



Capstone Courseware, LLC

33 Boylston Street
Jamaica Plain, MA 02130

877-227-2477
capstonecourseware.com

Securing Java Web Applications

Version 5.0

Instructor's Guide



Overview

Software security is a sprawling topic, and probably the biggest challenge in teaching Java web-application security – as it was in writing about it – is going to be pulling it together into a coherent “story.” The basic goal in anyone’s mind is simple enough: don’t get hacked. But the means of attack are myriad, and the countermeasures that we take in developing and deploying web applications must be likewise diverse.

I’ve tried as much as possible to categorize security threats and techniques into well-contained chapters, and to establish some clear front-to-back flow. But still the course is a mixed bag in many ways. Some of the techniques are declarative, some programmatic (more or less traditional API training really), and some are more design-level, or implementation best practices. Some tasks covered here are naturally for web developers, and that is the primary intended audience; but there are administrative chores as well.

On the flip side, probably the greatest asset in teaching the class is that, well, hacking is still kinda cool, if you look at all the creativity and cleverness that goes into some of these attacks. As much as possible I’ve organized the course to present possible attack vectors and then show how to close them off. Students usually find at least some sections of the course to be real eye-openers. There’s always one more trick to learn in this area. And the demos and labs do develop a nice spy-vs-spy sensibility that helps keep everyone engaged.

Hopefully students will leave the course well advised about the basic practices such as **web.xml** security constraints, using HTTPS, and steering clear of XSS and injection attacks; but also they’ll have developed an appreciation for the extent to which really secure development still involves application-level coding. The servlet filter has its reputation enhanced quite a bit by the time we’re done, and some basic JCA/JCE exposure should go a long way for students as well.





Timeline

The following breakdowns are approximate, and every class will vary.

Day 1

2½ hours Chapter 1

5½ hours (spans days) Chapter 2

Day 2

4½ hours Chapter 3

Day 3

2 hours Chapter 4

3 hours Chapter 5

1 hour Chapter 6

These chapters are not evenly weighted, as a quick look at the coursebook will confirm. Don't worry that you're only "halfway" through the six chapters at the end of the second day. Chapter 4 has some good meat to it but the lab is a prime candidate for skipping if you're running behind; and Chapter 6 really is quick, even if you do the lab.





Tools Deployed with the Lab Software

This course's software requires separate setups of the 6.0 JDK, the Crimson text editor, and/or Eclipse WTP; otherwise it is self-contained, as the lab installer sets up not only the lab software but necessary tools – these will all be found in directories under **c:/Capstone/Tools**:

- **JPA 1.0**, in **JPA1.0**. The **lib** directory provides the **persistence.jar** (API) and **toplink-essentials.jar** (provider/implementation) files to support some of the course exercises in their use of JPA entities.
- **JSTL 1.2**, in **JSTL1.2**. The JARs and TLDs for JSTL tags used in many of the course-exercise JSPs are found here.
- **MySQL 5.0**, in **MySQL5.0**. Our application databases are hosted by MySQL, and the **lib** directory also holds the necessary JDBC JAR file (which is promptly copied over to the **lib** directory under Tomcat).
- **Tomcat 6.0**, in **Tomcat6.0**. This is our target web server for deployment and testing.





Ant Build Process

We use **ant** for our web-application builds. (Standalone-application projects use simple **build** and **run** batch files.) Though most students will be happy to let the **ant** command take care of things, some students (and most instructors) will want to understand the inner workings here a little better. Each project in the course has its own **build.xml** and **build.properties** files; these rely on the master environment variable **CC_MODULE** to import targets and properties in a central directory, **%CC_MODULE%\Ant** which is typically **c:\Capstone\JavaWebSecurity\Ant**. Information here defines a routine for building a web application.

The build will create a **build** subdirectory that is the root of a Java web application, and compile Java classes from **src** into **build/WEB-INF/classes**. It will copy the full tree of files under **docroot** to **build** – these are our JSPs, CSS, supporting data and image files, and configuration files including **web.xml**. Then the build will draw in JPA and JSTL libraries and archives to the appropriate places in the web-app structure under the **build** directory.

Prior to building (and you could do the above tasks only, by typing **ant build**), the default Ant target (**all**) will undeploy any prior version of the application, and after the build it will (re-)deploy. It does this by generating a **<Context>** file into Tomcat's **conf/Catalina/localhost** tree, thus pointing Tomcat to the **build** directory as the root of the web application. For most applications this context includes declarations of a **JDBCRealm** and a **Resource**, which allow Tomcat and the application itself, respectively, access to the application database.

All this means that students can simply type **ant** from the command line set to any example step, demo or lab directory as the working directory – if there's a **build.xml** in the directory, you're good to go. Any change to Java, JSP, XML, etc., will be reflected in the re-deployed application – just give Tomcat it's 30 seconds to sweep for changes and re-install before testing from a browser.

There are also targets **create-DB** and **remove-DB** for all but a couple of the web-application projects. Each case study has its own database: Health, LandUse, Login, and Retail. Generally speaking, you need to **ant create-DB** once for each of these, before the first build and test of any steps in that case study. You don't need to rebuild the database each time you deploy or move to a new step. There are a few explicit instructions to rebuild, such as when changing over from plain to hashed passwords in the Health database, in Chapter 5.





Eclipse Overlays

Capstone provides an optional package of workspace and project files for Eclipse WTP 2.0 for this course. (See the course Setup Guide for download URLs.) Instructors, use this package on your own initiative and at your own risk. You should have experience yourself with Eclipse WTP before using the overlay package in the classroom, and preferably only students with Eclipse experience will use the overlay themselves. The workspace and projects have been tested lightly with the course but are not part of the standard product.

That said, this overlay should save a good deal of work for those who wish to use Eclipse as the primary code editor for the course. The sweet spot will be editing source code in Eclipse, but configuring and testing from DOS. See the file **ReadMe.html** (shown when the workspace is first opened) for general notes on how to use the Eclipse overlays for Capstone courses, and be sure to read the course-specific notes in **Docs\JavaWebSecurity Module Notes.html**. To summarize a couple key points from that document:

- You must start Eclipse from an environment with **CC_MODULE** correctly set. Various steps of various applications, though not all of them, will fail otherwise.
- You must build and populate databases from the command line, before building and testing corresponding applications from within Eclipse.

Again, Capstone Courseware can only offer complete technical support on the standard course, and while we hope this overlay is convenient, it is not as thoroughly tested as the core lab image at this time. If a given exercise is giving trouble, please be certain to build and run it from the command line, using the SDK tools as prescribed in the student guide, before contacting Capstone.

We always welcome feedback on our courseware, and we would appreciate whatever comments and criticism you might have. Eclipse, like all IDEs, tries to promote ease of use by providing many different ways of doing the same thing; this is convenient for users but does leave a lot of questions as to what's best practice. At this time we believe consensus is still forming around a number of basic practices, and we'd like to hear how you use Eclipse for training situations and how this overlay works out for you. Please contact Will Provost at provost@capstonecourseware.com.





Teaching Notes

Chapter 1

This overview chapter attempts to break down the complex problem of security for Java web applications, in a few dimensions. First we talk about major areas of vulnerability: server, network, and browser. While we're at it, we try to sift out the things we can control from the things we can't, so to speak: what sorts of threats can we manage or mitigate in our role as application developers? For example, URL history and cookie stores are browser vulnerabilities, but we can limit our exposure by decisions we make in the application design: avoid HTTP GET for form submissions, and narrow the use of a given cookie and possibly encrypt its value.

The one code example in this chapter is partly meant to set up a straw man that shows some really egregious web-security errors; but at least as importantly it provides an early opportunity to prove out the lab setup on everyone's machine. Environment variables, executable path, and availability of the necessary network ports, all will be proven by building and testing this simple example, and it will give students a good sense of how the rest of the course's labs are to be built and tested.

The rest of the chapter establishes some themes that will resonate through the rest of the course: defense in depth, "CIA" goals, and so on. There are also some stray introductory techniques that will get us going in a mode of thinking about secure development practice and carry nicely into the next few chapters.

The OWASP mention at the end of this chapter provides an important touchstone for the rest of the course. The OWASP Top Ten really helps to bind our chosen security topics together – helps tell that coherent story mentioned at the top of this document. By the way, instructors are strongly advised to spend a couple of hours browsing OWASP resources and getting familiar with them. It's a great way to go into class feeling well-versed and having a few extra stories to tell in class.





Chapter 2

This long chapter addresses both declarative and programmatic security techniques for servlet-based applications. It explains HTTP BASIC and DIGEST schemes, to a medium level of detail that is nevertheless more than that to which most students will have been exposed previously. The HTTPSneak application provides a fun way to investigate the inner workings of both schemes – and, later, to expose their weaknesses with regard to privacy of user credentials and replay attacks. (By looking at different possible DIGEST replays and attempting to use the wrong digest for new requests, you'll also be able to illustrate the overlap between HTTP schemes, which insist on re-sending credentials with each request, and how a Java container actually authenticates once and then trusts its own generated session key. Or you can prove this more simply by replaying a BASIC request with the authentication string deleted.) The first lab is somewhat labor-intensive but provides good practice in the essential skills of declarative security.

We then move on to FORM authentication and some related techniques such as custom error pages and logout functions. We also discuss briefly the option of taking over authentication logic entirely from the container – the “DIY” approach. There's a lot to consider in the choice of FORM or DIY strategies, in fact more than we can really discuss in the coursebook. But a branch of DIY implementations is included in the lab software, and will start to show up in later chapters. If students are interested you may want to dive into the basic logic of the **Authenticator** and **RoleAccess** classes; this is not covered explicitly anywhere else in the course.

Finally there's a section on programmatic authorization. Our exercises illustrate two common uses of the servlet security API: conditional production of UI and authorizing access based on user-to-resource matching. You might want to mention the `<security-role-ref>` element during this section. All our exercises take advantage of the default behavior of mapping role references to abstract role names – as many applications do – but it's good to know that there is one more level of indirection available to the developer.

Make sure when reviewing this final lab to ask students about the behavior of the **ShowHistory** servlet for users that are both nurse and patient, or both clerical staff and patient: (a) what should happen in this case, and (b) what do they think will happen? Give it a test with Edgar Frank or Larry Best and review the code again to see how the logic is arranged to get the right results.





Chapter 3

With this chapter we depart from Java EE specifics of XML models and API syntax, and think more generally about possible vulnerabilities in the application. This chapter threatens to break down into a hodgepodge of unrelated threats and appropriate countermeasures. But each of these styles of attack is pretty interesting in its own right, and as we go through them some patterns start to emerge. Especially, by the time we get to the last section on input validation, students probably won't need to be sold on the dangers of un-checked user input!

Give the CSRF hack a try or two before attempting it in class. It's perfectly sound, but it does have to be run in just the right sequence to work – which reflects the challenge faced by the would-be hacker in perfecting a CSRF attack on a real user and web application. Another one that's worth some careful testing beforehand is the JPQL injection attack where subqueries are injected through the new-username field of the form for changing profiles. It's a subtle enough hack that it will require some careful explanation in class.

Chapter 4

This chapter is shorter and more manageable than the previous two. But it also contains a lot of information that will be new to most students: concise explanations of how cryptographic processes achieve various desired results, and how keys and certificates work to establish trust relationships. So your lecture here may be longer than the page count would indicate. Also, leave some time for the HTTPS demonstration, as you will probably have to walk individual students through the browser process of installing the new server certificate. Make sure everyone follows along with you in that demo: point out that later labs will not work right until they get the server certificate in place and trusted by the browser.

The shorter section that ends the chapter, on two-way SSL and client certificates, is probably the most likely to be jettisoned. It's interesting stuff, but for most corporate web applications client certificates are a non-starter, and very few students' companies will be going so far as to base fine-grained authorization policies on different user certificates. If however you deliver this class in a government/DoD or DoD-contractor context, it may be a whole other story.





Chapter 5

It seems to have become nearly a conventional wisdom that a web application is “secure” if it has solid authentication and authorization policies and runs over HTTPS. Chapter 3 should have done a fair amount to contradict this idea already. But especially developers seem to shy away from direct practice of cryptography, and in this chapter I aim to illustrate several areas in which application-level cryptography is both necessary and reasonably straightforward to implement.

In the demonstration of hashing passwords, for anyone using the Eclipse overlay, make sure to mention that the **Context.xml** is to be edited directly (adding **digest=“MD5”** to the **JDBCRealm** declaration) rather than altered indirectly via **build.properties** as described in the coursebook.

Probably the one major technique for which we don't provide hands-on exercise is encrypting values in persistent storage. You might want to mention this, and note that the practice is really just a combination of the encryption we do in the second lab, and the interactions with the DAO that we do in the password-hashing demo.

Chapter 6

This chapter goes by very quickly, compared to all the others. Here we discuss some basic best practices that just don't fit into those other categories: error handling, logging, auditing, and a few matters of sound, secure development practice. The final lab is simple, not a bad example of an auditing implementation but not really anything new in terms of technique. The back-to-back examples in the middle of the chapter will probably be more interesting: here's a last opportunity for an interesting group discussion as we consider the relative merits of account lockout and progressive delay as strategies for discouraging brute-force attacks on a login page.





Revision History

Version 5.0 is the initial release of the course.





Troubleshooting

If you run into any trouble with code exercises, the first and best thing to do is to double-check that the classroom machines have been set up precisely according to the course setup guide. Especially, the wrong version of a tool can cause significant problems; don't wander off-book in this way unless absolutely sure you can support the software that you prefer and that we haven't tested. Check environment variable settings carefully, too; these are the cause of a great many classroom glitches.

Below are some specific pitfalls that have come up in previous offerings of the course:

-





Errata

At this time there are no errata for the course.





Feedback

We very much appreciate whatever feedback we can get on our courseware – especially from the instructor's perspective. Naturally, the more specific, the better, and we strongly encourage you to make notes on issues you may encounter in the classroom, whether they're typos, missing files, or suggestions for clearer language to explain a concept. We can't guarantee that we'll act on every suggestion, but we're aggressive about stamping out problems and try to be highly responsive. Hopefully this means that when you give us good feedback, you get a better course the next time you need to teach it.

Please direct all courseware feedback to

Will Provost
Capstone Courseware
<mailto:provost@capcourse.com>
877-227-2477

For anyone who's interested, we have a very informal defect-tracking system, based in Excel spreadsheets with columns to capture defect location, nature, status, and author feedback. Ultimately, feedback goes into these sheets, so if you want a template, we'll be happy to provide one, to facilitate the reporting process.

