. Platform-specific Vulnerabilities
    d. Code Review Techniques
    e. Automating Web Tests
8. Operational Security
    a. Secure Server Configuration
    b. Web Application Firewalls
    c. Web Server Auditing

The first topic provides an essential foundation for understanding the following topics. Topics three through six address the first of the goals, while topic two addresses the second goal about client-side security. The third topic also addresses the third goal of detecting security vulnerabilities with appropriate tools. The seventh topic is directly tied to the fourth goal of writing secure web applications, and the eighth topic is directly tied to the fifth goal of secure deployment.

Since there is limited time in existing courses, it may be impossible to address all of the relevant topics. In this case, it is best to focus on the topics which have the largest impact on web application security today: topic one and topics three through five, which address the security issues arising from web application input, including the three most common vulnerabilities in applications.

4. LAB DEVELOPMENT

Labs provide students with an opportunity to apply their learnings and are also important for engaging student interest. Students cannot understand web security without applying the tools and techniques described in lecture, so each topic in lecture should be accompanied by a lab covering that material. Labs can also be used to cover variants of vulnerabilities and tool functionality that are not discussed in lecture. The lab assignments are described briefly in Table 1. While the labs could all be offered in a single class on web application security, they were divided between information security, web programming, and software security classes in our curriculum.

Lab	Description
Lab 1	WebGoat exercises on specific vulnerabilities.
Lab 2	Using a testing proxy to solve more advanced WebGoat exercises.
Lab 3	Assessing an application using a web vulnerability scanner.
Lab 4	Assessing a web application using a testing web proxy.
Lab 5	Deploying a web application firewall.
Lab 6	Reviewing the code of an application using a static analysis tool.
CTF	Participating in the international Capture the Flag web security contest.

Figure 1: Laboratory Exercises

Web application security labs are particularly effective at motivating students to understand the danger of web vulnerabilities. Many students have difficulty appreciating the danger of such vulnerabilities until they experience how easy it is to construct an exploit themselves. While it takes hours or days of effort for students to construct an effective buffer overflow exploit, they can build a SQL injection exploit in minutes. After such assignments, at least one student each semester reports discovering vulnerabilities of the same type in their own code at work. Of course, it is important to emphasize the ethics of using web security assessment skills.

As with other types of security labs, web security labs need to be conducted in an isolated environment. While an isolated network lab is one possibility, virtual machines are a more versatile alternative. VMWare and most other virtual machine software provide host-only network options, which isolate virtual machines from the physical machine's network. The primary advantage of using virtual machines is that they allow students to do lab assignments in any university computer lab or on their own machines at home. This advantage is particularly important for labs that are too long to complete during a class session.

Virtual machines provide an identical environment on every machine, which is ideal for deploying web applications. It can be difficult and time-consuming for students to get a web application running on their machine, due to the necessity of installing and configuring each component of the web environment, including a web server, database server, and language runtime. While constructing a virtual machine image is a time-consuming task, images are reusable and often take less time to construct than the time required to support students running on a wide variety of platforms.

4.1 Tools

Students need to learn to use a variety of web security tools, including web testing proxies, vulnerability scanners, and web application firewalls. Web proxies and vulnerability scanners are typically used by security auditors to evaluate the security of an application, while web application firewalls are used to block dangerous input. Freely available open source tools exist in all three of these categories. A web security unit in an administration course can use all three types, while a web security unit in a web programming course will find web proxies essential and the other classes of tools optional. A web programming course can

benefit by the introduction of non-web specific tools, such as static source code analysis tools to detect vulnerabilities before deployment of an application.

Web proxies are primarily used for security assessment of a web application. Testing proxies typically run on the same machine as the web browser and offer the ability to edit web inputs, including form parameters, cookies, and HTTP headers. They range from simple tools that can only intercept requests and edit inputs, such as the Tamper Data[5] extension for Firefox to integrated suites of web testing tools, such as Burp Suite[13], Paros Proxy[12], and Web Scarab[11]. The integrated suites typically include tools designed to spider web sites, decode data, analyze session identifiers, and automatically create and analyze HTTP requests containing user-defined input for testing purposes. While the integrated suites provide more features, Tamper Data has the advantage of being a small Firefox extension that can be quickly installed without needing administrative privileges.

Vulnerability scanners allow auditors to test a large number of web pages in a short amount of time. These tools spider a web application, then submit a series of test strings to each input that is discovered. The responses are analyzed for signatures of common vulnerabilities. While scanners can detect simple cases of some web vulnerabilities, they do not have any understanding of the design or code of the web application and are not a replacement for an experienced tester. It's important for students to gain an appreciation for the advantages and disadvantages of such tools.

If the application does not produce an overt response to one of the scanner's probes, a vulnerability will not be detected. Scanners cannot detect most access control problems or application logic flaws. Flawed input validation routines may be good enough to block the limited set of test strings used by the scanner without being good enough to block an experienced tester with access to the source code. Scanners usually cannot discover all of an application's inputs, so they need additional manual help to test the entire application. The best vulnerability scanners are commercial, but a variety of open source vulnerability scanners, such as wapiti[16], are freely available to use.

While application layer firewalls can defend an application from common attacks against vulnerabilities like SQL injection out of the box, they need to be configured to meet the specific security needs of the application. The attack surface of a web application is more complex than that of the network perimeter, so secure configuration of an application firewall is a more difficult task than configuration of a network firewall. Network firewalls deal with a relatively small number of simple entities, such as IP addresses, port numbers, and TCP flags. Web application firewalls need to deal with every type of input that the web application can accept, such as names, credit card numbers, session identifiers, and so forth. While each type of input has its own syntax and semantics, all input must be filtered by the web application firewall as text strings included in an HTTP request.

Like network firewalls, application firewalls can be implemented in hardware or software. A wide variety of products exist, but the web application firewall mod_security[14] for the Apache web server is the most widely deployed web application firewall. It is an open source project that can be deployed either by embedding mod_security into the web server on which the web application runs or by deploying it on a network gateway machine as a reverse proxy to the server where the web application is running. Like most web application firewalls, mod_security can also serve as an intrusion detection system. Due to the complexity of web input, it is often useful to test a firewall rule by logging traffic that matches it first before enabling blocking, which can stop legitimate traffic if the rule matches input too broadly.

Even with proper configuration, application firewalls cannot defend against every possible attack. To completely protect a web application using an application firewall, the administrator would have to not only identify every entry point to the application but also fully understand what types of data should be allowed to enter that entry point. The number of entry points is typically far larger than the number of pages, since it includes not only every form input, but also URLs, cookies, and HTTP headers.

Even once the entry points are enumerated, infinite ways to encode attacks like SQL injection exist, using different SQL metacharacters, SQL string building, URL encoding, and Unicode. Worse yet, encoding techniques can be applied multiple times or combined. There are also attacks specific to the web application, such as reducing the purchase price of an item, which the application firewall cannot detect since it does not understand the semantics of the input. Application firewalls are of little use in preventing attacks that target the application's authentication or access control functions.

While blocking attacks with a web application firewall is less secure than fixing the code of the application, web application firewalls can be deployed quickly and are useful in situations where fixing the code is not possible. A new rule that blocks attacks against the vulnerability can often be implemented and tested more quickly than new application code can be introduced, validated, and deployed. Web application firewalls can protect vulnerable applications for which the administrator does not have the source code. They are also useful for their detection capabilities, since it is useful to identify attackers even if their attacks have not yet been successful.

4.2 Labs

The first two labs are focused on understanding specific vulnerabilities, including cross-site scripting, path traversal, and SQL injection. The difference between the two labs is that the second lab teaches students how to use a web testing proxy to identify the vulnerabilities, whereas the first lab only uses manual testing techniques. We use WebGoat[7], a teaching application from OWASP that consists of a series of lessons on common vulnerabilities, for these labs. WebGoat lessons are short and focused on a particular aspect of a single vulnerability. For example, there are several different labs covering SQL injection, including variants such as numeric, string, and blind injection.

Lessons provide multiple levels of hints, with additional levels beyond the first providing more and more help for the student. Some of the lessons contain final hints that are complete solutions, but other lessons are sufficiently difficult that questions are frequently posted about them on the WebGoat mailing list. As the main purpose of these labs is to introduce students to web application vulnerabilities, the disadvantage of the hints is minimal. The lessons provide an awareness of web vulnerabilities for classes that offer

minimal coverage of web security, while students in classes with a more in-depth coverage will study the vulnerabilities again in more realistic contexts.

Lab 3 and lab 4 focus on assessing the security of a web application. While the WebGoat labs teach students about vulnerabilities in isolation using a single web page, these labs teach students how to find vulnerabilities in a complete application. Students use a vulnerability scanner in lab 3 to assess a web application, then repeat the assessment in lab 4 using a testing proxy. Using both assessment techniques helps students understand the limitations of vulnerability scanners compared to manual assessment, as well as understanding that manual testing is time-consuming and plagued by human errors. In the software security course, students modify the source code of the application to fix the vulnerabilities that they discovered.

We wrote BlogEngine, a simple blogging application with a large number of inserted vulnerabilities, for these two labs. We also used web application training software that was designed for this purpose, such as Bad Store[9] and Hacme Bank[6]. These applications are freely available for a number of web platforms, but it may be possible for students to find solutions online that describe how to exploit the vulnerabilities. This problem matters less in a web programming class where students also perform security assessments of their own code or of the code of their classmates.

In lab 5, students deploy the mod_security web application firewall and evaluate how effectively it protects an application with known vulnerabilities. The firewall is deployed in embedded mode instead of being used as a reverse proxy, which would require configuration of an additional web server. Students test mod_security in logging mode to verify that attacks can be detected before configuring it to block traffic. They use their own rules to block specific types of attacks described in the lab in addition to using the core rule set.

Lab 6 should only be used in web programming and software security courses. In this lab, students use a static analysis tool to detect security vulnerabilities in the source code of a web application. Detected vulnerabilities are fixed, then the application is examined again with the tool to validate the fixes. Modern static analysis tools can find most common web vulnerabilities, though like all vulnerability detection tools, they will sometimes incorrectly report vulnerabilities that do not exist in the application or fail to report some vulnerabilities that are present. The source code analyzed is either code written by the students or the code of one of the teaching applications described above.

As open source static analysis tools produce too many false positives[19] or do not support the programming languages or web programming frameworks that we use, we use Fortify Software's Source Code Analyzer (SCA) for this lab. This tool is available via an academic license. A demonstration version is also included with the book *Software Security: Building Security In*[8]. SCA contains several tutorials, which teach how to use the tool, including one tutorial in which the student analyzes a web application written in Java. These tutorials can serve as another lab in themselves, which would be given before lab 6 to ensure that students are ready to use SCA on a nontrivial project.

Students are offered extra credit to participate in the international Capture the Flag (iCTF) competition. The iCTF competition is a distributed information warfare exercise that lasts for nine hours and involves dozens of universities from around the world. Each team is given a virtual network on which a number of insecure web applications are running. The applications are written in a variety of languages and use several different web development platforms. Source code is not provided for all of the applications. Teams score points by exploiting vulnerabilities of other teams and by defending their own web applications from exploitation.

5. CHALLENGES AND FEEDBACK

While web application security is an essential topic to cover, we experienced a number of challenges introducing it into the curriculum. It is difficult to find the time to add additional topics into the curriculum, and there are a variety of courses where web application security topics need to be covered. Both information security and web programming courses need to cover web security, though the topics covered differ due to the administrative perspective of information security and the focus on software development in web programming courses.

Since the classes in which web security topics need to be introduced often do not form a prerequisite chain, some topics must be covered multiple times despite the difficulty of finding space in the curriculum. Information security is not a prerequisite for web programming in our curriculum, so we had to cover certain topics, such as common web vulnerabilities, in both classes. We also had to introduce certain topics, such as the various types of web input, into the information security course, since web programming is an elective class.

We discovered that lectures are insufficient by themselves to improve the security of student code. Due to a lack of time to introduce additional labs, we introduced security into our web programming class through lectures. While the percentage of student final assignments containing SQL injection vulnerabilities improved from 100% the semester before the security unit was added to 94% afterwards, the improvement is not significant. However, none of the final projects in our software security class had such obvious vulnerabilities. That class included a series of web application security labs. Students also noted the importance of labs in course evaluations, though they also wrote that the labs were the most challenging aspect of the course and wanted to spend more time on them.

Students found the international Capture the Flag exercise to be exciting and extremely challenging. Keeping a set of vulnerable new web applications operating securely while under attack from dozens of other sites is a challenging task for security professionals too. One student remarked that he liked "the intensity and sense of urgency that went along with it, the fast paced learning." While the competition was fun, students might find a smaller in-class exercise more educational.

6. FUTURE DIRECTIONS

We would like to introduce a course focused on web security into the curriculum to cover all of the topics discussed in this paper. Having a dedicated course would allow web security to be studied in greater depth and for additional topics to be introduced. A web-focused Capture the Flag exercise could serve as a capstone for such a class. However, having such a course does not remove the need for the basics of web application security to be included in information security

and web programming courses. Wherever possible, students should be taught the secure technique for performing a task, such as issuing a SQL query.

The set of topics needs to be expanded to include AJAX (Asynchronous JavaScript And XML) security, as this technology is being used by a rapidly increasing number of web applications. AJAX applications do more work on the client side and use a server API. Unfortunately, both of these changes cause problems. The client can always be subverted or replaced by the user who controls the web browser, while the server API exposes much more of the server-side code's functionality than is accessible in traditional web applications. Developers may believe that the calls to the server API are invisible to an attacker since they are not visible through the web browser. However, these calls can be observed using a web proxy or network sniffer and are no more confidential than hidden form fields.

The server API establishes a large attack surface for AJAX applications, as it is accessible not just by the application's client-side code, but by any code written by attackers as well. For example, if an application exposes separate functions to deliver an item to the customer and to charge the customer's credit card, an attacker can call the function to obtain an item without calling the function to pay for it. The asynchronous nature of the client-server interaction also opens up AJAX applications to race condition vulnerabilities. Race conditions are always possible when the developer relies on the client to submit operations in the correct order, as the attacker can write their own client.

Since web programming courses are offered before software engineering in our curriculum, we cannot introduce a complete secure development life cycle[4] in these classes. However, there are some aspects of the life cycle that can be introduced earlier as part of every assignment, such as code reviews and testing. Multiple experiences with these techniques should prove more beneficial to students than introducing these activities as isolated assignments.

7. CONCLUSIONS

While attackers are rapidly moving malicious activities to the web, university curricula are not adapting as quickly. Web application security topics need to be integrated into the IT curriculum, particularly in courses on information security and web programming. This paper discussed the need for introducing web security, the content that needs to be addressed, and a set of lab assignments. The discussion of lab assignments included techniques for implementation, the web security tools used, and recommendations for where to introduce them in the curriculum. Future directions for this work include the creation of a dedicated class in web application security and the addition of topics such as AJAX security.

8. REFERENCES

[1] SIGITE Curriculum Committee (2005). Computing curriculum 2005, IT volume. http://sigite.acm.org/activities/curriculum/, 2005.

[2] CERT. Vulnerability remediation statistics. http://www.cert.org/stats/vulnerability_remediation.html, 2007.

[3] M. P. Dafydd Stuttard. *The Web Application Hacker's Handbook*. Wiley, 2007.

[4] J. Gregoire, K. Buyens, B. D. Win, R. Scandariato, and W. Joosen. On the secure software development process: CLASP and SDL compared. In *SESS '07: Proceedings of the Third International Workshop on Software Engineering for Secure Systems*, page 1, Washington, DC, USA, 2007. IEEE Computer Society.

[5] A. Judson. Tamper data. https://addons.mozilla.org/en-US/firefox/addon/966, 2008.

[6] R. A. Mark Curphey. Hacme bank. http://www.foundstone.com/us/resources/proddesc/hacmebank.htm, 2006.

[7] B. Mayhew. Webgoat. http://www.owasp.org/index.php/Category:OWASP_WebGoat_Project, 2008.

[8] G. McGraw. *Software Security: Building Security In*. Addison-Wesley, 2006.

[9] NetContinuum. Bad store. http://www.badstore.net/, 2008.

[10] OWASP. OWASP top 10. http://www.owasp.org/index.php/Top_10_2007, 2007.

[11] OWASP. Web scarab. http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project, 2008.

[12] Paros. Paros proxy. http://www.parosproxy.org/, 2008.

[13] PortSwigger. Burp suite. http://portswigger.net/suite/, 2008.

[14] I. Ristic. mod_security. http://www.modsecurity.org/, 2008.

[15] R. S.Christey. Vulnerability type distributions in CVE. http://cve.mitre.org/docs/vuln-trends/index.html, 2007.

[16] N. Surribas. Wapiti. http://wapiti.sourceforge.net/, 2008.

[17] A. van der Stock (ed). OWASP guide to building secure web applications. http://www.owasp.org/index.php/OWASP_Guide_Project, 2008.

[18] Websense. Research highlights q3-q4: 2007. http://www.websense.com/securitylabs/docs/SecurityLabsReport_Q4_011808.pdf, 2007.

[19] M. Zitser, R. Lippmann, and T. Leek. Testing static analysis tools using exploitable buffer overflows from open source code. 29(6):97–106, 2004.