Advanced Java Programming

Security

Chris Wong
chw@it.uts.edu.au
Based on notes by Wayne Brookes

Topics

• Security background
  – Confidentiality, Integrity, Availability
  – Authentication, Authorisation, Non-repudiation
  – Cryptography primer
  – Digital certificates
• Basic Java security
  – JCA, JCE, JSSE, JAAS
• Enterprise Java security
  – Declarative vs Programmatic

Introduction

• Security is about protecting your application from attacks
  – from those external to the application
  – from application users who try to exceed their authority
• Basic Principles
  – Confidentiality
  – Integrity
  – Availability

Confidentiality

• Ensuring that data is only seen by valid users
  – again, protection of "in-transit" data important
• Relies upon correct authentication/authorisation
  – valid users can't see data they're not permitted to
• Encryption of data commonly used
• Classes of encryption algorithm:
  – symmetric (DES, 3DES, RC5, etc)
  – asymmetric (RSA, Diffie-Hellman)

Confidentiality 2

Typical methods:

• Physical/Logical security
  – Eg: firewalls, DMZ, VPN, tunnelling
• Encryption of data
  – symmetric (DES, 3DES, RC5, AES, etc)
  – asymmetric (AES, RSA, Diffie-Hellman, IDEA)
• Authentication ie: identify userid
• Authorisation ie: Access Control
Integrity

- Ensuring that data is not tampered with
  - In transit?
    - Protect data over network. What about the Internet?
  - In storage?
    - Protect Database/File System/Operation System
  - Also needs correct authentication & authorisation!

- Typical methods:
  - Message Digest ("fingerprint")
    - Eg: Message Digest 5 (MD5)
  - Hash functions
    - SHA (Secure Hash Algorithm)
  - CRC Provide a complex "checksum" of the original data

Availability

- Ensure valid user can access data/system WHEN required

- Typical techniques:
  - Fail-safe systems eg: Hot Standby
  - Backup/redundancy eg: Clustering
  - Resilience to attacks eg: Denial of Service attacks, Viruses, Worms etc
  - Access controls – from valid locations only eg: intranet/extranet

Topics

- Security background
  - Confidentiality, Integrity, Availability
  - Authentication, Authorisation, Non-repudiation
  - Cryptography primer
  - Digital certificates
- Basic Java security
  - JCA, JCE, JSSE, JAAS
- Enterprise Java security
  - Declarative vs Programmatic

Authentication/ Identification

- Establishing who the user is
- Classes of techniques:
  - Password ("something you know")
  - Token ("something you possess")
  - Biometric ("something you are")
  - Certificate ("trusted 3rd party knows you")

Authorisation/ Access Control

- Establishing what the user is permitted to do
- Relies upon correct authentication of user
  - Access Control List (ACL) is a common technique

ACLs are common:
  - for a resource, provide a list of users who can access it, and what level of access each user has

Access control may be:
  - discretionary (ACLs based on users or groups)
  - role-based (ACLs based on user "roles")

Non-repudiation

- Ensuring that a user cannot deny having carried out some action
- Relies upon all other security services

- May be based on digital signatures
  - if you have a message digitally signed by a user, the user cannot deny having sent it ...
  - ... assuming the user was correctly authenticated, access control was correctly applied, the message was not tampered with, and no other user could have generated the digital signature

- Also relates to auditing
Topics

- Security background
  - Confidentiality, Integrity, Availability
  - Authentication, Authorisation, Non-repudiation
- Cryptography primer
  - Digital certificates
- Basic Java security
  - JCA, JCE, JSSE, JAAS
- Enterprise Java security
  - Declarative vs Programmatic

Cryptography primer

- Encryption takes as input some plaintext data and an encryption key, and generates ciphertext
- Decryption takes ciphertext and a decryption key, and generates plaintext

- Two main kinds of cryptography:
  - Symmetric (aka "private key" or "secret key" crypto):
    Same key for encryption and decryption – faster
  - Asymmetric (aka "public key" crypto):
    Different keys for encryption and decryption – slower

Symmetric Key Crypto

- Used for confidentiality

Asymmetric Key Crypto 1

- Used for confidentiality

Asymmetric Key Crypto 2

- Used for identity and non-repudiation
- aka "digital signature"

Topics

- Security background
  - Confidentiality, Integrity, Availability
  - Authentication, Authorisation, Non-repudiation
  - Cryptography primer
- Digital certificates
- Basic Java security
  - JCA, JCE, JSSE, JAAS
- Enterprise Java security
  - Declarative vs Programmatic
Digital Certificates

- A certificate can be used to identify a user:
  - a form of identification/authorisation
  - sometimes used as an alternative to passwords
  - based on asymmetric cryptography

- Concept:
  - A Certification Authority (CA), aka trusted third party
  - issues you with your own digital certificate
  - The CA is vouching that they believe you to be who you claim to be
  - Common CA companies are Verisign, Entrust
  - Typically require some form of real ID before they issue you with a valid certificate

Digital Certificate contents

- A digital certificate contains:
  - your public key
  - serial number
  - valid period
  - etc.

- A digital certificate is:
  - digitally signed by the CA
  - encrypted with your own public key

Digital Certificate – application

- When an application receives a certificate, it:
  - Verifies the digital signature of the CA by attempting to decrypt the certificate using the CA’s public key
  - The application will keep a local copy of the public keys of well-known CAs so it can do this.
  - Verifies the data contained in the certificate (expiry date, domain name/email address to whom the certificate was issued, etc)
  - Extracts your public key

  - The application can then send you messages encrypted with your public key
    - i.e. confidentiality function

SSL / TLS

- SSL = Secure Sockets Layer
- TLS = Transport Layer Security

- Basic concept:
  - server sends certificate to client, to establish its identity
  - client may optionally send certificate to server, to establish client’s identity
  - once authentication is complete, the client and server agree upon a “session key” – a symmetric key used only for the current session
  - key exchange is done using asymmetric encryption
  - data exchange is done using symmetric encryption

Topics

- Security background
  - Confidentiality, Integrity, Availability
  - Authentication, Authorisation, Non-repudiation
  - Cryptography primer
  - Digital certificates

- Basic Java security
  - JCA, JCE, JSSE, JAAS

- Enterprise Java security
  - Declarative vs Programmatic

Basic Java Security

- Java 1.0
  - sandbox model for applets
  - no flexibility

- Java 1.1
  - extended to allow “trusted” code to run outside sandbox
  - still fairly limited flexibility

- Java 1.2 (Java 2) core security
  - configurable policies based on codebase and signer
  - for a particular application, can limit resources accessible to that application
**Java security architecture**

- Java Authentication and Authorization Service (JAAS)
- Java Secure Socket Extension (JSSE)
- Java Cryptography Extension (JCE)

**Core Java 2 Security**

- When a class is loaded into the JVM, it goes to:
  - Class Loader
  - Byte code verifier
  - Security Manager / Access Controller

- The Security Manager / Access Controller rely on the definition of:
  - permissions
  - policies
  - protection domains
  - defined declaratively in a policy file

**Java Cryptography Architecture**

- JCA provides supporting classes for underlying cryptography mechanisms
- Concepts like:
  - Key (and PrivateKey, PublicKey)
  - Principal
  - Certificate
  - Policy
- Provides an architecture through which different crypto implementations can be plugged in
- Doesn't actually do the encryption itself

**Java Cryptography Extension**

- JCE provides some algorithms for actually performing encryption
- Defined separately to JCA because of U.S. (and other) export restrictions

**Java Secure Socket Extension**

- JSSE is the Java interface to the SSL/TLS protocol
- Allows all SSL/TLS features:
  - exchange of certificates for authentication
  - encrypted data exchange for confidentiality, integrity
- Allows SSL/TLS for more than just HTTP
  - IIOP for CORBA exchanges
  - WTLS for crypto in Wireless contexts

**Authentication & Authorisation**

- JAAS provides support for:
  - login authentication
  - ACL-based authorisation
- Uses "Pluggable Authentication Module" (PAM) architecture
  - like Solaris, Linux
  - allows different underlying authentication mechanisms, e.g. passwords, smart cards, etc.
JAAS classes

- java.security.Principal
  - a user identity
- javax.security.auth.Subject
  - represents an individual or organisation with multiple principals
- javax.security.auth.login.LoginContext
  - an API for Subjects (Principals) to log in or log out
- javax.security.auth.spi>LoginModule
  - interface that login service providers must follow
- javax.security.auth.login.Configuration
  - allows configuration of multiple LoginModules

JAAS client example

```java
try {
    // Create LoginContext
    loginContext = new LoginContext("SampleLoginModule",
                                    new MyCallbackHandler());

    // Attempt authentication
    loginContext.login();

    // Retrieve authenticated Subject and perform SampleAction as Subject.
    Subject subject = loginContext.getSubject();
    SampleAction sampleAction = new SampleAction();
    Subject.doAs(subject, sampleAction);
} catch (LoginException le) { System.out.println("Error logging in."); }
```

Topics

- Security background
  - Identification, Access Control, Integrity, Confidentiality, non-repudiation
  - Cryptography primer
  - Digital certificates
- Basic Java security
  - JCA, JCE, JSSE, JAAS
- Enterprise Java security
  - Declarative vs Programmatic

Enterprise application security

- Containers provide access to security services
  - web containers provide access to HTTP security services
  - EJB containers provide access to EJB security services
- Security may be implemented:
  - declaratively:
    - let the container do most of the work
  - programmatically:
    - you write code to do the work

Security Points

- security infrastructure (App Server "may" use JAAS to access)
- basic abp
- business logic (vendor a)
- business logic (vendor b)
- resource tier
- new InitialContext(props)
- JAAS
- rmi/iiop
- https
- servlets/jsp
- x.509 certificates/LDAP/etc.
- kerberos
- application server is responsible for 'adapting' the security infrastructure in the deployment environment to the J2EE application's needs
- Every App server does this differently

Security Infrastructure

- J2EE specs dictate little concerning actual security implementation.
- Many implementations possible
  - X.509 Certificates/LDAP/etc.
  - Kerberos
- Application server is responsible for 'adapting' the security infrastructure in the deployment environment to the J2EE application's needs
  - Every App server does this differently
Weblogic Security Infrastructure
- Weblogic provides extensions to J2EE 1.4
  - Users and Groups configured using console or Mbeans
  - Defines Service Provider Interface so various security implementations can be overridden
    - Authentication
    - Identity Assertion
    - Authorization
    - Auditing
    - Adjudication
    - Role Mapping
    - KeyStore
    - Credential Mapper

WL Security Infrastructure (Cont)
- Contains an embedded LDAP adapter
  - Adapters to most commercial LDAP servers also
- Can define advanced policies for access to resources
  - times of day, from where, etc.
- Adds concepts of hierarchy of roles ("groups") to standard security model

Enterprise applications & Java 2
- Java 2 security services provide a foundation for the container, but your applications may not necessarily need to use them
  - Container manages these features for you
- e.g. to use SSL, you configure the web container for SSL support, you don't use JSSE
- JAAS?
  - at the moment, designed for client-side authentication, for Java client applications (note: not web clients)
  - maybe in future used for authentication between web and EJB servers

Web tier Security Points
- Security Infrastructure (App Server *may* use JAAS to access)
- Web Tier (Servlets/JSP)
- EJBs
- Resource Tier
- JAAS

Declarative web security
- `<login-config>`
  - `<auth-method>BASIC</auth-method>`
  - `<realm-name>My Application</realm-name>`
- `<security-role>`
  - `<role-name>sales</role-name>`
- `<security-constraint>`
  - `<web-resource-collection>`
    - `<web-resource-name>HelloWorldServlet</web-resource-name>`
    - `<url-pattern>/HelloWorld</url-pattern>`
  - `<auth-constraint>`
    - `<role-name>sales</role-name>`
  - `<auth-constraint>`
Declarative web security (2)

- `<login-config>`
  - specifies what style of login to use. Some options:
    - BASIC – HTTP Basic (popup dialog) authentication
    - FORM – uses a user-written HTML form
    - CLIENT-CERT – requires client certificate to be presented

- `<security-role>`
  - just defines a role name that will be used later

- `<security-constraint>`
  - defines what to protect: `<web-resource-collection>`
  - defines authentication needed: `<auth-constraint>`
  - defines confidentiality needed: `<transport-guarantee>`

Programmatic web security

```java
import java.security.Principal;

public class HelloWorldServlet extends HttpServlet {
  public void doGet(HttpServletRequest req, HttpServletResponse res) throws ... {
    ... String username = null;
    Principal p = req.getUserPrincipal();
    if (p != null) {
      username = p.getName();
    }
    if (username == "MySalesUsers") {
      // Sales-specific code
    }
  }
}
```

Declarative vs. programmatic

- Declarative security is better
  - let the container manage the security
  - changing security doesn't require changing compiled code
  - however, it is coarse-grained:
    - granularity is on a per-file (servlet, JSP or HTML/GIF/JPG) level

- Programmatic security gives you more control
  - but may mean introducing business logic into presentation tier – bad!

EJB Security points

- Web Tier (Servlets/JSP)
- Business Logic (Vendor A)
- Resource Tier
- Business Logic (Vendor B)

EJB Security

- Authentication
  - validates the identity of the user
  - implemented through username/password logins, ID Cards, security certificates, etc.
- Authorization/Access Control
  - controls what a user can and cannot do within the system
- Secure Communications
  - ensuring the privacy of a communications
  - implemented through private communication (infrequently) channels or (more commonly) encryption
  - not covered by EJB Specification
Authentication

- EJB external clients (eg: RMI)
  - Specify principal and password properties when creating JNDI initial context
    - Use `jndi.properties` or when you use InitialContext
    - `set java.naming.security.authentication=none | simple | strong`
    - `set java.naming.security.principal=userid`
      `java.naming.security.credentials=password|key`
  - Or use JAAS with a client-login module

- Web services clients
  - can also use basic authentication
  - `Set javax.xml.rpc.Stub.USERNAME_PROPERTY, PASSWORD_PROPERTY properties before executing Stub`
  - Similar for Dynamic Invocation Clients (use `javax.xml.rpc.Call. properties`)

- Web Clients
  - Authenticated with FORM, Basic, or certificates

Authentication -2

- EJB Spec requires that every client access be associated with a security identity
  - user or role
  - `getCallerPrincipal` always returns a valid principal
- User logs into EJB System and authenticated through an implementation-specific method
- EJB Server passes security identity along with method invocation
- EJB objects or EJB homes check access

Authentication -3

- Declarative EJB security
  - `<security-role>`
    - just defines a role name that will be used later
  - `<method-permission>`
    - allows you to only permit certain users (roles) to invoke certain EJB methods
    - can use `<method-name> *</method-name>` to set the default security for all methods on an EJB, unless specifically overridden
  - Container may allow you to map `<role-name>` into specific users as defined on the server
    - WebLogic does, in `weblogic-ejb.xml`

Declarative EJB security (2)

- Caller principal:
  - Declare the security roles via annotation
    - `@DeclareRoles((HR_Manager, HR_Admin))`
  - You can set the role that your method/class runs under:
    - `@RunAs(HR_Manager)`
  - You can choose which method/class is authorised:
    - `@RolesAllowed(HR_Manager)`
    - `@PermitAll, @DenyAll`
EJB3 security example

```java
@RolesAllowed(HR_Manager)
@Stateless
public class PayrollBean implements Payroll {
    public void setSalary(int empId, double salary) {
        ...
    }
    @RolesAllowed({HR_Manager, HR_Admin})
    public int getSalary(int empId) {
        ...
    }
}
```

Programmatic EJB security

```xml
<enterprise-beans>
    <session>
        <ejb-name>MyTestEJB</ejb-name>
        <security-role-ref>
            <role-name>MySalesUsers</role-name>
            <role-link>sales</role-link>
        </security-role-ref>
    </session>
</enterprise-beans>
```

Programmatic EJB security (2)

```java
import java.security.Principal;

public class MyTestEJB implements SessionBean {
    public void updateSalesFigures(...) {
        String username = null;
        Principal p = sessionContext.getCallerPrincipal();
        if (p != null) {
            username = p.getName();
        }
        if (username == "MySalesUsers") {
            // Sales-specific code
        }
    }
}
```

Container security support

- How the container chooses to implement security is not specified by J2EE
- Different containers may support retrieving authentication information from different sources
  - filesystem (stored in a file)
  - LDAP directory
  - Database server
  - Operating system (NT or Unix login information)
- Different containers may support different SSL implementations
  - e.g. with different supported ciphers

Summary

- Java security based on traditional foundations
- JAAS, JSSE, JCE are more for client apps
  - but maybe relevance to enterprise apps in future, especially JAAS

- Two main aspects of Java enterprise security are:
  - authentication/authorisation (container-managed)
  - cryptography (SSL, certificates)

- Can implement declaratively or programatically
  - declaratively is easier/better when you can do it