KEEPING YOUR CUSTOMERS OUT OF THE HEADLINES
JAVA SECURE DEVELOPMENT PRACTICES

Eric Raisters
Manager, Secure Software Development
Attachmate Corporation
Agenda

• Introduction
• Secure Coding in Java
• Secure Development Practices
• Summary
Introduction

- Who is Attachmate?
- Who is Eric Raisters?
Quick Survey

• How many started programming with Java:
  - 1.7 (2011)
  - 1.6 (2006)
  - 1.5 (2004)
  - earlier versions

• How many develop:
  - Servers/servlets
  - Java desktop application
  - Java web applications
  - Java plugins, applets or WebStart applications
  - Require use of JNI
Facts About Software Vulnerabilities

• “Web applications abound in many larger companies, and remain a popular (54% of breaches) and successful (39% of records) attack vector.” - Verizon Data Breach Investigations Report (2012)

• “SQL injection was the means used to extract 83 percent of the total records stolen in successful hacking-related data breaches from 2005 to 2011.” - privacyrights.org

• 75% of Microsoft security bulletins are for applications

• 70% of successful attacks are through application vulnerabilities - 2012 (ISC)2 survey

• Half of application vulnerabilities are design flaws - 2012 Cigilent-McGraw statistics
## Operating System Vulnerabilities

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Number of Vulnerabilities</th>
<th>Date of Last Vulnerability</th>
<th>Compromised User Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unisys ClearPath OS2200</td>
<td>0</td>
<td>-</td>
<td>No</td>
</tr>
<tr>
<td>Unisys ClearPath MCP</td>
<td>1</td>
<td>12/31/2002</td>
<td>No</td>
</tr>
<tr>
<td>IBM zSeries</td>
<td>18</td>
<td>11/11/2011</td>
<td>Yes</td>
</tr>
<tr>
<td>IBM iSeries</td>
<td>19*</td>
<td>06/11/2009</td>
<td>Yes</td>
</tr>
<tr>
<td>OpenVMS</td>
<td>31</td>
<td>11/26/2011</td>
<td>Yes</td>
</tr>
<tr>
<td>HP-UX</td>
<td>322</td>
<td>04/05/2012</td>
<td>Yes</td>
</tr>
<tr>
<td>AIX</td>
<td>339</td>
<td>03/20/2012</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Java</strong></td>
<td><strong>752</strong></td>
<td><strong>01/27/2012</strong></td>
<td><strong>Yes</strong></td>
</tr>
<tr>
<td>Windows</td>
<td>2983</td>
<td>04/15/2012</td>
<td>Yes</td>
</tr>
<tr>
<td>Linux</td>
<td>3663</td>
<td>04/10/2012</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Java</strong></td>
<td><strong>223</strong></td>
<td><strong>04/14/2012-present</strong></td>
<td><strong>Yes</strong></td>
</tr>
</tbody>
</table>
Secure Coding in Java
# Java Architecture

<table>
<thead>
<tr>
<th>Java Language</th>
<th>Java Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>java</td>
<td>javac</td>
</tr>
<tr>
<td>javac</td>
<td>javadoc</td>
</tr>
<tr>
<td>javadoc</td>
<td>jar</td>
</tr>
<tr>
<td>jar</td>
<td>javap</td>
</tr>
<tr>
<td>javap</td>
<td>JPDA</td>
</tr>
<tr>
<td>JConsole</td>
<td>Java VisualVM</td>
</tr>
<tr>
<td>Java VisualVM</td>
<td>Java DB</td>
</tr>
<tr>
<td>Java DB</td>
<td>Security</td>
</tr>
<tr>
<td>Security</td>
<td>Int'l</td>
</tr>
<tr>
<td>Int'l</td>
<td>RMI</td>
</tr>
<tr>
<td>IDL</td>
<td>Deploy</td>
</tr>
<tr>
<td>Deploy</td>
<td>Monitoring</td>
</tr>
<tr>
<td>Monitoring</td>
<td>Troubleshoot</td>
</tr>
<tr>
<td>Troubleshoot</td>
<td>Scripting</td>
</tr>
<tr>
<td>Scripting</td>
<td>JVM TI</td>
</tr>
<tr>
<td>JVM TI</td>
<td>Web Services</td>
</tr>
</tbody>
</table>

## Deployment
- Java Web Start
- Applet / Java Plug-in

## JavaFX
- Swing
- Java 2D
- AWT
- Accessibility

## User Interface Toolkits
- Drag and Drop
- Input Methods
- Image I/O
- Print Service
- Sound

## Integration Libraries
- IDL
- JDBC
- JNDI
- RMI
- RMI-IIOP
- Scripting

## Other Base Libraries
- Beans
- Int'l Support
- Input/Output
- JMX

## Security
- JNI
- Math
- Networking
- Override Mechanism

## Language Tools
- lang and util
- Collections
- Concurrency Utilities
- JAR
- Logging
- Management
- Preferences API
- Ref Objects
- Reflection
- Regular Expressions
- Versioning
- Zip
- Instrumentation

## Java Virtual Machine
- Java HotSpot Client and Server VM

---

CONFIDENTIAL

[Attachmate logo]
Java Security Features

• Sandbox architecture
• Object oriented through use of classes and libraries
• Access/security through per thread Security Manager and controllers enforcing principle of least privilege
• Policy files used to grant permissions
• Protection domains for classes based on grants
• Type safety (no buffer over/underflows)
• Class inheritance
• Mobile code (write once, run anywhere)
• Final, private, public class declarations
But…

• Sandbox can be escaped any number of ways
• Object oriented design must be done carefully
• Access/security must be mapped to avoid data leakage
• Grants can be temporarily overridden in code
• Protection domains are per thread
• Strict type checking does not apply to JNI code
• Class inheritance not always intuitive or obvious
• Mobile code so attackers can “see” your code.
• Final, private, public declarations can be modified by inheritance
Secure Coding Principle #1
Validate All User Controlled Input

• 6/10 of OWASP Top 10 vulnerabilities, and 14/25 of Mitre/SANS Top 25 coding errors are due to some form of improper input validation

• User controlled data = attacker controlled data - all user data is evil until proven otherwise
  - Check size before using, copying, passing on
  - Blacklists vs. whitelists
  - Proper encoding (local code page, vs. ASCII vs. UTF8/16)
  - Files too!

• If you are a server, do not trust the client to validate

• XSS, SQL injection, buffer/integer overflows
Secure Coding Principle #2
Compile With Strictest Checking

• Compilers are designed to point out unsafe code usage – don’t ignore them
  - Do not use deprecated classes or methods – there is a good reason they have been deprecated
  - Yes, Java code is compiled JIT, but if you ship bad code in class libraries it will be exposed in the customer’s environment
  - Only silence warnings if you fully understand the implications

• Static code analysis tools can also help – especially with legacy and inherited code
Secure Coding Principle #3
Least Privilege

• Most vulnerabilities occur in non-security related code
  – Run with the least possible privileges required to perform 80% of the most commonly performed tasks
  – If you must run at elevated privilege, run for the shortest possible time/lines of code and reduce when task completed

• Minimizes chance of escalation of privilege attacks succeeding
Secure Coding Principle #4
Use the Available Tools

• use the permissions system
  – least-privilege code is safer code
  – use the SecurityManager when extending the API, e.g. Toolkit, Provider, Socket implementations

• use the exception handling system
  – robust code is safer code

• write clean code
  – pay attention to compiler warnings
  – use static analysis tools at broad scope
Secure Development Practices
What is Secure Development Lifecycle

- Addresses the 50% of vulnerabilities due to design flaws as well as prevents most of the 50% due to coding flaws
- Outgrowth Capability Maturity Model developed by the Software Engineering Institute at CMU
- Embraced and extended by Microsoft in 2001 – tools and methods incorporated into IDEs
- Set of best practices in every phase of developing and delivering products
Secure Development Lifecycle

- Engineering Excellence
- Security Development Lifecycle
- Security Response Center
- Feedback
- Improved quality of updates & tools

Copyright Microsoft, 2004
Training and Education

• Most important part of SDL is awareness
• Annual training because attackers and attacks change constantly, so must defenses
• Not every developer will be a security expert, but everyone needs to understand their part in securing products
• Recruit 1-2 members from each team to be SMEs and evangelists
Requirements phase

• Who is the product targeted to?
  - What regulations apply?
    - Health care? HIPAA/HITECH
    - Financial? GLBA, SOX, PCI-DSS
    - Government – FISMA, FIPS
  - What quality/assurance measures apply?
    - Common Criteria, ISO 2700X, vertical specific

• What security features required?
  - Confidentiality? Encryption, secure authentication
  - Integrity? Hashing, digital signatures
  - Accessibility? Fail safe, hardening, throttling
Architecture and design

- Attack surface analysis
  - What does an attacker have access to?
  - Helps focus next task

- Threat modeling
  - STRIDE defines possible attacks (threats)
    Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Escalation of Privilege
  - DREAD helps determine priorities of mitigating threats
    Damage Potential, Reproducibility, Exploitability, Affected Users, Discovery

- Misuse test cases
Threat Mitigation Methods

<table>
<thead>
<tr>
<th>Threat</th>
<th>Mitigation Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spoofing</td>
<td>Authentication</td>
</tr>
<tr>
<td>Tampering</td>
<td>Integrity</td>
</tr>
<tr>
<td>Repudiation</td>
<td>Nonrepudiation</td>
</tr>
<tr>
<td>Information Disclosure</td>
<td>Confidentiality</td>
</tr>
<tr>
<td>Denial of Service</td>
<td>Availability</td>
</tr>
<tr>
<td>Elevation of Privilege</td>
<td>Authorization</td>
</tr>
</tbody>
</table>
Secure development

• Code repository
  - Secure and role-based access
  - Versioning control

• Strict compiler settings
  - All warnings treated as fatal
  - Do not use deprecated functions or classes

• Static code analysis (SAST)
  - Built into most IDEs (Eclipse and IDEA have it)
  - Useful in cleaning up legacy code as it finds backdoors, easter eggs and trojans, as well as deprecated function use
  - Run on third-party code as well
Testing and Q&A

• Fuzz testing
  - File fuzzing (will find 75% of vulnerabilities)
  - Protocol fuzzing – provides resilience

• Dynamic testing (DAST)
  - Finds vulnerabilities in generated code and scripts
  - Exercises logic path dependencies

• Penetration testing
  - Many free tools but requires knowledge to be effective
  - Often contracted to specialists

• OWASP Top 10 for web applications
Install and Maintenance

• All jars should be:
  – Signed with a code signing certificate (especially since this will be the secure default in 1.7u17 and above)
  – Delivered/deployed in as secure a manner as possible

• When (not if) a vulnerability is reported
  – You need an Incident Response process defined
  – You need an Incident Response Team formed and ready, including development, support and legal representation
Implementing SDL

- Define what you do now
- Decide which tasks make sense to do (may be different for different product lines)
- Add the tasks that provide the most return first
- Slowly add the other tasks with each subsequent release
- This is a multi-year process, so don’t get discouraged
- It is a life-style change, so celebrate accomplishments
Secure Java Products

- Coding securely is important, but not enough
- Preventing design flaws equally as important
- SDL process helps to make both part of everyday life
- Be prepared to handle incidents after product deployed
- All of these help protect customers’ investment and security and make them happy
Resources

• National Vulnerability Database
• Mitre/SANS Top 25 coding errors
• US-CERT Top 10 secure coding practices
• Microsoft SDL
• OWASP Top 10 vulnerabilities in 2013
  – OWASP Java project secure coding
  – OWASP toolkit
• Security testing tools
  – OWASP Top 10 Automated test framework
  – CERT fuzzer
  – Peach fuzzer
Contact Information

Eric Raisters, CISSP, CSSLP
Manager, Secure Software Development
Office of Applied Technology & Practices
705 - 5th Ave South, Suite 1100
Seattle, WA  98104
voice: 1.206.217.7855
Eric.Raisters@Attachmate.com