Enabling Java 2 Runtime Security with Eclipse Plug-ins

Analyzing Security Requirements for OSGi-Enabled Platforms

Marco Pistoia, Ted Habeck, Larry Koved
IBM T.J. Watson Research Center
New York, USA
1. Motivation for This Work
2. Review of Java Security Concepts
3. IBM Security Workbench Development Environment for Java (SWORD4J)
4. SWORD4J Demo
1. Motivation for This Work
Developing Secure OSGi Applications

- The OSGi framework is a robust platform for deploying and managing applications from handheld devices to servers.
- It supports *security sandboxing* by limiting access to resources applications can access at runtime.
- OSGi security is based on the Java 2, Standard Edition (J2SE) security architecture.
- Eclipse is built on top of the OSGi framework.
Authorization – A Layered Perspective

Applications

OSGi Security

J2SE Security
2. Review of Java Security Concepts
If all the code source was granted Permission p…

Otherwise…

SecurityException

Problem:
What Permissions are required?
• Not too many permissions
• Not too few permissions
import java.io.*;
import java.net.*;
public class LibraryCode {
    private static String logFileName = "audit.txt";
    public static Socket createSocket(String host, int port)
        throws UnknownHostException, IOException {
        Socket socket = new Socket(host, port);
        FileOutputStream fos = new FileOutputStream(logFileName);
        BufferedOutputStream bos = new BufferedOutputStream(fos);
        PrintStream ps = new PrintStream(bos, true);
        ps.print("Socket " + host + ":" + port);
        return socket;
    }
}
## Multiple Permission Requirements

<table>
<thead>
<tr>
<th>Method/Macro</th>
<th>q</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Client.main()</code></td>
<td>q</td>
<td>p</td>
</tr>
<tr>
<td><code>LibraryCode.createSocket()</code></td>
<td>q</td>
<td>p</td>
</tr>
<tr>
<td><code>Socket.&lt;init&gt;(host,port)</code></td>
<td>q</td>
<td></td>
</tr>
<tr>
<td><code>sm.checkConnect(host,port)</code></td>
<td>q</td>
<td></td>
</tr>
<tr>
<td><code>sm.checkPermission(q)</code></td>
<td>q</td>
<td></td>
</tr>
<tr>
<td><code>AccessController.checkPermission(q)</code></td>
<td>q</td>
<td></td>
</tr>
<tr>
<td><code>FileOutputStream.&lt;init&gt;(logFileName)</code></td>
<td>p</td>
<td></td>
</tr>
<tr>
<td><code>sm.checkWrite(logFileName)</code></td>
<td>p</td>
<td></td>
</tr>
<tr>
<td><code>sm.checkPermission(p)</code></td>
<td>p</td>
<td></td>
</tr>
<tr>
<td><code>AccessController.checkPermission(p)</code></td>
<td>p</td>
<td></td>
</tr>
</tbody>
</table>

q = new SocketPermission("ibm.com","80");

p = new FilePermission("audit.txt","write");
import java.io.*;
import java.net.*;
import java.security.*;
public class LibraryCode2 {
    private static final String logFileName = "audit.txt";
    public static Socket createSocket(String host, int port)
            throws UnknownHostException, IOException,
            PrivilegedActionException {
        Socket socket = new Socket(host, port);
        File f = new File(logFileName);
        PrivWriteOp op = new PrivWriteOp(host, port, f);
        FileOutputStream fos = 
                (FileOutputStream) AccessController.doPrivileged(op);
        BufferedOutputStream bos = new BufferedOutputStream(fos);
        PrintStream ps = new PrintStream(bos, true);
        ps.print("Socket ":host + ":" + port);
        return socket;
    }
}

class PrivWriteOp implements PrivilegedExceptionAction {
    private File f;
    PrivWriteOp (File f) {
        this.f = f;
    }
    public Object run() throws IOException {
        return new FileOutputStream(f);
    }
}
### Access Control with Privileged Code

#### Problems:
1. What portions of library code should be made privileged?
2. What permissions are implicitly extended to client code?
3. How can unnecessary privileged code be detected?
4. How can “tainted variables” be detected?

```
q = new SocketPermission("ibm.com","80");
p = new FilePermission("audit.txt","write");
```
OSGi Security Issues

• **Bundle Developers**
  - What Java 2 permissions are required?
  - Are there inter-plug-in permission dependencies?
  - Where should privileged code be inserted?
  - Are there any tainted variables?

• **Administrators**
  - What Java 2 permissions are required?
  - Should all permissions assigned by a bundle provider be trusted?
  - Are plug-ins signed and are digital certificates valid?
  - Are there inter-plug-in permission dependencies?
What Are Your Choices?

1. Leave code unmodified
2. Refactor code so permissions not required by clients. This entails:
   - Moving code into initialization routines, etc.
   - Adding the required permissions to the associated bundle (policy file)
3. Treat the privileged operation as a trusted library function. This entails:
   - Wrapping the privileged operation in a java.security.PrivilegedAction or PrivilegedExceptionAction
   - Adding the required permissions to the associated bundle (policy file)
Traditional Approach

SecurityExceptions due to:
- Client code being insufficiently authorized
- Library code making restricted calls on its own

Limitations:
- Tedious, time consuming, and error prone
- Some permission and privileged-code requirements may never be discovered until run time due to insufficient number of test cases
- Applications may be unstable
A Better Way with SWORD4J

- Interprocedural analysis for automatic detection of:
  - Library code instructions that are good candidates for becoming privileged
    - The candidate instructions are the closest to the library/core boundary
    - The permissions implicitly granted to client code are reported
    - Explanation for privileged instruction
      - “Unnecessary” or “redundant” privileged code
      - Permissions required by code
3. IBM Security Workbench Development Environment for Java (SWORD4J)
Securing OSGi, Eclipse, and Java Code with SWORD4J

1. Determining Java 2 security permission requirements
   - OSGi bundles
   - Eclipse plug-ins
   - Java Applications

2. Authorization
   - Adding privileged code where appropriate
   - Granting permissions by bundle

3. Digital key management
   - Code signing
   - Certificate management

4. Deployment
   - Verify signed bundle signatures
   - Verify granted bundle permissions
4. SWORD4J Demo
SWORD4J Architecture

KeyStore Editor | JAR Signer

JAR Inspection
- Code Architecture Inspection
- Certificate Inspection
- Permission Inspection

Static Analysis Engine (Eclipse and OSGi Aware)
- Access-Rights Analysis
- Privileged-Code Placement Analysis
- Tainted-Variable Analysis
- Call Path Analysis

Call Graph

Java Bytecode Analysis (JaBA)

Security Policy

Object Code
SWORD4J Summary

- New Eclipse-based analysis tool for inspecting and analyzing OSGi bundles, Eclipse plug-ins, and Java programs
- Includes the following plug-ins:
  - Java 2 Security Analysis
  - Jar Inspection
  - Jar Signing
  - KeyStore editor
SWORD4J Features

- Security Analyses
  - Permission analysis
  - Privileged code analysis
- Code Signing
  - SWORD4J provides a JAR signing GUI
- KeyStore Management
  - SWORD4J provides a KeyStore editor for managing Java KeyStores
    - Change password, edit certificate aliases, export certificates, import certificates, generate self-signed certificates, change key entry passwords, copy key entries between KeyStores
- Jar Inspector
  - Displays JAR architecture
  - Displays Signing information
  - Displays OSGi bundle permission requirements.
E-mail:
- pistoia@us.ibm.com
- habeck@us.ibm.com
- koved@us.ibm.com

Web
- SWORD4J Download:
  • http://www.alphaworks.ibm.com/tech/sword4j
- Personal:
  • http://www.research.ibm.com/people/p/pistoiap
  • http://www.research.ibm.com/people/k/koved
- Java Security Project:
  • http://www.research.ibm.com/javasec
- Eclipse security: http://www.eclipse.org/equinox/

Books