An Evaluation of Java Applications using Security Requirements

Shachindra Kumar Dubey
Faculty of Computer Science and Engineering Department
Jabalpur Engineering College, Jabalpur (M.P), India

Abstract— The importance of software security has been evident, since it has been shown that most attacks to software systems are based on vulnerabilities caused by software poorly designed and developed. Furthermore, it has been discovered that it is desirable to embed security already at design phase. Therefore, patterns aiming at enhancing the security of a software system, called security patterns, have been suggested.

Web browsers, Web servers, Java application servers and OSGi frameworks are all instances of Java execution environments that run more or less untrusted Java applications. In all these environments, Java applications can come from different sources.

This paper investigates the requirements that need to be imposed on such a container from a security point of view and how the requirements have been implemented by different Java application containers. These security requirements are then used to evaluate the security architecture of existing Java containers for Java applications, applets, servlets, OSGi bundles, and Enterprise Java Beans. For comparison, the requirements are also examined for a C++ application.

Keywords — Java applets, Computer security, Information security, Web security, Risk analysis.

I. INTRODUCTION

Application Server Frameworks contain a comprehensive service layer model. An application server acts as a set of components accessible to the software developer through an API defined by the platform itself. For Web applications, these components are usually performed in the same running environment as their web server(s), and their main job is to support the construction of dynamic pages. In the case of Java application servers, the server behaves like an extended virtual machine for running applications, transparently handling connections to the database on one side, and, often, connections to the Web client on the other.

The Web modules include servlets and JavaServer Pages. Enterprise JavaBeans (EJB) are used to manage transactions. According to the J2EE blueprints the business logic of an application resides in Enterprise JavaBeans—a modular server component providing many features, including declarative transaction management, and improving application scalability.

Some Java Application Servers leave off many Java EE features like EJB and Java Message Service (JMS) including Jetty from Eclipse Foundation. Their focus is more on Java Servlets and JavaServer Pages. A Java Server Page (JSP) executes in a web container. JSPs provide a way to create HTML pages by embedding references to the server logic within the page. HTML coders and Java programmers can work side by side by referencing each other's code from within their own. The application servers mentioned above mainly serve web applications, and services via RMI, EJB, JMS and SOAP. Some application servers target networks other than web-based ones: Session Initiation Protocol servers, for instance, target telephony networks.

Certain aspects of web application security can be configured when the application is installed, or deployed, to the web container. Annotations and/or deployment descriptors are used to relay information to the deployer about security and other aspects of the application. Specifying this information in annotations or in the deployment descriptor helps the deployer set up the appropriate security policy for the web application. Any values explicitly specified in the deployment descriptor override any values specified in annotations.
II. MOBILE APPLICATION SERVERS

A mobile app server is mobile middleware that makes back-end systems accessible to mobile applications to support Mobile application development. Much like a web server that stores, processes and delivers web pages to clients, a mobile app server bridges the gap from existing infrastructure to mobile devices.

The mobile application server will contain both a server operating system and server hardware. These function simultaneously to allow the server to provide remote access and services to apps, which can include authentication, updates, and security features. In this system, the mobile application server communicates with the client component, which operates on the mobile device to receive apps, download updates, and follows the commands of the server. All of these functions occur OTA, or over-the-air. This allows the administrator of the system to configure app settings, send out updates, and wipe all data from a remote location. In a mobile application server, the key features include user management, data redundancy, security of the application and data, high levels of availability, a centralized interface for management, and load balancing. Mobile application servers are beneficial because they allow employees to use the mobile device of their choice, and they offer a simple solution the issues the BYOD platform creates.

Mobile application servers, Application servers, and web servers serve similar purposes: they are pieces of middleware that connect back-end systems to the users that need to access them, but the technology in each of the three differs. Application servers—developed before the ubiquity of web-based applications—expose back-end business logic through various protocols, sometimes including HTTP, and manage security, transaction processing, resource pooling, and messaging. When web-based applications grew in popularity, application servers did not meet the needs of developers, and the web server was created to fill the gap. Web servers provide the caching and scaling functionality demanded by web access and not provided by application servers. They convert requests to static content, and serve only HTTP content. Over time, application servers and web servers have morphed from two previously distinct categories, blended features, and arguably have merged.

The emergence of mobile devices presents the need for functionality not anticipated by the developers of traditional application server developers, and mobile application servers fill this gap. They take care of the security, data management and off-line requirements not met by existing infrastructure, and present content exclusively in REST.

III. WEB APPLICATION SECURITY

In the Java EE platform, web components provide the dynamic extension capabilities for a web server. Web components can be Java servlets or JavaServer Faces pages. The interaction between a web client and a web application is illustrated in given figure.

Security for Java EE web applications can be implemented in the following ways.

- **Declarative security:** Can be implemented using either metadata annotations or an application’s deployment descriptor.
Programmatic security: Is embedded in an application and can be used to make security decisions when declarative security alone is not sufficient to express the security model of an application. Declarative security alone may not be sufficient when conditional login in a particular workflow, instead of for all cases, is required in the middle of an application. Servlet 3.0 provides the authenticate, login, and logout methods of the HttpServletRequest interface. With the addition of the authenticate, login, and logout methods to the Servlet specification, an application deployment descriptor is no longer required for web applications but may still be used to further specify security requirements beyond the basic default values.

Message Security: Works with web services and incorporates security features, such as digital signatures and encryption, into the header of a SOAP message, working in the application layer, ensuring end-to-end security. Message security is not a component of Java EE 6 and is mentioned here for informational purposes only.

IV. COMPUTER AND APPLICATION SECURITY

Ensuring security for Java apps is not a straightforward task because, in today's software development world, Java programmers routinely import thousands of lines of code from third-party code libraries. The imported code is typically used to perform common, generic tasks, such as database access, XML processing, logging, and the like. But when code is open-sourced, there is no single party who warrants its security or undertakes rigorous penetration testing. Therefore, vulnerabilities can be repeatedly introduced into in-house code through this "imported code backdoor." These vulnerabilities may be unknown to the enterprise, but well-known to attackers. When flawed code is present, attackers may be able to attack a server or access data at the back end using SQL injection attacks or other exploits. Developer training and static software analysis tools can reduce the danger, but neither of these safeguards can mitigate vulnerabilities discovered after an application goes into production. A security approach that does not require developer intervention is a better option. Basically, a security bug was found in the Open SSL cryptography library that affected an estimated two-thirds of Web servers on the Internet.

Network-based defense involves Web application firewalls and intrusion prevention systems operating at runtime to shield production systems from external threats. To avoid blocking legitimate traffic, such systems must be tuned quite loosely, making them less effective. Otherwise months must be spent tuning them to admit only authorized traffic. After all, if authorized traffic is blocked, users are blocked. Meanwhile, when network-level defense mechanisms (such as IP firewalls) do block traffic, they use clumsy methods that terminate the network connection. Application testing tools analyze software for the presence of vulnerabilities. The results generated by application testing tools may be highly educational for the app's developers, but they can also be overwhelmingly numerous, making it difficult to prioritize the critical problems versus minor issues and false positives. When vulnerabilities inevitably slip through, such tools do nothing to protect the system during runtime.

A new approach to securing Java applications is Runtime Application Self-Protection (RASP). Implemented by the application runtime itself (or tightly coupled to it), RASP combines real-time analysis of application behavior (what the application is doing right now) with real-time contextual awareness (what has led up to the application doing what it is doing right now). Thus, continuous security analysis becomes a native function of the runtime environment, with the system responding immediately to any recognized attacks. Because security functions are applied from within the application, with instruction-by-instruction awareness of what the application is precisely doing, RASP avoids the false positives that plague other mitigation methods.
V. JAVA SECURITY

In the Java Security API, there is a package, java.security.acl, that contains several classes that we can use to establish a security system in Java. These classes enable our development team to specify different access capabilities for users and user groups.

The concept is fairly straightforward. A user or user group is granted permission to functionality by adding that user or group to an access control list. For example, we consider a java.security.Principal called testUser as shown below:

```java
Principal testUser = new PrincipalImpl("testUser");
```

Now you can create a Permission object to represent the capability of reading from a file.

```java
Permission fileRead = new PermissionImpl("readFile");
```

Once we have created the user and the user's permission, we can create the access control list entry. It's important to note that the security APIs require that the owner of the access list be passed in order to ensure that this is truly the developer's desired action. It is essential that this owner object be protected carefully.

```java
Acl accessList = new AclImpl(owner, "exampleAcl");
```

In its final form, the access list will contain a bunch of access list entries. We can create these as follows:

```java
AclEntry aclEntry = new AclEntryImpl(testUser);
```

```java
aclEntry.addPermission(fileRead);
```

```java
accessList.addEntry(owner, aclEntry);
```

The preceding lines create a new AclEntry object for the testUser, add the fileRead permission to that entry, and then add the entry to the access control list.

We can now check the user permissions quite easily, as follows:

```java
boolean isReadFileAuthorized = accessList.checkPermission(testUser, readFile);
```

VI. AUTHORIZATION EXAMPLES

The preceding examples show how a form-based authentication scheme can work, how the EJB model can restrict access based on a username/password combination, and how we can use the access control lists within the Java security model to restrict access to functionality. Now we intend to leverage these examples to demonstrate an overall system approach to the security problem.

In the first example, the form-based authentication scheme was implemented in a rudimentary fashion. It checked only to ensure that the user was contained in the database of authenticated users. A user contained in the database was granted access to all functionality within the system without further definition.

In the second example, the EJB authorized the user attempting to execute restricted methods on the bean. This security protects the bean from unauthorized access, but does not protect the Web application.

In the third example, we discussed the Java Security Access Control package and considered how we could use a simple API to verify that a user had access to certain functionality within the system.

By gathering these three examples, we can create a simple authentication scheme that limits the user's access to Web-based components of a system, including back-office systems.

VII. CONCLUSIONS

Securing a Web system is a major requirement for the development team. This article has put forth a security scheme that leverages the code developed by Sun Microsystems to secure objects in Java. Although this simple approach uses an access control list to regulate user access to protected features, we can expand it based on the requirements of our user community to support additional feature-level variations or user information. Additional enhancements could include XML and would include the migration of code from a simple object making SQL calls to a bean, possibly even an entity bean.
Java also provides some other additional methods of security ranging from digital signatures to the JAAS specification that can be used to protect the class files against unauthorized access. A simple security approach can minimize our system development time, and vulnerability to malicious attack, while allowing for expanded features with minimal coding effort. As many companies just relearned, however, software alone cannot secure a Website against all forms of attacks.

REFERENCES