Webseclab Security Education Workbench

Elie Bursztein, Baptiste Gourdin, Celine Fabry, Jason Bau, Gustav Rydstedt, Hristo Bojinov, Dan Boneh, John C. Mitchell
Stanford University
elie,bgourdin,jbau,rydstedt,hristo,dabo,mitchell@cs.stanford.edu

Abstract

We have developed and tested a virtual-machine-based web-application security student laboratory, *Webseclab,* comprising a LAMP (Linux, Apache, MySQL, PHP) stack, a variety of development tools, and the three most popular browsers for the Linux platform. This environment, tested in weekly participatory labs and weekly homework, hosts a teaching framework, exercise sets and labs, and a sandboxed student development environment. Eighty incremental exercises based on recent security research, and challenge projects, including one based on real open-source applications, teach the major web application vulnerabilities and defenses, in an encapsulated environment that allows students to experiment freely without interfering with each other or with public networks. In contrast to problems experienced with hands-on projects used in previous years, student response to this platform and its contained exercises has been remarkably positive.

1 Introduction

While traditional computer security threats remain an ongoing concern, web application security vulnerabilities and attacks rose sharply over 2000-2005, eclipsed traditional system security vulnerabilities in 2006, and have persisted as close to half of all vulnerabilities since (See Figure 1). One reason that securing web applications is complex is that the web application delivery platform is complex: web applications are written in JavaScript, PHP, and other languages, delivered using a stateless protocol over an insecure Internet, and rendered on varying browsers over varying operating systems and an increasing complex array of content-specific plug-ins.

Teaching web application security is challenging because of the inherent complexities of the web platform and the rapidly changing nature of the field, with at least 65 new types of attacks every year since 2006 [5]. In addition, hands-on programming projects that require students to find vulnerabilities and develop and test defenses have proven difficult to deploy. While students can often experiment with some forms of client-side functionality, they have been previously limited to software that is readily available for the specific computers that each of them own. In addition, network-based experiments with servers may cause one student's activities to interfere with others and university network administrators have been uncomfortable with some of the network activity generated by hands-on security projects. We therefore developed a uniform, self-contained virtual-machine-based environment that allows each student or student project team to have their own isolated combination of browsers, server, database, and network. Within this environment, we have developed a series of project exercises, equal in heft to textbook exercises for a *semester*-long university-level course, that expose students to a full range of web application security problems and solutions. While we describe many exercises in this paper, we continue to develop additional projects and anticipate that further distribution of our platform will allow others to contribute demonstrations of other important concepts.

Webseclab is the product of more than 750 person-hours of work. It is comprised of a LAMP (Linux, Apache, MySQL, PHP) stack, a variety of developer tools, and the three most popular browsers for the Linux platform. Webseclab hosts, at localhost URLs, both a custom application utilizing focused exercises to teach topics important to secure web programming, and a second student development sandbox isolated from the exercise

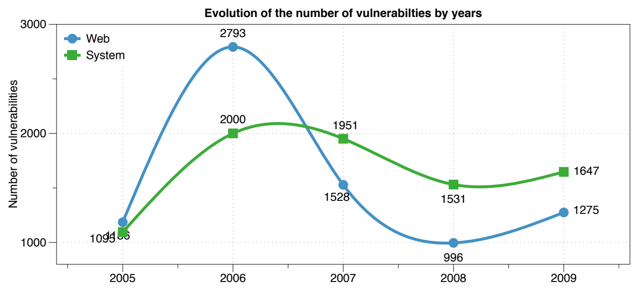


Figure 1: Web application vulnerabilities versus system vulnerabilities in VUPEN database

databases. The teaching system, with 10,000 custom lines of PHP, is designed for student accessibility and interactivity, featuring convenient access to reference resources and instant feedback for exercise answers. Further, webseclab is up-to-date with current web application security research, comprehensive of the major web vulnerabilities and defenses, and encapsulated to allow student to experiment freely without interfering with each other or any resources external to the VM. Webseclab has received positive student response regarding its incremental exercises and capstone challenges based on real open-source applications, suggesting the success of the approach. Anyone currently interested in evaluating Webseclab may access it at `http://md5.stanford.edu/vm/`.

The remainder of the paper is organized as follows: In section 2, we introduce the Webseclab architecture and discuss the rationale behind our choices. In section 3, we discuss how we designed the exercises and how users interact with them. Section 4 discusses a sample capstone project based on an open-source web application, and section 5 presents additional related work. Finally, in section 6 we reflect on lessons learned and discuss future work based on the feedback of our students.

2 Architecture

2.1 VM and Application Hosting Stack

Webseclab is distributed as an entire LAMP (Linux, Apache, MySQL, PHP) stack contained in a virtual machine (VM). The Webseclab software stack is illustrated in Figure 2. The student-facing teaching framework is

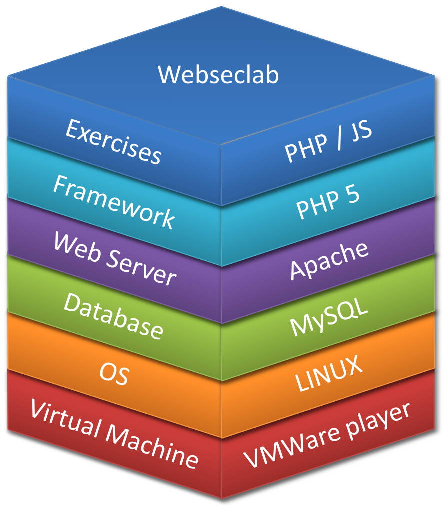


Figure 2: Virtual machine stack

itself implemented as a MySQL+PHP web application. While we could have chosen to host the Webseclab application for students to access remotely, as in the Google Jarlsberg project [9], we made the conscious decision to distribute the lower layer hosting environment and an entire VM along with the teaching application for several reasons:

- **Completeness.** Web vulnerabilities extend beyond the application layer to the lower layer of the stack. For example, four of the Open Web Application Security Project (OWASP) Top Ten [11] web application vulnerabilities originate in either server or cryptographic layer errors. Injection vulnerabilities also involve back-end server software beyond the the web application language. Having an entire stack allows us to include material for the security of these layers, including SQL Injection and SSL configuration.

- **Sandboxing.** Hosting the security exercises, especially ones inviting students to attack vulnerable code, on their individual VMs quarantines these attacks from the public Internet. It also prevents individual students from bringing down resources shared

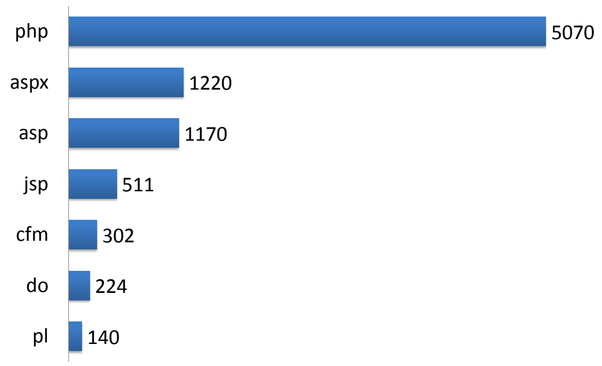


Figure 3: Google URL count for well known language extensions (in millions)

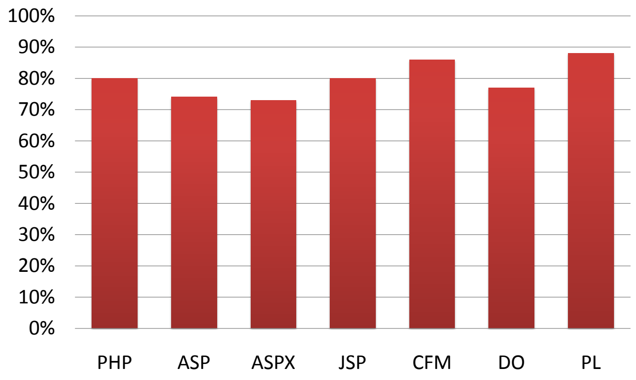


Figure 4: Percent of websites with at least one serious vulnerability, broken-out by language (Whitehat survey)

by the entire class, allowing for a smoother course experience.

- **Uniformity.** Packaging an identical software stack (both server- and client-side) into the VM reduces student concerns over compatibility between the environment in which they solve exercises and the environment used to grade them. In addition, preloading a host of utility software such as IDEs, browser-extensions, etc, simplifies student technical support and allows detailed, software specific instructions to be included in student tutorials.

We chose LAMP as the application stack for Webseclab mainly because it is a popular way to support PHP, the predominant server-side web application language, as illustrated in Figure 3. While some may regard PHP as a contributory factor for certain vulnerabilities (e.g., SQL injection or poor session management), PHP is correspondingly effective for demonstrating a wide range of vulnerabilities, and as Figure 4 shows, there is little evidence that alternative languages are more inherently more secure.

2.2 Dual Hosting Environments

The LAMP stack of Webseclab contains two different web application environments. The first environment hosts the teaching framework and its exercise sets at a localhost URL (Figure 7). Student interaction with the teaching framework is limited to accessing exercises using a browser interface. Any code that students write in completing exercises must be entered via the browser (details in Section 3.2), and students are prevented from accessing the PHP source code of the exercises in the file system via access control.

To support student code development (such as programming projects) of a larger scale than exercises, Webseclab offers a second hosting environment accessible via a different localhost url than the exercises. This environment hosts a full development sandbox including a directory where the student can write her own PHP files and a MySQL database granting the student user full access. The development sandbox database is also isolated from the exercise databases using access control privileges, to prevent student tampering with the exercise system.

2.3 Client-Side and Utility Software

In order to accommodate varying student preferences for development environments, the VM includes a variety of text editors, including gedit and emacs, as well as a full-blown IDE (Netbeans). We also include the most popular browsers for the Linux platform (Firefox, Chrome and Opera) to expose students to the challenging heterogeneity of web development. A range of browsers is useful, for example, for illustrating the Same-Origin Policy, because Chrome and Firefox support different policies than Opera

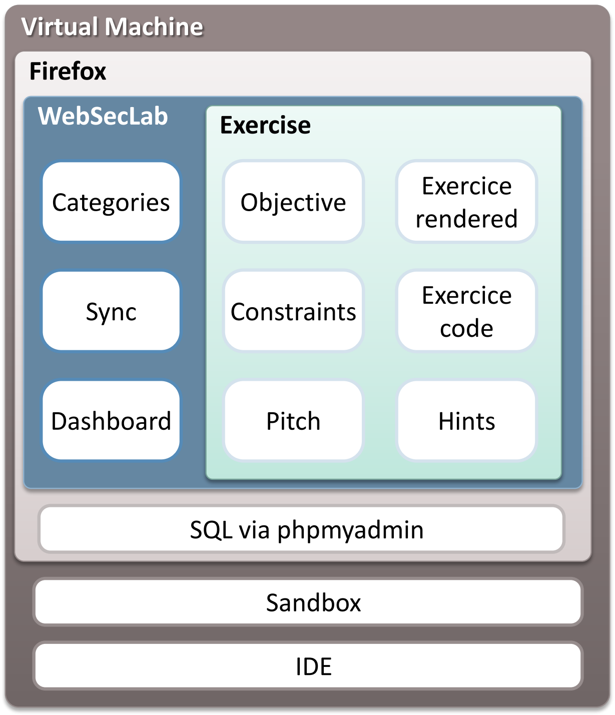


Figure 5: Security lab functional representation

for cross-origin AJAX requests. Finally, the teaching framework supplies a variety of Firefox extensions that allow students to observe web site behavior, including the Firebug, Firecookie, TamperData and UserAgentSwitcher extensions.

2.4 Webseclab.com

For student data migration and backup from the VM, Webseclab supports import and export of student data from the exercise framework. This exports the exercise progress of the student into a portable file that can be loaded into another VM instance of Webseclab (for grading, for example). At Stanford, we have also set up webseclab.com as a central repository for student data. Each individual student establishes an account and periodically uploads their progress. Webseclab.com also contains functionality for the teaching staff to assign programming projects, whose specifications will automatically be downloaded to student VMs. Teaching staff may also use webseclab.com to monitor student progress (see Figure 6) and each student may view their progress as seen by course staff.

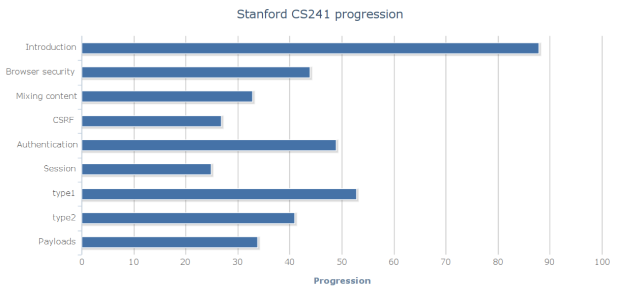


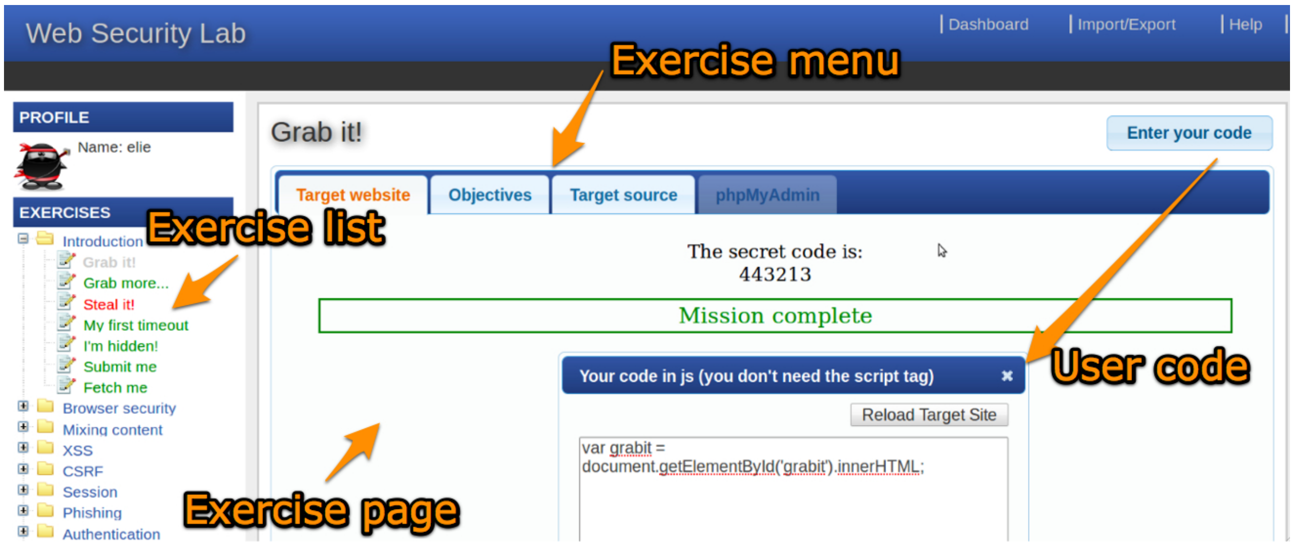
Figure 6: Student exercise progress via webseclab.com

3 Exercise Design

In this section, we first describe our experience using Webseclab to teach Secure Web Programming (CS241) at Stanford [6]. Then, we describe the user-interface of the exercise system and how we designed exercises to meet our goals of comprehensiveness, timeliness, and student interest.

3.1 Teaching Experience

Based on our experience in previous courses, we believe that hands-on experience is the optimal way to teach the concepts of web application security to students. There is no better way to provide concrete understanding of the challenge of secure web programming. In lieu of textbook or homework, we designed a series of short, topical programming exercises for the students to complete as a supplement to the lectures, the majority of which invite students to craft attacks against intentionally vulnerable code in pages supplied by our teaching framework. When the attack is successful, the teaching framework will automatically provide positive feedback to the student, as shown in Figure 7. By pairing this positive feedback with intentionally focused exercises with short time-to-completion, we provide the student a steady stream of encouragement



The secret code is:
443213

Mission complete

```
var grabit =
document.getElementById('grabit').innerHTML;
```

Figure 7: Overall page screenshot

that hopefully increases learning interest and accomplishment.

All of our current exercises may be completed by the students on their own time, under the guidance of automatic feedback from the system. In addition, the exercises also adapt well to a weekly "lab" setting, where one or more instructors meet with a group of students for one-to-two hours to complete the exercises. During the lab sessions, we typically have very active student interaction, with students often staying beyond the scheduled time to engage the course staff. The session covering Cross-Site Scripting, for example, is illustrative. In this session, we ran a student contest using a set of XSS exercises covering increasingly difficult filter-evasion techniques. The first student to solve each level and demonstrate the solution to the rest of the students was awarded points, with the overall winner receiving a small material prize. This contest enjoyed the highest attendance of the term, ran 30 minutes past the scheduled one-hour deadline, and caused students to stay even after the prize had already been awarded to compare solutions and complete the exercises they had failed during the contest. The exercises are tied with the lectures as the highest-rated part of the course, according to mid-term student evaluations.

3.2 User Interface

The user interface of the exercise system, illustrated in the Figure 7, organizes all relevant information for ease of student access. The list of exercises appears as a menu in the left pane of the teaching system, with some exercises only becoming available after the completion of prerequisites. Most exercises ask the student to attack vulnerable code in pages supplied by the teaching framework, while some exercises ask the student to insert defensive code. The horizontal tabs provide views of the target site, as rendered by the browser in the "Target website" pane and in source-form in the "Target source" pane. The "Objectives" pane provides exercise objectives, such as using SQL injection to bypass authentication checks, any constraints imposed by the teaching staff, such as disallowed attack techniques, and references and hints on how to solve the exercise.

Some exercises, such as the SQLI authentication bypass, may be completed simply by interacting with forms on the rendered target page. If the interaction satisfies the exercise-completion check (run in either Javascript or PHP), the teaching framework automatically provides a success message to the student. Other exercises, such as creating Javascript timing attacks, validating cookie data in server-side code, and crafting click-jacking pages, require the student to write code (in Javascript, PHP, or HTML). Students enter code using a pop-up interface generated by the "Enter Your Code" button, as shown in Figure 7. When complete, the student code is inserted into the existing code of the target site at a location selected by the teaching staff. The modified target site is then re-rendered by the browser, and exercise-completion checks are run again for instant student feedback. Whether code entry is required or not, exercises may be completed and relevant reference information obtained on a single browser tab, for student convenience.

3.3 Exercise Goals

Student Interest. Aiming to maximize student interest through careful "packaging", we designed the exercises to contain a small element of entertainment by conforming each to a cohesive storyline of a fictitious Ninja clan. The objectives of each exercise advances the interests of this clan in some manner, such as gaining access to a rival clan's member website or executing a mission for a client.

Comprehensiveness. We aim to give students thorough training in the most important web application security topics by providing the most comprehensive set of exercises of which we are aware. Table 1 presents the categories covered by our exercise set and the number of exercises in each category. The exercises begin by familiarizing the student with the teaching framework, Javascript, and the Document Object Model (DOM), using the "Introduction" and "Browser Security" sets; they then illustrate the operations and limitations of the DOM Same Origin Policy (SOP) with "Mixed Content" exercises.

After the introductory sets, the teaching framework offers training for high-incidence categories of web-application vulnerabilities: *Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), SQL Injection (SQLI), Incorrect User Authentication, Incorrect Session Management, Phishing, Incorrect SSL Usage, Breaking Javascript Isolation, and Cross Channel Scripting*. These exercise categories cover most topics considered important by the security community: Table 1 shows that our current set of *80* exercises cover *9* of the topics in the OWASP Top 10. We plan to add exercises for Insecure

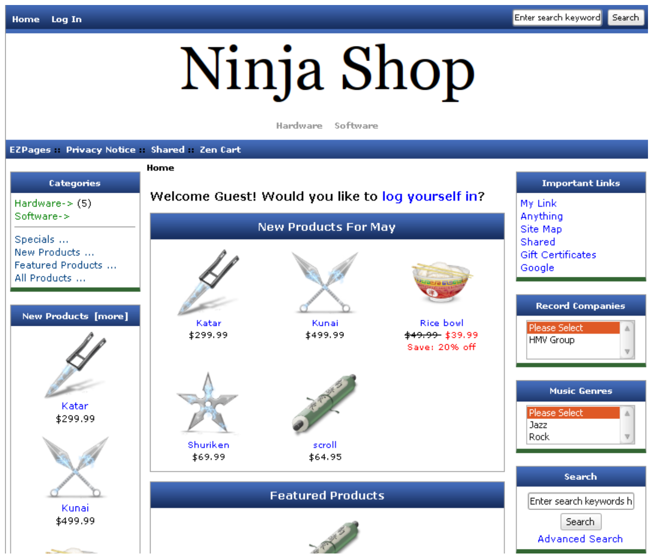


Figure 8: Capstone project

Direct Object Reference shortly as we build up the mock database used in the teaching framework.

Timeliness. In addition to comprehensively covering topics deemed important by the web security community, we also want our exercises to reflect current research in web application vulnerabilities and attacks. To this end, we included exercises covering topics in cross-site request forgery [1], phishing and click-jacking protection [12], Javascript mashup isolation [10], and cross-channel scripting attacks against the web interfaces of consumer devices [4], all of which are derived from recent academic research published by the co-authors of this paper.

Webseclab also covers recent topics in browser security through exercises about browser denial-of-service, the SOP and also extension security. We feel it is important that students thoroughly understand the client-side model of web security, and we plan to add more advanced browser security exercises that teach students how browser defenses can mitigate server side vulnerabilities such as click-jacking and XSS [2].

Table 1: Webseclab Exercises Covering OWASP Top 10

OWASP Top 10 Category Exercises	Introduction	Browser security	Mixed Content	XSS	CSRF	Authentication	Session	Phishing	Javascript Isolation	SQL Injection	SSL	Cross Channel
Injection										✓		
Cross-Site Scripting				✓								
Broken Authentication and Session Management						✓	✓					
Insecure Direct Object References												
Cross-Site Request Forgery					✓							
Security Misconfiguration				✓	✓		✓					
Insecure Cryptographic Storage										✓		
Failure to restrict URL Access					✓		✓					
Insufficient Transport Layer Protection							✓					
Unvalidated Redirects And Forwards				✓	✓							
Total Exercise Count	7	8	7	17	6	5	12	7	7	6	4	5

4 Projects in Webseclab

While most Webseclab exercises are narrowly focused on a specific vulnerability stated in the exercise objective, we also added longer programming projects and a capstone vulnerability-audit project.

For the most-recent term, we developed three longer-term programming projects to be completed by students using the Webseclab sandboxed student-development environment [6]. These projects, to be completed over 2-3 weeks, required students to build server-side code to fingerprint each visitor, client- and server-side code in a framework for exploiting XSS, and a browser extension for detecting insecure web page configurations such as passwords sent in the clear. We also graded these projects in the Webseclab environment, which eliminated all cases of incompatibility between development and grading environments, a problem which has occurred a few times per term in the past. We feel that Webseclab can easily support other web-security related programming projects of similar heft.

Webseclab's capstone vulnerability audit project was created by injecting vulnerabilities into a well-known open source application. As shown in figure 8, we developed a vulnerable version of the popular e-commerce application: Zen-cart [14]. We used Zen-cart to create a fake "ninja shop" (See Fig. 8) that contains vulnerabilities from all categories available in the VM, including XSS, CSRF and SQL injection. It also has insecure cookies, insecure authentication and predictable session tokens. We used this real application as a final take-home exam for our class this year and plan to roll out more capstone exercises based on popular open source application such as Mediawiki, Wordpress and phpBB, with various degree of difficulty. We may also leverage our study of vulnerabilities in past versions of these sites, and the methods needed to detect them [3].

5 Related work

Related work on web security education has followed one of two different models: VM environments executed on the student's hardware, or hosted web sites with embedded vulnerabilities—open to experimentation by students.

While VMs have been used for several years for student education, including security [8], web security content has only now started to appear, as far as we are aware.

Pace University's NSF SWEET project [13] has goals that are conceptually similar to ours. However, the SWEET VMs available for both Linux and Windows are not focused on web security. Instead, they offer a variety of modules, one of which is on web security. In contrast, the WebSecLab VM is dedicated to web application security, offers an integrated and focused environment, and leverages recent web security research. In part, this is made possible by the Webseclab VM, which also serves as the development environment for the staff, allowing for straight-forward content development, testing, and addition.

Hosted web services are another natural way to offer web security education. In fact, remotely connecting to web services might be a more "natural" way to learn about security on the open Internet. The Jarlsberg Web Application Exploits and Defenses codelab [9] offers a wide range of exercises in a web format, accessed from the convenience of the user's browser. When it comes to offering the best possible coverage of security topics, we have found several difficulties with the hosted approach. First, not all of the current web security topics can be covered without significantly impacting the user's OS: in some of the exploits the browser can become effectively unusable, requiring a restart; in others, a specific browser or extension must be used, which would require the user to install new, possibly vulnerable software on their primary machine. Second, some areas of web security are simply not possible to cover using only a web interface. One such area is cross-channel scripting, where access to a diverse set of protocols (outside HTTP) is required. Third, from a student workflow perspective, having a VM allows learning without requiring non-stop access to the Internet.

The report in [7] outlines the strengths and weaknesses of a VM-based environment, specifically in teaching computer security. We believe that for web application security, a VM is particularly attractive: web vulnerabilities are inherently client-server, thus easy to abstract into a VM. At the same time a VM reduces the impact on the user's primary OS, thus a core strength described in [7] is preserved.

6 Lessons Learned

In this paper, we have presented **Webseclab**, a virtual-machine-based web application security student laboratory based on open source technologies, which we believe is a comprehensive, up-to-date, and compelling teaching platform. We are planning to incorporate feedback from our spring-quarter course into Webseclab and will release an improved version for free to the public this summer.

The experience of running similar security courses in previous terms taught us the importance of *encapsulation* in hands-on projects. By using the VM to isolate student users from one another and to prevent their attacker-like traffic from reaching public networks, Webseclab made this course a much smoother experience than previous editions for students, course staff and also Stanford's network administrators. In addition, the hosting stack provided by Webseclab allowed us the ability to exercise lower-layer vulnerabilities, to conveniently show students exercise source code when helpful, and to ask students to write defense code as exercise solutions. We thus feel the inclusion of the *entire host stack* was a valuable feature.

We also learned from midterm student evaluations that they enjoyed the hands-on experience of the exercises and also liked, in principle, the convenience provided by the VM and the user-interface of the exercise system. We found achieving user convenience in both the UI and the VM to be a non-trivial exercise, as we devoted thousands of lines code to smoothing out the teaching application and experimented with nearly 20 Guest-OS and VM-player combinations to optimize factors such as stability, distribution-size, length of support-term, and performance. Furthermore, we are continuing to improve the user-interface for greater intuitiveness in response to student feedback.

We feel that, even given the present content of Webseclab, the continuing process of keeping the exercise topics up-to-date with the latest vulnerabilities and defenses will be a challenge. To this end, we have designed an administrator interface to the Webseclab application that will hopefully ease the process of adding newly relevant exercises.

Finally, given the way our students learn to adopt the perspective of an attacker, we find it difficult to completely secure our auto-grading system against forgery, since student code often has name-space access in PHP or Javascript to functions that signal exercise completion. While we have not encountered any actual instances of forgery, our solution is two-pronged: Webseclab records all submitted student answers, allowing for later evaluation on an instructor-controlled VM, and in conjunction with discussion of ethical use of security knowledge, we emphasize an honor code that stresses ethical use of the teaching environment and its software functions.

References

[1] A. Barth, C. Jackson, and J. C. Mitchell. Robust defenses for cross-site request forgery. In *15th ACM Conference on Computer and Communications Security*, 2008. 6

[2] D. Bates, A. Barth, and C. Jackson. Regular expressions considered harmful in client-side xss filters. In *19th International World Wide Web Conference*, 2010. 6

[3] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell. State of the art: Automated black-box web application vulnerability testing. In *IEEE Symp. on Security and Privacy*, 2010. 7

[4] H. Bojinov, E. Bursztein, and D. Boneh. Xcs: Cross channel scripting and its impact on web applications. In *16th ACM Conf. on Computer and Communications Security*, 2009. 6

[5] E. Bursztein and B. Gourdin. Web security trends 2010. `http://www.inftoint.com/security/web-security-trends-2010/`, May 2010. 1

[6] Cs241: Secure web programming. `https://courseware.stanford.edu/pg/courses/81544`. 4, 7

[7] A. Davidson, J. de La Puente Martinez, and M. Huber. A swot analysis of virtual laboratories for security education. *9th IFIP World Conference on Computers in Education*, 2009. 8

[8] J. Hu, D. Cordel, and C. Meinel. A virtual laboratory for it security education. *EMISA 2004 Informationssysteme im E-Business und E-Government*, 2004. 7

[9] Web application exploits and defenses. `http://jarlsberg.appspot.com/`. 2, 8

[10] S. Maffeis, J. C. Mitchell, and A. Taly. Object capabilities and isolation of untrusted web applications. In *IEEE Symposium on Security and Privacy*, 2010. 6

[11] Owasp top ten. `http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project`. 2

[12] G. Rydstedt, E. Bursztein, D. Boneh, and C. Jackson. Busting frame busting: a study of clickjacking vulnerabilities at popular sites. In *IEEE Oakland Web 2.0 Security and Privacy Workshop*, 2010. 6

[13] L. Tao and L.-C. Chen. Improving web security education with virtual labs and shared course modules. *7th Annual Research Day, Seidenberg School of Computer Science and Information Systems, Pace University*, 2010. 8

[14] Z. Ventures. zen cart the art of e-commerce. `http://www.zen-cart.com/`. 7