



**Capstone Courseware, LLC**

33 Boylston Street  
Jamaica Plain, MA 02130

877-227-2477  
capstonecourseware.com

# Securing Java Web Services

**Version 1.4**

**Instructor's Guide**



## Overview

The industry has been grappling with the problem of security for web services for several years, and there is by no means a single set of standards just yet. This course attempts to focus as much as possible on the broad concepts of service security – especially message-level security – and on individual standards that have been finalized and enjoy wide acceptance, such as XML Signature and Encryption and WS-Security. As a practical matter we need some implementations to support our work, and of the ones we use only one – the XML signature API – is any sort of standard.

Generally we prefer to deploy courseware that relies on accepted standards, because it helps to assure that what skills students develop will translate to their own work. That proposition is less compelling in this case – this is just the nature of the beast, but it may take some explaining. There's a gap between the standard technology (the W3C and OASIS standards mentioned above, and the use of JAX-RPC handlers) and the things that will bring the most immediate rewards (practical tools such as XWSS and OpenSAML). Try to emphasize the standards and patterns as much as possible, and remind students that while most of the later course exercises use APIs that may or may not be part of their company's architecture, they nonetheless prove the concepts of message-level security and that the SOAP messages they pass are excellent examples of interoperable security tokens, signatures, and encrypted data.





## Timeline

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

### Day 1

2 hours	Chapter 1
1½ hours	Chapter 2
3 hours	Chapter 3 – partial

### Day 2

1 hour	Chapter 3 – completed
1½ hours	Chapter 4
2½ hours	Chapter 5
1½ hours	Chapter 6

### Day 3

1½ hours	Chapter 7
2½ hours	Chapter 8
2½ hours	Chapter 9

Two options are possible that produce a roughly two-day timeline:

- Cut the chapters on SAML, or treat them very lightly. This can be appropriate for an audience that is interested in the W3C technologies and WS-Security, but has no immediate need for SAML – a common situation.
- Cut or skim the chapters on HTTP security, XML Signature and Encryption. This makes possible a two-day study of the OASIS standards, and still gives exposure to signature and encryption by way of WS-Security implementations.

On the other hand, groups with less than solid understanding of Java and Java web services, including SAAJ and JAX-RPC, and who want to cover the whole range of course topics, may need four days to work through this material.





## Tools Deployed with the Lab Software

This course's software requires separate setups of the J2EE 1.4 SDK, the Crimson text editor, and/or Eclipse 3.x; otherwise it is self-contained, as the lab installer sets up not only the lab software but necessary tools:

- **The Java XML Digital Signature API**, or JSR 105. Chapter 3 builds use the JAR files from this distribution.
- **Apache XML Security**. Same story here, for Chapter 4 labs on encryption.
- **XWSS**. This is the main tool that supports our work with WS-Security. Builds in Chapters 5-9 use these libraries, but beyond that, the JAR files must be deployed to the application server. Instructions in Chapter 5 describe how to carry out this installation; it's a simple matter of running a prepared batch file that copies JAR files, and doing a simple edit to the server's configuration.
- **OpenSAML**. This supports work in the SAML chapters.





## Ant Build Process

We use **asant** as provided by the J2EE SDK for all our builds. Though most students will be happy to let the **asant** 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\JavaWSSecurity\Ant**. Information here defines a routine for building a Hibernate application.

There are three main types of Ant builds. For standalone applications, the build just compiles Java classes and makes sure that necessary JARs are in the class path. The provided **run.bat** script simplifies the launching of a target application class.

For web services, the build also undeploys the service, performs JAX-RPC compilation, assembles a WAR file and then an EAR file, runs the J2EE verifier, and deploys the new version of the service.

For service clients, the build is mostly like that for standalone applications, but it also does the JAX-RPC compile based on the service's deployed WSDL. Always have the service built and completely deployed before trying to build the client. (That is, don't try to build them concurrently.)

For both services and clients, when using XWSS, the one less obvious trick involves the configuration and properties files that parameterize the handlers. These need to be colocated with the class files, and so we need to copy them from the **src** tree to the **build** tree. This means that the service must be rebuilt and re-deployed, or the client rebuilt, after any change to **security.properties** or the XML configuration files.





## Eclipse Overlays

Capstone Courseware provides an optional package of workspace and project files for Eclipse 3.2 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 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.

Our Eclipse overlays have a weak spot where web services are concerned. Specifically, they are not built to anticipate the code-generation that will occur in a JAX-RPC WSDL-to-Java compile. Unfortunately, in this course, all the clients and nearly all the services have this step. This means that almost all of the Eclipse projects will show errors when they are opened.

The good news is that this doesn't have to get in the way of using Eclipse as a code editor. The files that will show errors for this reason are the service implementations and the client applications – because the service endpoint interface, JAX-RPC value types, and service stubs will be missing at the start. But almost all of the Java coding happens in JAX-RPC handlers, which are independent of the service models and therefore of the WSDL compile. So students can open those files and work in them without distractions. And the final build and deploy must be done from the console in any case.

Other notes:

- The source files in this course's projects are in a subdirectory **src** of the main project directory. To make Eclipse happy in attaching to this directory structure, we need the **JavaSource** folder that you see in each Eclipse project – this attaches to the **src** directory. But then, to see the remaining resources in the exercise, we need a second folder that attaches to the root directory of the exercise: this is the **Resources** folder. This is not intuitive, and students will need some guidance. Especially, the **Resources** folder is important because it gives access to the **build.xml** which can and should be used to build and deploy the project using Ant.
- The **DSig**, **J2EE**, **OpenSAML** and **XMLSec** folders are only there to hold JARs which are then placed in the classpath.

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.



## Teaching Notes

### Chapter 1

This overview chapter is designed to be appropriate for those taking the entire course and also for audiences who will not learn the details of the W3C and OASIS specifications or the Java implementations but who do want a high-level understanding of the terminology, concepts and issues. There are no hands-on exercises, not even a code demonstration; it is more of an overture for the rest of the course.

Still, there is a lot to discuss here. Most students will be curious to understand the positioning of web-service security technology, and to learn what sorts of tools are used to solve what sorts of problems. Especially, this is the time to make the argument for message-level security as a distinct solution from transport-level (HTTPS) – maybe better overall, or maybe not, but certainly able to solve a class of problems that TLS cannot.

### Chapter 2

This quick chapter illustrates what is really more of a web-application technique, which is HTTP BASIC authentication. But the demonstration does highlight some interesting issues specifically for web services: who supplies the logon, since there's no interactive user? And how can the build continue to rely on the WSDL if it's been secured?

Not mentioned explicitly in the coursebook is the fact that the JAX-RPC reference implementation does support HTTPS – see the bibliography for a link to documentation on this. Most other implementations will, too, at this stage of the game.

The SOAPSniffer that's introduced in this chapter will be an essential tool the rest of the way through; make sure students are comfortable with its use.





### Chapter 3

Now we get into the nuts and bolts; this chapter and the rest of the course will be much more detailed and laden with hands-on work. We cover signature in good detail here, partly because it will help students understand the effects of using a WS-Security implementation to sign their messages later, and partly because at some point many applications develop a real need for direct implementation of signature and/or encryption of data or messages.

This is also where we introduce the JAX-RPC handler as a key technique in any application-level implementation of message security. Hopefully the students have a good grounding in this already, and this section will be review; if not, it's worth taking plenty of time to investigate that Lumberyard example, because handlers will be central to almost every lab in the course.

The tools introduced in this chapter are helpful in later exercises. The SOAPSneak should be especially fun, both in this chapter and Chapter 5, as it allows students to simulate real service-oriented attacks.

### Chapter 4

This chapter is another quickie, mostly because encryption is such a deep subject, and much more than signature it is something usually best left to professionals and/or generic tools. The theory in this chapter outstrips the practice. If we wanted to perform a real, practical test of symmetric encryption plus asymmetric key wrapping, we'd need more tools (hello, Bouncy Castle), a more complex setup, and a lot more detailed coding. The exercise that's given already takes students into some pretty deep waters. So we leave it at that.





## Chapter 5

This chapter is usually a lot of fun to cover. It brings in a number of big concepts, introducing WS-Security and the various token types. It gets us started with real message-level security examples. And the ability to modify existing client-to-service traffic by plugging in different XML configs, and see fully-worked examples of WSS headers as a result, all this tends to perk students up. Though the sample messages are of course quite detailed, it's worth taking plenty of time to deconstruct them, following the references and IDs around the message header elements and the body itself. Again, as a standard for interoperable security, WSS is much more interesting (or should be) than XWSS, so try to channel the discussion in that direction as much as possible.

You may trip across a quirk of XWSS in your demonstrations, or students may during the labs. You'd expect that an XWSS processor would simply ignore a message feature it wasn't configured to process, such as a signature header that it didn't happen to require. No: it will choke on the signature if it isn't told explicitly that it requires the signature to be there. Not quite so for username tokens, but the XWSS processor will attempt to authenticate the user represented by such a token – whether the configuration tells it to or not. This is not a reflection of WSS itself, though it's not really a compliance failure, either.

## Chapter 6

This chapter extends the study of WSS with some interesting practical issues. It also presents more of an opportunity to get hands dirty in the Java code, for those who've felt a little too much like spectators plugging in those XML configs and watching XWSS take it from there. The timestamp processing is especially instructive; the second lab is interesting work, but it doesn't really introduce any new features and is more a reinforcement of what we can do with JAX-RPC handlers. In fact you might mention here that, if one chooses to use the enhanced JAX-RPC compiler that comes with XWSS (and commit to a specific server product in the process), configurations can indeed be specified per operation as part of generating the souped-up JAX-RPC stubs and ties.



## Chapter 7

SAML is just large enough a language that we break it out into the following two chapters; this chapter serves as a general introduction, and has no hands-on exercises. It's a good place and time to talk about the general usefulness of SAML; where and why it is used; how it relates to LDAP, ADS, and so on. A nice pithy little statement that helps to establish SAML as useful apart from WSS itself is that WSS sets a standard for carrying security information in the headers of messages between parties who are interested in other things; while SAML (especially SAML P) expresses information that might also be the actual body of conversation between two parties whose primary interest is the security assertion itself.

## Chapter 8

Most of the information in this and the next chapter is pretty pedestrian, to be honest: just a lot of details about a particular XML model. There are some interesting concepts tucked in there though, and hopefully the lab exercise helps to drive home the point that with SAML we can express a much broader range of security information than WSS is designed to support.

## Chapter 9

The SAML protocol is more or less just a glue layer between SAML assertions and transports such as HTTP and SOAP. OpenSAML makes this crystal-clear by making the translation from one to the other a matter of a single method call in either direction. This chapter is again mostly just model details, but the final exercise should prove interesting, especially because it takes us into the sort of multi-node, federated service architecture for which WSS and SAML are really designed.





## Revision History

**Version 1.4** 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:

- We've tried to design the error handling in these exercises to provide as much immediate feedback as possible when the student missteps in some way. If a client application shows an exception message, but the cause isn't evident, have the student try running the traffic through the SOAPSniffer. This will often show the full exception stack trace of the root cause in a SOAP fault detail element.
- Beyond this, there are a few sorts of problems that will only manifest themselves on the server side. Whenever there's an error observed by the client of a service that is not sufficiently explained in the SOAP fault or other evidence, get students in the habit of viewing the server log file:  
`%J2EE_HOME%\domains\domain1\logs\server.log`. This will show exception stack traces that couldn't be caught and packed into the SOAP response. They may even want to set up a desktop shortcut to this file.





## Errata

There is one outstanding issue with the lab software:

- Beginning with Lab 8A, the application server will not be able to redeploy the Hospital service without being restarted. This is fairly painless, it only takes 30 seconds or so, but it's a little irritating nonetheless. One technique that will save some aggravation is to have students run **asant build** as they're doing their Java coding, and once they're ready, run **asant** as usual to build and deploy. That's when they'll need to stop the default server and start the default server again. (The Assurance service doesn't have this problem.)





## 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.

