There's Something Stuck In My Shoe!

Reflections on the adoption of fine and course grained authorization frameworks

Larry Koved
IBM T.J. Watson Research Center
Yorktown Heights, New York 10598
koved@us.ibm.com

ABSTRACT

A usable system has many layers. There is the end-users' experience through web sites, interactive voice response systems, ATMs, etc. Below these interfaces are the tools and technologies to create and operate these systems. Security of deployed systems is often dependent on the functionality and usability of these underlying technologies. This paper focuses on usability issues surrounding these underlying security technologies and our attempts to transfer these into products. Specifically, technology related to *Java 2 Standard Edition* (J2SE), *Java 2 Enterprise Edition* (J2EE) and Web 2.0 mashup security.

General Terms

Security, Human Factors, Standardization, Languages.

Keywords

Technology Transfer, Java Standard Edition, J2SE, JSE, Java Enterprise Edition, J2EE, JEE, Eclipse, IDE, Static Analysis, Netscape Navigator, Applet, Permission Analysis, SWORD4J, Mashup Security, OpenAjax Alliance Hub 2.0, Secure Mashups, SMash.

1. INTRODUCTION

Creating secure software systems is a challenge for most developers, architects, system administrators and others involved in the creation, deployment and operation of systems. Much of the software currently deployed, whether for the internet, departmental usage or cloud based software services, is increasingly built on top of complex software frameworks, middleware components, 3rd party software and deployment configurations. Experience with securing composition of these software elements has had mixed results. Securing of such systems is often as complex, or more complex, than the applications themselves. We have seen this phenomenon in the deployment of Java-based systems and browser-based mashups. This paper will describe some of our experiences with securing such systems, and our attempts to deploy usable solutions to securing these systems.

Security & Privacy Usability Technology Transfer: Emerging Research (SPUTTER) Workshop, at the Symposium On Usable Privacy and Security (SOUPS) 2010. July 14, 2010, Redmond, WA, USA.

2. JAVA STANDARD EDITION

In the early days of Java, Netscape Communications adopted Java to provide client-side functionality not previously possible in web browsers. The result was the Applet programming model [Applet] where Java code could be downloaded from one or more web sites and executed in the browser. The marketing slogan was *Write Once, Run Anywhere* [WORA]. From a security perspective, the challenge was to limit code access to sensitive resources, including networking, file I/O, and runtime specific functionality [Dean96]. The initial thought was to assign a security principle to each class that was in the application, otherwise known as the "code source" -- the origin of the code (e.g., host) and/or a digital signature [Gong98]. "Stack introspection" identified which principles were currently executing, and authorization decisions were based on the intersection of the authorizations (Permissions) associated with the executing principles [Gong98, Pistoia04].

Netscape defined a "capability" framework whereby the logic to assign construct an authorization policy database was embedded within the deployed application. Authorization policies for the principles were typically built up through user interaction via pop-up windows. In effect, the end-user became the security policy administrator. Because the Java runtime would periodically change as security vulnerabilities were identified and patched, authorization policy changes would be needed. To the surprise of and end-user, a web site that was working one day could stop working right after Java security patches were installed. Frequently the application developer would not be in a position to (rapidly) update the authorization policy pop-up code to account for the new authorization requirements. This became an annoying and frustrating aspect of using Netscape's Java.

As Netscape was leaving the Java business, IBM proposed to JavaSoft[1] that a static analysis tool be created to identify the authorization requirements / policies for Applets and Java 2 Standard Edition (JSE). The purpose was to identify the authorization requirements prior to executing the Applet, or other J2SE application, and distribute the proposed authorization policy with the code (e.g., in the JAR file [JAR]). The Applet framework in the browser, rather than in the application code, could then prompt for allowing / denying the Permissions based on pre-computed authorization requirements.[2]

[1] During the early days of Java, *JavaSoft* was a business unit in Sun Microsystems that focused on the development and distribution of the Java™ technology.

[2] The alternative would be to use dynamic analysis whereby the authorizations would be captured during runtime, e.g., in the SecurityManager, and generate a policy file from running the test cases. An example of this is jChains [jChains]. The merits and drawbacks of static and dynamic analysis are outside the scope of this paper.

Creating an very accurate Java Permissions static analysis tool that was not overly conservative and complete turned out to be a hard task [Koved02]. Java permissions require one or two String values that (1) specify the protected resource, and (2) the required operations (e.g., read, write, update, etc.) are non-trivial string analysis computations. Lack of path-sensitive analysis could result in over specification of authorization requirements. Also, significant parts of the Java runtime is written in C or C++, which is a non-trivial exercise to incorporate into a Java code analysis.

We did have modest success with several IBM products using our static analysis technology. Our "technology transfer" was largely based on running the core permission analysis algorithms [Koved02] and producing textual reports (flat files and HTML) for the development groups building the products. These reports described required specific authorization requirements. While the reports were sufficient for the products to enable the Java code-based authorization system, product performance could suffer and the security analysis process was very time consuming.

The biggest challenges in the security enablement process was in identifying places in the application code where calls to *AccessController.doPrivileged()* were required, and to verify that these calls were being done safely. Correct *doPrivileged()* placement can require substantial time to trace the call paths to verify (1) the control flow path is likely to be traversed by the application, and (2) no tainted data will slip through and violate the intended security policy. This also would require some refactoring of the source code for the call to *doPrivileged().* We recognized that working from textual reports probably limited wider-scale adoption of the static analysis technology at that time.[3]

In 2000, for server based Java code, security of composite software from multiple sources/origins was not a pressing concern. Code was *not* being dynamically downloaded from the internet into servers. The prevailing assumption was that all component software would need to be trusted when run in a server. This was no different than earlier, non-Java, applications run in servers. As a result, server-side products were not seeing a clear benefit from adopting the code-based Java security model.

However, in about 2003-2004, Java-based desktop applications were being built. The Eclipse Rich Client Platform (RCP) [RCP, McAffer] was intended to provide an underlying runtime to support them. A number of commercial products have been built on this platform. RCP includes the OSGi layer [OSGi], which supports the Java Standard Edition (JSE) authorization model. We took our static analysis technology for Java permissions [Koved02], integrated it into the Eclipse Integrated Development Environment for Java (IDE) to ease many of the time consuming parts of the Java permission analysis. This integration provided inline notification of authorization requirements, and with a few clicks of the mouse, the developer could update the Java authorization policy [alphaWorks, Habeck08]. This tooling also supported a number of other security analyses and limited code refactoring to enhance code security.

[3] This was prior to wide-scale deployment of the Eclipse IDE, so there were multiple choices for source code editing and debugging. Once Eclipse became IBM's primary Java development environment, the choice for our tool integration became obvious for subsequent projects.

Our goal was to get Eclipse RCP updated so that we could turn on the OSGi code-based authorization so that RCP applications could use the Java security sandbox to isolate untrusted code (e.g., Eclipse plug-ins). Since Eclipse is an open source project, the tools needed to be freely available. We met this requirement by making our tool, SWORD4J, available via IBM's *alphaWorks* web site [alphaWorks]. Using SWORD4J [Habeck08], we did analysis of most of Eclipse RCP and its dependent components to gauge the effort required to do the permission analysis. We did this work iteratively, each time identifying additional manual operations and time consuming steps that could be further improved through automation. In the end, we had substantially reduced the time required to perform the analysis. However, the real "technology transfer" hurdles were not technical.

We needed to convince the Eclipse Equinox project [Equinox] that (1) there was a business need, (2) its importance relative to other possible projects, (3) the cost of maintaining the security of the RCP code after the changes were made was reasonable, (4) identify who would make the initial and future code changes to maintain security and policies, (5) there would be support for SWORD4J (or similar tooling) to support this activity on an ongoing basis, and (6) the runtime overhead with and without Java runtime authorization turned on would not be unacceptable to the Eclipse RCP development community.

We built a "secure" version of Eclipse RCP and ran performance benchmarks. The performance numbers were generally very good, although there was modest performance overhead in a few components. We created educational material to describe the Java 2 security model, work required to update Eclipse RCP and dependent components. The big challenge was in getting the developers signed up to do the initial work and the ongoing maintenance. While it was feasible for IBM to build the initial version of securing the code, ongoing maintenance would need to be done by the component owners. There were some organizational changes, and this effort lost steam.

2.1 Technology Transfer Lessons Learned

We were working with a 'standard', Java. We operated under the naïve assumption that system developers would put in the extra resources to make their technologies 'secure'. This was true for systems that needed to conform to a standard (e.g., J2EE compliance) or had customers who wanted systems that were more secure. We had worked closely with the JavaSoft team and understood the technology very well. The original target for Java security had been Applets, which generally were relatively small. Perhaps everyone involved in the Java security technology development underestimated the effort to enable JSE security in large commercial-scale systems.

We had very good working relationships with some of our early customers based on prior work. That helped with the initial 'technology transfer' effirts. In the absence of that relationship, it may have been much harder to have affected the transfer given the cost and technical challenges in effectively deploying JSE security.

Our attempts to transfer the technology to Eclipse RCP were not successful to the extent that we wanted due to the requirements for ongoing support, not just the initial development effort. Each successive round of open source developers to develop and maintain RCP would need to learn JSE security. In addition, significant resources would have been required to test / validate security *each time* there was a new build of RCP. The Eclipse development teams typically worked in six week sprints, so addressing security on such short windows would have required significant resource expenditure, even for small changes to the code base.

Tools to make a security model more usable can not overcome fundamental attributes of the core security technology. JSE security may be technically elegant, but is expensive to implement and maintain in real applications.[4] Without SWORD4J it is technically challenging and time consuming. There are many steps to secure an application. Without tooling, some of the steps may not be feasible in a reasonable amount of time, and they are very tedious (e.g., taint analysis for XSS, CSS, SQLI vulnerabilities). With appropriate tooling, it is *less* time consuming, but still challenging. These same lessons apply to web application development and many other domains. If the security technology you are trying to transfer is cost prohibitive with respect to the resources available (people, skills, compute resources, etc.) and the perceived need (e.g., necessary conformance to a standard), technology transfer is far less likely to happen.

3. JAVA ENTERPRISE EDITION

As opposed to Java Standard Edition, Enterprise Edition (EE) [JavaEE, Pistoia04] was a multi-company collaboration. Unlike with Java Standard Edition, there was no need to address code composition authorization. In addition, significant deployment aspects of EE applications are driven by declarative statements (deployment descriptors) that described the operational environment of the code, including security. Use of declarative security was a positive step since the Netscape experience with embedded code for security pop-ups and shifting authorization requirements for the runtime had been a negative experience. The challenge here was around defining principles and subjects for authorization. Confidentiality and integrity, also declaratively specified, would nominally be handled by SSL. For the purposes of this paper we'll use Enterprise Java Bean (EJB) [EJB, Pistoia04] security as the example. Once the Java EE specification was defined, it would need to be adopted by all vendors that wanted to be "compliant".

The biggest questions were about the principles and resources to be protected. Choices for resources included object, such as an Enterprise Java Bean (EJB), a method in an EJB, or the data upon which the EJB operates. Declaratively defining authorization requirements on an EJB or one of its methods is straightforward. Defining declarative security on the data is more challenging. For example, there are some authorization decisions that can only be made based on data values, not functions. A typical example is authorization to withdraw funds from a bank account. A principle may be authorized for the withdraw funds method, but should not be allowed to withdraw funds from *any* account.

A decision was made to define authorization with respect to operations (EJB methods). The negative effect has been that data-centric authorization requires application specific authorization code. Embedding authorization logic in the applications makes the applications more brittle with respect to policy requirement changes, as we had seen with the Netscape security model. Also, in practice, course-grained authorization policies typically are defined for these applications, thus violating the principle of least privilege [Saltzer74]. As we had done with JSE Permissions [Koved02], we developed a static analysis algorithm to determine the JEE authorization policies [Pistoia05]. Unfortunately this was not a customer problem of great interest since it did not cover data access authorization, just function access authorization. In practice it was easier to define very course grained authorization policies for J2EE applications.

4. Technology Transfer Lessons Learned

A wise (non-security) sage once told me that the way to get a technology into products and have broad impact is to get it into the standard. (The corollary is "be careful what you wish for"!).

Our J2EE security technology transfer was 'successful'. We had very good working relationships with the JEE standardization community. Our security proposal made it into the standard. Security concerns were addressed. As with JSE security, JEE security ended up being focused on access to function rather than the data. That shortcoming has reflected negatively on JEE security and has not yet been addressed at the standards level.

The details of usability JEE security is largely left up to the vendors implementing JEE. It will largely depend how well you can work with the folks who create the security tooling to get the right use cases and think through the complexity of deploying declarative security. Again, the most important issue is to think through, model and validate (where possible) the number of steps required to secure an application, as well as the complexity and cognitive load associated with each step. Authorization, integrity and confidentiality are just a few aspects of the security puzzle. The other aspects (XSS, CSS, SQLI, etc.) are as challenging, if not more so.

In retrospect the usability of the technology can be questioned since there are so many security vulnerabilities discovered in web applications. Although a recent (2010) WhiteHat study of web applications [WhiteHat] found that JSP applications (JSP is a subset of JEE) have roughly the same level of security in practice as many other web application programming models.

5. WEB 2.0 MASHUP SECURITY – SMash / OAA Hub 2.0

An important software trend for web applications is 'mashups':

> In web development, a mashup is a web page or application that uses or combines data or functionality from two or many more external sources to create a new service.[5]

Mashup security was virtually non-existant when we started this work. The "best practices" for building mashups described how to *deliberately bypass security*. As a result, there was significant negative press about mashup security. Because of the investment that IBM was planning in this area, it was important to address the security requirements and change the perception about mashup insecurity.

[4] We have described securing an RCP application when you don't have supporting tools versus when you are using SWORD4J [Koved07, Habeck08].

[5] http://en.wikipedia.org/wiki/Mashup_%28web_application_hybrid%29

We started working with the OpenAjax Alliance[6] (OAA), an industry standards group addressing the software development requirements for interoperable components in support of mashups. There was a proposal to use mechanisms similar to Java Standard Edition (JSE) security for composite software applications in the browser. Based on our experience with JSE security, we strongly recommended against taking that approach. Rather than fine-grained security policies as found in JSE and functional security as found in JEE, we focused instead on course-grained mechanisms that would address information flow security. We proposed using strong isolation of components with well defined intermediated inter-component communication. Our proposal was to use a mediated pub-sub model for all inter-component (cross-domain) communication, where authorization policies could be deployed in the pub-sub mechanism, and focus on information flow policies [DeKeukelaere]. Pub-sub is a well known programming model that has been in use for many years. In addition, many programmers have had extensive experience with Microsoft COM [COM] and CORBA [CORBA], so they are familiar with writing code to communicate with program interfaces.

OAA is made up of over 100 organizations, including most of the large software vendors. Within OAA there are task forces to address specific technical areas. The Security Task Force was formed to address mashup security requirements. This task force had a small active group of participants from several companies, with varying levels of security expertise. We focused on understanding the existing programming models used by the developers of these mashup applications. A key objective was to minimize the number of new concepts and code that would be needed for these developers to adopt secure programming practices. If possible, the communication mechanisms would be secure by default and efficient enough that programmers would want attempt to bypass security due to performance or functional limitations.

Our proposal meshed very well with the existing OAA inter-component communication mechanism, OpenAjax Hub 1.0 [Hub1.0], which was already based on a pub-sub model. We designed and developed the secure communication mechanisms and API's [Hub2.0, DeKeukelaere]. The API's for the mashup developers was a small extension to the existing API's, thus meeting the goal of reducing the number of new concepts and code needed to secure mashups.

5.1 Technology Transfer Lessons Learned

Again, we worked closely with the standards group. We defined the scope of security requirements with the task force, interlocking with the other work groups and proposals in the organization. For mashup security we chose to focus more on information flow security. We were able to piggy back on existing API's and extend a few to cover the cross-domain communication requirements. Many of the security mechanisms are not visible to the programmers.

To be successful, we needed the technology to be adopted by products and/or open source code. So, in parallel, we initiated discussions with several products and services that were building mashup-based tools and systems. Fortunately their customers were asking for mashup security.

As noted earlier, getting the technology into *the standard* implies that others will implement (or adopt) the technology. In this case, we released an open source a reference implementation on SourceForge [HubSource]. OpenAjax Hub 2.0 has been adopted by IBM products and seems to be on track for adoption by other OpenAjax Alliance members. There is ongoing work to incorporate OpenAjax Hub 2.0 into OpenSocial [OpenSocial] since OpenSocial lacks cross-domain security mechanisms.

6. FINAL THOUGHTS

As with all research, security technology transfer is difficult. Having significant supporters can make a huge difference. Although, no matter how much 'support' you may get, the complexity of the security technology may sink your best efforts.

Understand your customer and the target deployment environment. What goals are your customers trying to achieve? Are there security models that can provide a simple security abstraction? How well does that abstraction mesh with the other of the tasks that need to be performed by your customers?

Usability of security technology occurs at many levels. If the underlying security models are complex, and if that complexity can not be abstracted away, then the security technology may not be well understood or appreciated by the target community. This can significantly inhibit adoption by those whom you are targeting.

As Alan Kay said, "*Simple things should be simple, complex things should be possible*." Consider the number of steps needed to configure and deploy security. Are these steps consistent and in alignment with or well integrated into the other tasks need to configure and deploy applications? Keep in mind that the person configuring and deploying the applications may not be the person(s) who developed the code. How do these communities (developers, deployers, and operators) communicate security requirements, assumptions and concerns?

How many steps are required to secure an application? Many (most?) deployers and operators are not security savvy. If there are many steps, you may be asking the security naïve to make decisions about which they lack a well formed base or mental model to draw upon. It may be best to offer secure defaults. Remember Alan Kay.

Having focused on making security and privacy fit into the environment, making it simpler to understand, configure and deploy, you should have a simpler story to 'sell' to your customers – the developers and product organizations.

Cultivate relationships with your customer community. They are the ones you need to adopt your technology. If you have a clear security and privacy story that fits well with their mental model of the deployment environment, they can carry the story out to their customers to validate that it meets their customer's requirements. You need them to validate that your ideas are on target, and have them give you useful feedback on how to adapt and refine your technology. They can be your best allies. Or they totally dismiss you if they don't feel that you understand their (and their customers') needs.

Visit your customer's customers. Understand their security and privacy needs. Even more importantly, understand their business

[6] http://www.openajax.org

and how technology fits into their business. Then you can evaluate where security and privacy fits into their business model. Is your technology a "cost of doing business", or providing value-add to *their* customers? If you are a cost, you will have a harder sell (unless mandated by standards, regulation or law). If you can identify a value-add to the customer, you are no longer a "cost" that strictly needs to be minimized.

7. ACKNOWLEDGEMENTS

Many people contributed directly or indirectly to the technologies described in this paper, including Sumeer Bhola, Marina Biberstein, David Boloker, Ian Brackenbury, Suresh Chari, Hyen V. Chung, Don Ferguson, Matthew Flaherty, Frederik De Keukelaere, Jon Ferriaolo, Don Ferguson, Aaron Kershenbaum, Eric Li, Jeff McAffer, Bilha Mendelson, Tony Nadalin, Nataraj Nagaratnam, Adam Peller, Marco Pistoia, Sara Porat, Jay Rosenthal, Naohiko Uramoto, Sachiko Yoshihama, Mary Ellen Zurko, members of the OpenAjax Alliance and the Security Task Force, members of the Eclipse Equinox project, my former managers who supported the work reported here, and others whom I have inadvertently omitted (I apologize!).

8. REFERENCES

[alphaWorks] http://www.alphaworks.ibm.com/tech/sword4j

[Applet] http://java.sun.com/applets/

[COM] http://www.microsoft.com/com/

[CORBA] http://www.omg.org/gettingstarted/corbafaq.htm

[Dean96] D. Dean, E. Felten, D. Wallach, *Java Security: From HotJava to Netscape and Beyond.* Proceedings of the 1996 IEEE Symposium on Security and Privacy, IEEE Computer Society, Berkeley, May 1996.

[EJB] http://java.sun.com/products/ejb/

[Equinox] http://www.eclipse.org/equinox/

[DeKeukelaere] Frederik De Keukelaere, Sumeer Bhola, Michael Steiner, Suresh Chari, Sachiko Yoshihama: *SMash: secure component model for cross-domain mashups on unmodified browsers.* Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008.

[Gong03] Li Gong, Gary Ellison, Mary Dageforde, *Inside Java™ 2 Platform Security: Architecture, API Design and Implementation (2nd Edition).* Prentice Hall, June 2003, ISBN 0201787911.

[Gong98] L. Gong, R Schemers, *Implementing Protection Domains in the Java™ Development Kit 1.2*. Proceedings of the Network and Distributed System Security Symposium, NDSS 1998, San Diego, California, USA.

[Habeck08] Ted Habeck, Larry Koved, Marco Pistoia, *SWORD4J: Security WORkbench Development environment 4 Java.* IBM Research Report RC24554, IBM T.J. Watson Research Center, Yorktown Heights, New York 10598, May 2008.

[Hub1.0] http://www.openajax.org/member/wiki/OpenAjax_Hub_1.0_Specification

[Hub2.0] http://www.openajax.org/member/wiki/OpenAjax_Hub_2.0_Specification

[HubSource] http://sourceforge.net/projects/openajaxallianc/

[JAR] http://java.sun.com/docs/books/tutorial/deployment/jar/index.html

[JavaEE] http://java.sun.com/javaee/

[jChains] http://www.jchains.org/

[Koved02] Larry Koved, Marco Pistoia, Aaron Kershenbaum, *Access rights analysis for Java.* Proccedings of the 17th ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA), Seattle, Washington, 2002.

[Koved07] Larry Koved, Ted Habeck, *Making Security Accessible to Programmers: Lessons Learned*. IBM Research presentation, 2007.

[McAffer] Jeff McAffer, Jean-Michel Lemieux, *Eclipse Rich Client Platform : Designing, Coding, and Packaging Java™ Applications*. Addison-Wesley Professional, October 2005, ISBN 0321334612.

[OpenSocial] http://code.google.com/apis/opensocial/

[OSGi] http://www.osgi.org, http://www.eclipse.org/osgi/

[Pistoia04] Marco Pistoia, Nataraj Nagaratnam, Larry Koved, Anthony Nadalin, *Enterprise Java™Security: Building Secure J2EE™Applications.* Addison-Wesley Professional, February 2004, ISBN 0321118898.

[Pistoia05] Marco Pistoia, Robert J. Flynn, Larry Koved, Vugranam C. Sreedhar, *Interprocedural Analysis for Privileged Code Placement and Tainted Variable Detection.* ECOOP 2005 - Object-Oriented Programming, 19th European Conference, Glasgow, UK, July 25-29, 2005, Proceedings. Lecture Notes in Computer Science 3586 Springer 2005, ISBN 3-540-27992-X

[RCP] http://wiki.eclipse.org/index.php/Rich_Client_Platform

[Saltzer74] Jerome H. Saltzer and Michael D. Schroeder, *The protection of information in computer systems*. Fourth ACM Symposium on Operating System Principles, October 1973, Communications of the ACM 17, 7, July 1974.

[WhiteHat] http://www.whitehatsec.com/home/resource/stats.html

[WORA] http://www.computerweekly.com/Articles/2002/05/02/186793/write-once-run-anywhere.htm