



Capstone Courseware, LLC

33 Boylston Street
Jamaica Plain, MA 02130

877-227-2477
capstonecourseware.com

Java Development for Secure Systems

Version 6.0

Instructor's Guide



Overview

It is true of many technology topics that various people will have widely different expectations of a course in that topic; but I think it's unusually so for Java security. There are security aspects to code design and implementation strategy, authentication practices, authorization policies, network protocols, server administration, and several other areas of practice for each one I just listed. The 5.0 version of this course focused on Java SE security – specifically the code-security model, signed JARs, and JAAS – and since its release we've seen a clear pattern of feedback to the effect that people want more on web applications.

So this new version balances things out a bit, with a little less on Java SE, a little less on JAAS, and more on Java EE security and web applications in particular. The single best-practices chapter from before has been split into an SE chapter and an EE chapter that now closes the course. This is still first and foremost a course on Java SE security, but typical delivery might spend as much as half of the class time on JAAS, Java EE security, and common web-application security practices.





Timeline

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

Day 1

4 hours	Chapter 1
2½ hours	Chapter 2

Day 2

2 hours	Chapter 3
2 hours	Chapter 4
4 hours, runs to day 3	Chapter 5

Day 3

2½ hours	Chapter 6
2 hours	Chapter 7

The main thing is, don't panic if it's getting late in the first day and you're still on Chapter one! There's a whole lot of information to communicate there, and though I've looked for it there just doesn't seem to be a sensible way to break this topic down into smaller units. So we have a sort of behemoth chapter to get started. But it, like the rest of the course, is very heavy on quick, targeted examples and interactive demos – so hopefully students will still be with you by the end of it. You can certainly opt out of Lab 1B and the whole dynamic-policy section, if there's low interest and/or time's running short.

The next few chapters should hum right along. For most audiences, Chapter 4 is the best candidate for skipping if you need to re-capture some time for later chapters, and parts of 2 and 3 can be compressed as well.

Chapter 5 is another big boy, but it includes as much demo and lab content as it does specifically for groups that know they need to learn JAAS in depth. For most audiences, cover the chapter normally but have students do either Lab 5A or 5B, but definitely not both. 5C can be reviewed instead of done step-by-step, too.

Chapters 6 and 7 form a mini-module on Java EE and web application security. You can simply let student interest level guide you as to how much time to allot for these. Note that the XSS example and demo in Chapter 7 occupy a lot of book space and will take a while to work through – this is just the nature of the beast as XSS is a complex attack vector.



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. The workspace and projects have been tested lightly with the course but are not part of the standard product. Also, owing to the nature of a course on security, there are many tasks that must be carried out beyond simply compiling and running, and so relatively few of the projects can be tested or debugged in Eclipse without some additional student effort. (See **c:\Capstone\JavaSecurity\Eclipse\Docs\JavaSecurity Module Notes.html** for a rundown of all the reasons for this.)

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 **c:\Capstone\JavaSecurity\Eclipse\ReadMe.html** for general notes on how to use the Eclipse overlays for Capstone courses. Be prepared to walk students through the first few exercises in Eclipse; the notes in this file are for experienced Eclipse users, and will not be clear to many students on their own.

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

As mentioned in the timeline section, this chapter is a bit of a monster. What it does have going for it is a nice linear progression of topics, such that each new concept should be easy to digest: security manager to access controller to permission to policy. At the same time, the basic code-security model does have several major moving parts, and often the biggest challenge of this chapter is to reach the point where students see that whole picture, without getting confused along the way by details of one part before other parts have been introduced. Try to get students to accept a little hand-waving over the latter concepts – protection domains and policies – while you establish the earlier ones.

You may also run into some impatience among students who came to the class looking for Java EE security training. They may not think they have any real interest in or need for the SE model – and they may be right! But it's often a good move to point out the parallels early on, by first noting that a Java EE server is really nothing more than a very sophisticated, standalone SE application – and one that's publicly available and often responsible for sensitive resources. Essentially these are the points made at the beginning of Chapter 6, and it may help to forward-reference them in the early going. How will a Java EE server protect itself and its host from malevolent or (more likely) carelessly-written code?

One note on the demonstration where we set up a security policy to govern the SOAPSniffer application: as of the final step, the application is actually not able to function fully. We've given it sufficient permissions to start up and to listen for traffic, but the moment it gets a request, it's going to need additional socket permissions: accept and then connect. You might illustrate this by simply opening a browser and directing it to "http://localhost". You'll see the running SOAPSniffer crash with an access-control exception. If students are having fun and there's time, you could get all the way to letting the application pass HTTP traffic.

An interesting point about the start of Lab 1B that might be worth posing as a question in class: how is it possible for the project-manager application to set its own security manager and policy, when it's running under a security manager to start with? Of course the answer is that the **NIH.jar** is granted all permissions, but note that there are really two security manager instances, and two policy instances, in play for this application: one right at the start, and then the dynamic policy after that.



Chapter 2

This chapter is, largely, an extension of the first one. Some topics just had to be pulled out to a separate chapter, and the details of key management were severable in a way that many other topics were not. Here we learn about keys, certificates, and keystores, and in the process we begin to expand on our policy vocabulary, because now we can grant to a signer as well as a code source. (Chapter 5 will complete this progression, of course, by bringing in principals.)

Often the toughest concepts to convey are those having to do with certificates, certificate chains, and certificate authorities. There's a Russian-doll quality to this problem that frustrates many students, as certificates are supposed to solve the problem of trust but sometimes seem only to perpetuate it. The key point, and the one that usually ends the discussion (satisfactorily!), is that a root CA solves the trust problem not by being technologically special in any way, but by transcending the technical domain completely, and allowing the trust chain to travel into the business domain – which is where questions of who trusts whom ultimately should be solved. Digital signatures are a great invention, but they don't tell you who's trustworthy and who isn't – and they couldn't possibly.

Chapter 3

This chapter moves away from technical details and API specifications, and focuses on certain widely-accepted techniques of secure code design. As such it's naturally a lot more subjective, and you may get some spirited questions and discussions here.

Do be sure to cover the lab in this chapter – at least as a demo or a thorough code review if not as a full, step-by-step lab. For one thing, the LandUse application that's introduced here is used heavily the rest of the way, and for another, this is a nice example of defining and using a custom permission class, which is mentioned in Chapter 1 but not demonstrated.

Chapter 4

Most audiences will either have a known requirement for cryptography, or they won't. Or, it's also pretty common that there's a need for signatures but not for encryption. This material is easily from the rest of the course, and this can be an asset: it means that each piece of the chapter is a nice self-standing section, easy to digest and move on or easy to skip. If there is time and interest, the encryption lab at the end can be a fun one, as students can export their public keys to each other and (giggle) pass secret messages.

Chapter 5

This chapter approaches JAAS in an unconventional way, which is to carry out coding exercises on the authorization part of the API first, with no authentication piece in play at all. This may seem a bit odd, but it has a couple significant advantages. First, it's a way of chipping off one piece of the larger API at a time. That is, you can't have LoginContexts and LoginModules without Subjects, but you can, and we do, have Subjects without the other pieces. So it's more incremental. Second, it immediately ties JAAS in to what's been studied so far, because we see the connection to policy files right off the bat.

The demo on ANDs and ORs with **doAs** and **doAsPrivileged** methods is quite the brain-teaser, and usually students really turn on to it. But it's a challenge – good idea to go over this one with a fine-toothed comb before presenting it, because it's awfully easy to get things backward or confused while standing at the front of the room.

This chapter is potentially the most lab-heavy, with two labs on the application side and one on the provider side of JAAS. But Labs 5A and 5B overlap considerably, and it's not intended that students do both; you'll definitely have to cut something else to make time for both labs in sequence. Lab 5C is an important one though, as it's the only exercise in building one's own login module.

You might want to point out some additional code in the answer to Lab 5A: the callback handler actually handles all standard callbacks, except for the language callback, which means that it's ready to respond to a wide range of possible questions posed by the login module. See the longer if/else chain in the **handle** method, and the additional helper methods that process text input/output, choices, and confirmations. This isn't exercised by the configured login module but it might be interesting code to review.

Another note about 5A: by the end of it, the Bank can no longer process transactions! A nice demo after the lab is to give that a try, and see the failure. Add a line to dump the exception stack trace in **SignatureUtil.verifySignedTextFile**, and see that the policy file created at the end of the lab forbids the application from reading the transactions file! Change the policy to allow all file reads, and things will work correctly again.

The optional material at the end of 5B will rarely be of interest to enough students to make it worth much discussion. It is illustrative, but maybe it's as much about JAAS' limitations as it is about JAAS' usefulness. But you might sound students out about their level of interest in JFC, and then allocate time appropriately. Do note, though, that the problem with JFC event-handling threads will show up again in a place more near and dear to some hearts, and that is in Tomcat's JAAS implementation. More on this in a moment.



Chapter 6

For this chapter we have to change gears in a couple ways. One is we're no longer studying a coherent, single architecture, as we were for Java SE, because there are still several overlapping approaches in Java EE that have yet to be sorted out into anything all that consistent, or that integrates with Java SE completely. The other is that we have to set at least one foot outside the area of standards and portable code, because so much of Java EE security is left to the server vendor. To this end we focus on one popular tool and use it for hands-on exercises: this of course is Tomcat.

The chapter covers pure-Java-EE approaches such as web security constraints and HTTP authentication, and then tries to tie it all together by showing the Tomcat JAASRealm as a clean integration of JAAS with Java EE. The demo and lab that get JAAS and Java SE policies working for a Java EE web application can be rewarding, and we've simplified the process considerably by having the **buildLoginModule** script and the Ant build copy necessary files to Tomcat's **bin** and **lib** directories.

During any of the later demonstrations and labs that involve on HTTP authentication, if you have extra time or strong interest in the subject, you might show off the actual HTTP headers involved. An example project not mentioned in the student guide is **HTTPSniffer**, which will forward packets from one local port to another (8079 to 8080 by default), and show the traffic in the console and in a log file. Try putting it between the browser and server for any Java EE application that declares a login configuration.

If you want to show the LandUse web application and JFC application sharing the **Proposals.ser** file, do note a quirk of our build processes here. **buildLoginModule** must occur before the Ant build for the web application, but it must happen after the **build** of the JFC application; and to gum things up fully, the Ant build will remove the JFC application! So the sequence that works, even if it's a little silly, is **buildLoginModule**, then **ant**, then **build**, then **buildLoginModule**.



Chapter 7

This chapter is a counterpart to Chapter 3. As that chapter departs from the technical details of SE security to focus on best practices, so here we step beyond servlet security declarations and APIs and talk about practical choices for securing web applications. A great deal of this chapter is occupied with the cross-site scripting attack, which is of course the most problematic area for web applications right now (according to OWASP anyway). To illustrate XSS takes time, as it is a subtle style of attack; students generally find these examples to be real eye-openers though and it's worth working through them in detail.

Students will also often have suggestions of simple ways to stop XSS attacks – why don't we just set the browser to refuse to make any outside connections? or let's just never echo any user inputs to response pages; etc. But there are only a few tried-and-true techniques to defeat XSS, or arguably only one, which is to escape markup characters in dynamic output. Point out especially that client-side fixes aren't satisfactory, because XSS has the unique property of requiring a gullible user but also complicity on the part of our application. Asking users to set their browsers in some way isn't practicable for a large site, and ultimately even if the user does something stupid we can assume that the owner of the website will take most of the blame if any real damage is done.

The auditing lab at the end of the course is really more review than new stuff, but it does introduce two new techniques: servlet filters, which are good general-purpose tools for security implementations, and programmatic security through **getUserPrincipal/isUserInRole**. (Remind students that servlet filters are also the way to patch up a web server that doesn't associate JAAS subjects with request-handling threads.)

Revision History

Versions 6.0 updates the course for Java SE 6 and Java EE 5. It also restructures the material to put more time into EE and web applications, and adds some new topics. Major changes are summarized here:

- A new code-injection attack is illustrated in an additional step of the **Science** case study, in the context of Chapter-3 discussions of factories, singletons, and **final** classes and methods.
- Chapter 3 has been pared down, partly by moving some material to a new Chapter 7 and partly by focusing more tightly on a few key topics such as code injection and serialization vulnerabilities.
- The four labs for Chapter 5 have been condensed to three. The old Lab 5C is now just an optional step at the end of 5B. This was rarely done as a lab anyway and so we're treating it more as an optional/challenge lab for those who've finished 5B and have time to spare; or as an interesting final step for instructor review after the lab.
- JAXB 2.0 is built into Java SE 6, so there are no separate JAXP and JAXB directories and environment variables, and the scripts to generate JAXB classes are simpler. Also we don't see the huge pile of warnings that we used to get, because JAXB 2.0 generates classes that make proper use of Java-5/6 generics.
- Web applications now build using Ant, which simplifies things and spells the end of those split files (LandUse1.xml and LandUse2.xml) that we were using before when we built using DOS batch files.
- JAAS configuration for Tomcat is simpler now. The scripts that build login modules automatically copy them to Tomcat's **lib** directory, so there's no need to change Tomcat's classpath variable. The Ant builds copy the JAAS config files, and if necessary the XML files with user information, into the Tomcat **bin** directory. The only external steps that are needed now are to change the **java.security** file by adding login configuration URLs, and to stop and restart Tomcat itself.
- The new Chapter 7 carries a few best-practice topics from the old Chapter 3. This means that we're able to wait to talk about things like MVC and input validation until after we've actually covered web applications! Then the chapter adds a large section on XSS, and a lab on auditing using servlet filters.

Versions 1.4 and 5.0 are the initial releases of the course. Each is intended for use with the corresponding version of the Java 2 SDK.





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:

- You can run into unpleasant surprises with the configuration of the machines, especially where multiple JREs and/or SDKs are installed. This tends to crop up for the first time in the SOAPSniffer demo near the end of Chapter 1: if you edit the **java.policy** file and don't see any changes in behavior, there's a good chance you're running a different **java.exe** than you think. To be sure, try setting the executable path to just your own **%JAVA_HOME%\bin**. (In later exercises, some build scripts use **xcopy**, which is in the Windows **System32** directory; so be sure to build up a more complete path, once you've diagnosed the problem.)
- If the course software is installed to a root other than **c:\Capstone**, most source files will adjust, by reading the value of the **CC_MODULE** variable at runtime. It's important to start Tomcat from an environment with this variable set as well – the LandUse web application, for one, reads this variable and will of course be running in the server process. If the web application fails to show the three proposals in the data file, double-check that **CC_MODULE** is set globally or in the DOS console from which you're launching Tomcat (or, in the environment from which you launch Eclipse, if you're using Eclipse to host Tomcat and test the web applications – which you probably don't want to do! see the earlier section on the Eclipse overlay).
- One source file that will not adapt cleanly is **Results.jsp** in the LoveIsBlind application; this file will need to be edited by hand to find the **DB.xml** data file if the course is installed to another path or drive.





- In the Calculator demo, it's important to close the browser and restart when you make the switch from un-secure to secure mode in the server. We haven't figured out why, but even after a server restart, if you reload the page from the browser after initially requesting it from the un-secure server (that then crashed), you'll probably get access-control exceptions in Tomcat, having to do with access to certain Catalina constants. Just stop both server and browser, restart both, and try again.
- In the final demo, it's easy to forget to type the protocol **https:** If you get a blank page in the browser, it's usually because you typed **http://localhost:8443/...**





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.

