



Security Foundations of Java

Simon Roberts

Simon.Roberts@earthlink.net





Focus/Agenda

- Java platform security
 - Primary goals are this topic
- J2SE APIs
- J2EE approach
- Web services special issues





Java Platform (JVM) Security





Conventional OS Security

- Hardware based
- CPU Kernel Mode
 - Flagged electronically
- Memory Management Unit (MMU)
- SWI Vector Area
 - SWIs enter kernel mode
- Each item protects the next





Java Issues

- No protection of a process against itself
 - (pointer errors and so forth)
- Entire process has a single trust level
- Combination is inappropriate for processes that download code





(What About Code Signing?)

- A useful, but partial, solution
- Doesn't protect against mistakes
- Doesn't protect against malicious staff within trustworthy company
- Doesn't protect if certificates are issued carelessly
- Java provides it too



Java's Memory Model Is Different

- Symbolic references, not numeric addresses
- Symbols are typed
 - Each symbol describes the type of its referee
 - Symbols cannot be misused





Java Bytecode Is Limited

- Restrictions on reference use result from bytecode language
- Bytecode can't express normal pointer dereferencing
 - Only symbol dereferencing
 - `getField #4 <Field int x>`





Java Bytecode Is Limited

- Reference-following instructions are typed
- Every instruction accepts only one argument type list
- Every instruction results in only one argument type



The Bytecode Verifier

- Enforces the rules of the bytecode language
 - `private/final etc.` mean it
- Verifies that program code is typesafe when it is loaded
- Variable accesses are valid w.r.t stack range
- If verification fails, the code never becomes executable





The Bytecode Verifier

- Performs a “proof by induction”
 - Start at each entry point, assuming correct argument type
 - Derive arguments (stack) for every subsequent point in the code
 - Includes effect of all branches, method calls in checks



The Bytecode Verifier

- Some checks must be deferred to execution
 - First checks on method arguments in classes not yet loaded
- Following successful first check, “quick” instructions may be substituted
 - This is not required VM behavior





The SecurityManager

- Checks calling conditions for sensitive functionality
 - What is the origin of the requester call stack?
 - What is being requested?
 - What arguments/parameters are given?





The SecurityManager

- In typesafe, verified, code arguments are relatively reliable
 - *e.g.* immutable `String` objects
- Decision based on security policy file
- `SecurityManager` checks before allowing linking to a native library
 - This controls direct access to natives
 - You must control indirect access using `private final`





The SecurityManager

- Applies to all code except from `bootclasspath`
- So code can't open a file, use a robot, grab the screen...





The SecurityManager

- Disabled by default in applications
 - Enable with `-Dsecurity.manager`
- Default policy prohibits everything, except for `$JRE/lib/ext`





Bending the Rules

- “Applets can’t read files”
- Well, sometimes...
 - *e.g.* to read font information
- Access might be permitted in specially controlled conditions
- This must be provided for in a privileged access method





Bending the Rules

- Access is performed by `doPrivileged(PrivilegedAction)`
- `doPrivileged` prevents the check up the call stack
 - But the caller of `doPrivileged` is still checked



The ClassLoader

- ClassLoader has `private/final` methods that install classes
- These methods invoke the bytecode verifier
 - So verification can't be bypassed
- ClassLoader also keeps track of where the class originated and any signatures
 - This pair defines the "Protection Domain" that governs the privilege set available to the class





The ClassLoader

- Any error in the classloader has the potential to break the “security circle”
 - *e.g.* if a classloader lies about protection domains
- So it’s important to avoid malicious or careless classloaders
 - Never permit downloaded classloaders
 - Create a `URLConnectionHandler` not a `ClassLoader`





The ClassLoader

- Must ensure correct class naming
 - Old (Vn 1.0) bug allowed classes to start with leading period
 - Resulted in reading files from absolute paths



ClassLoaders Define Namespaces

- Classloader instances form a hierarchy
- Loading always starts with “primordial” classloader first
- Prevents class “spoofing”
- Protects unrelated classes with the same names in highly distributed systems





The Protection Circle

- Bytecode design and verification enables typesafety
- Typesafety allows classloader to enforce bytecode verification & namespaces
- Typesafety and correct namespaces ensure that security manager can work correctly
- BUT don't ignore your host OS






J2SE Security



Java Cryptography Architecture

- Service provider interfaces (SPI) standardized
 - Allows plug-in implementations
 - Permits choice of implementation
 - Supports new algorithms later
 - `jre/lib/security/java.security` specifies implementations and search order
 - `security.provider.1=sun.security.provider.Sun`






Using the Cryptography Architecture

```
import java.security.*;

// Get a digester object
MessageDigest md=MessageDigest.getInstance("MD5");
// feed data to the digester
byte [] buf=Message.getBytes()
md.update(buf);
// finish the computation
byte [] digest=md.digest();
```

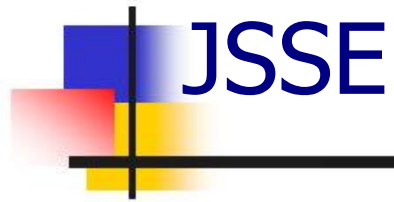




Using the Cryptography Architecture

- Operations:
 - getInstance
 - init
 - update
 - final





- Java Secure Sockets Extension
- Java API for SSL
- Provides security at session level
 - Confidentiality
 - Integrity
 - Authentication





Using JSSE

```
import java.io.*;  
import java.net.*;  
import javax.net.ssl.*;
```

```
ServerSocketFactory ssf =  
    SSLServerSocketFactory.getDefault();  
ServerSocket s = ssf.createServerSocket(port);  
Socket so = s.accept();
```





Using JSSE

```
import java.io.*;  
import java.net.*;  
import javax.net.ssl.*;
```

```
SocketFactory sf=  
    SSLSocketFactory.getDefault();
```

```
ServerSocket s =  
    sf.createSocket(host, port);
```

```
OutputStream o = s.getOutputStream();
```





- Java Authentication and Authorization Service
- Java platform security is based on code origin and signatures
- JAAS checks requesting user too
- Single sign-on framework
- Pluggable login modules
 - Per platform, per authentication type





JAAS Common Classes

- Common classes
 - Subject, Principal, Credential
- Authentication classes
 - LoginContext, LoginModule, CallbackHandler, Callback
- Authorization classes
 - Policy, AuthPermission, PrivateCredentialPermission





JAAS Permission Configuration

```
grant Codebase "friendlysite.com",  
    SignedBy "mymate", Principal  
    javax.security.auth.x500.X500Principal  
    "nigel" {  
        FilePermission "/cdrom/-", "read";  
    };
```





JAAS Login Configuration

```
Login2 {  
    sample.SampleLoginModule required;  
    com.sun.security.auth.module.NTLoginModule  
        sufficient;  
};
```

- required, sufficient, requisite, optional





JAAS Programming Model

```
import java.security.*;
import javax.security.auth.*;
LoginContext ctx = new LoginContext
    ("name", CallbackHandler);
ctx.login();
Subject s = ctx.getSubject();
Subject.doAs(sub, privelegedAction);
```



JAAS Programming Model

- `LoginContext.login` invokes login modules specified in configuration
 - `java.security.auth.login.config`
- Subject can define multiple `Principal` objects
- `doAs` invokes behavior with permissions for those principals





JAAS Configuration

- From JDK 1.4 JAAS permissions included in `jre/lib/security/java.policy`
- Enable the security manager





J2EE Security





Web Authentication

- At the Web tier, handled by web server or container
 - HTTP basic
 - Form-based
 - Client-certificate (SSL)
 - Digest





EJB Authentication

- At the EJB tier, handled by the EJB server/container
- At the client, handled by the EJB client container



Specifying Web Authentication

- Map usernames, passwords, and roles
 - (server specific)
- web.xml file indicates:
 - Type of authentication
 - Access-controlled URLs
 - SSL-required URLs





Specifying Web Authentication

```
<web-app>
  ...
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>MyCollection</web-resource-name>
      <url-pattern>/operation</url-pattern>
      <http-method>GET</http-method>
    </web-resource-collection>
    <auth-constraint>
      <role-name>admin</role-name>
    </auth-constraint>
    <user-data-constraint>
      <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
  </security-constraint>
  <login-config>
    <auth-method>BASIC</auth-method> <realm-name></realm-name>
  </login-config>
  ...
</web-app>
```



Form-Based Authentication

- Can be HTML or JSP page
- Contains HTML form like following

```
<FORM ACTION="j_security_check" METHOD="POST">  
  ...  
  <INPUT TYPE="TEXT" NAME="j_username">  
  ...  
  <INPUT TYPE="PASSWORD" NAME="j_password">  
  ...  
</FORM>
```





Programmatic Authorization

```
public interface javax.servlet.http.HttpServletRequest
{
    ...

    // Find out who is accessing your web resource
    public java.security.Principal getUserPrincipal();
    public String getRemoteUser();

    // Is the caller in a particular role?
    public boolean isUserInRole(String role);

    ...
}
```





Declare Method Permissions

```
<enterprise-beans>

  <assembly-descriptor>
    <method-permission>
      <role-name>manager</role-name>
      <method>
        <ejb-name>EmploySalary</ejb-name>
        <method-name>*</method-name>
      </method>
    </method-permission>
    <method-permission>
      <role-name>employee</role-name>
      <method><
        <ejb-name>EmploySalary</ejb-name>
        <method-name>doNothingMuch</method-
name>
      </method>
    </method-permission>
  </assembly-descriptor>
```

...





Programmatic Authorization at EJB-tier

- Programmer:
 - Code authorization logic inside the bean
 - Declare abstract security roles in D.D.
- Assemble/deploy
 - Map user identities to roles using vendor tools
 - Map abstract security roles to actual roles





Programmatic Authorization at EJB-tier

```
public interface javax.ejb.EJBContext{  
    ...  
    public java.security.Principal getCallerPrincipal();  
    public boolean isCallerInRole(String role);  
    ...  
}
```

- “Identity” based methods are deprecated





Restricting Instance Access

```
public double getSalary(String employeeId) {  
    java.security.Principal callerPrincipal =  
        ctx.getCallerPrincipal();  
  
    String callerId = callerPrincipal.getName();  
  
    if ( (ctx.isCallerInRole("manager")) ||  
        ((ctx.isCallerInRole("employee")) &&  
         (callerId == employeeId)) ) {  
        // return Salary information for the employee  
        getSalaryInformationSomehow(employId);  
    }  
    else {  
        throw new SecurityException("access denied");  
    }  
}
```





Mapping Abstract to Actual Roles

```
<enterprise-beans>
  <session>
    ...
    <!-- Abstract security role refers to real role -->
    <security-role-ref>
      <description> abstract "manager" role </description>
      <role-name>manager</role-name>
      <role-link>managerOfAcme</role-link>
    </security-role-ref>
  </session>
  <assembly-descriptor>
    <!-- Real security roles -->
    <security-role>
      <description>Manager in this corporation</description>
      <role-name>managerOfAcme</role-name>
    </security-role>
  </assembly-descriptor>
  ...
</enterprise-beans>
```





Delegation vs Propagation

- Delegation

```
<security-identity>  
  <run-as>  
    <role-name>  
      RoleNameToBePropagated  
    </role-name>  
  </run-as>  
</security-identity>
```

- Propagation

```
<security-identity>  
  <use-caller-identity/>  
</security-identity>
```





Web Services Issues





Web Services Issues

- Web Services bring new problems to security
- HTTPS isn't enough
 - Lack of granularity
 - Point-to-point only/security only “in transit”
 - No audit trail
 - Lack of signature





Web Services Cryptographic Styles

- Signatures:
 - Enveloped
 - Enveloping
 - Detached
 - Canonicalization & Transforms
- Encryption
 - Per-element including aggregate elements
 - Multiple encryption (caution!)
 - Key agreement

