Lecture 9: Authentication

Lecture topics

- Authentication and authorization for EJBs

Password-based authentication

- The most popular authentication technology
- Storing passwords is a problem
  - On the server machines
  - Could encrypt them, but would have to store the key somewhere
  - Solution: store only a hash of the password
    - An attacker wouldn’t be able to reconstruct the password from the hash
- Usability problem
  - good security -> long key -> long passwords
    - Many password hashing approaches use the password itself as a key
    - E.g., to get 128 bits of security, need 20-character passwords, with characters as random as possible
Lecture 9: Authentication

Password manipulation pitfalls

- Displaying password in cleartext on the screen as it is entered
- Unlimited login attempts
- Letting users use the same password over a long period of time
- Using a non-random seed
  - `crypt(pw, pw)` call in C
- Leaving passwords around in memory in plaintext
  - E.g. core dump attacks
  - E.g. swaps to the disk
  - Need “zero out” memory that was taken by the password
- Access rights to the password database

Password selection

- Problem: people pick things that are easy to remember
  - Prone to dictionary attacks
- Password systems commonly check that specific password quality standards are satisfied
- Any suggestions for good password selection criteria? Must be:
  - Easy to remember
  - Hard to guess
- `tbontbtitq`, anyone?
Dealing with the problem of having to remember multiple passwords

- Password safes
  - E.g. PasswordSafe from Counterpane
- Microsoft attempted to be more than that...
  - Microsoft Passport

One-time passwords

- People do wacky things
  - Write their password on a Post-It and stick it to their monitor
  - Send their password in an unencrypted email to their colleagues
  - ...
- One-time passwords improve security of a password by using a shared secret between the user and the application as a part of the password
  - E.g. SecurID
Lecture 9: Authentication

Reminder: role-based access control for EJBs

Java Authentication and Authorization Service (JAAS)

- Used for two purposes:
  - Authentication of users
    - Who is executing our code?
  - Authorization of users
    - Are they allowed to perform security-sensitive operations?

- Pluggable: different methods for authentication and authorization can be implemented and plugged in the framework
JAAS terminology

- **Subject** is a source of request for service
  - Could represent a person or a service
  - Modeled as a `javax.security.auth.Subject` object
- **Authentication** is the process of verifying the identity of a subject
  - The subject has to provide some evidence to demonstrate its identity
- **Principal** is the identity of a subject
  - A subject can have several principals
  - After authentication, a subject is populated with its principals
  - E.g. a person can have a name principal “Joe” and an SSN principal “666-66-6666”
- **Credentials** are security-related attributes of a subject (other than principals)
  - E.g. passwords, public key certificates, Kerberos tickets
  - May contain data that enables the subject to perform certain activities
    - E.g., cryptographic keys enable the subject to encrypt messages

JAAS authentication

- **User authentication** is done in two steps:
  - Instantiate a `LoginContext`
  - Call the `LoginContext`’s `login` method

Instantiating a `LoginContext`:

- Provide a config file entry name and a `CallbackHandler` to use for communication with the user, e.g.:
  ```java
  import javax.security.auth.login.*;
  import com.sun.security.auth.callback.TextCallbackHandler;
  LoginContext lc = new LoginContext("JaasSample", new TextCallbackHandler());
  ```

- The config file `jaas.conf` will contain:
  ```jaas
  JaasSample {
  com.sun.security.auth.module.Krb5LoginModule required;
  }
  com.sun.security.auth.module.Krb5LoginModule can be replaced with any class that implements interface `javax.security.auth.spi>LoginModule`
  ```
JAAS authentication, cont.

- **Using a CallbackHandler instance**
  - A LoginModule may need to communicate with the user
    - E.g. to ask for username and password
  - LoginModule uses a javax.security.auth.callback.CallbackHandler instance for this communication

Calling the LoginContext's login Method

- It actually carries out the authentication process: lc.login()
- Instantiates a new empty javax.security.auth.Subject object
  - Represents the user or service being authenticated
- Constructs the LoginModule and initializes it with the subject and CallbackHandler
- Calls methods in the LoginModule to perform the login (uses CallbackHandler to get the required info)
- If authentication is successful, the LoginModule populates the subject with a principal representing the user and user credentials
**JAAS-based security in JBoss**

- The EJB container intercepts calls to EJB methods and delegates the tasks of principal authentication and principal role mapping to two different security interfaces:
  - `org.jboss.security.EJBSecurityManager`
  - `org.jboss.security.RealmMapping`
- JBoss has a number of implementations of these interfaces in package `org.jboss.security.plugins.samples`
- The default security implementation consists of a JMX service bean `org.jboss.security.plugins.JaasSecurityManagerService` and a JAAS-based implementation of both interfaces `org.jboss.security.plugins.JaasSecurityManager`

**Configuring LoginModules to work with JaasSecurityManager**

- The `UsersRolesLoginModule` class is a simple properties file based implementation that uses two Java Properties (`users.properties` and `roles.properties`) to perform authentication and role mapping

  - `users.properties`:
    ```
    username1=password1
    username2=password2
    ...
    ```
  - `roles.properties`:
    ```
    username1=role1[,role2,...]
    username2=role1
    ...
    ```
Lecture 9: Authentication

The LoginModule configuration file

- This is where JAAS looks to find out what LoginModule implementation to use
- The default file for JBoss: 
  ${jboss_home}/conf/default/auth.conf
- Syntax and example:
  ```
  name {
    login_module_class_name (required|optional|...)
    [options]
  ;
  }

  example1 {
    org.jboss.security.auth.spi.UsersRolesLoginModule required
  ;
  }
  ```

Use of the configuration file

- The name of the security configuration has to be used in jboss.xml
  ```xml
  <jboss>
    <security-domain>java:/jaas/example1</security-domain>
    <enterprise-beans>
      ...
  </jboss>
  ```
Lecture 9: Authentication

**Example server code**

```java
public class StatefulSessionBean implements SessionBean {
    private SessionContext sessionContext;
    public void ejbCreate() throws CreateException {
        System.out.println("StatefulSessionBean.ejbCreate() called");
    }
    public void setSessionContext(SessionContext context) {
        sessionContext = context;
    }
    public String echo(String arg){
        System.out.println("StatefulSessionBean.echo, arg="+arg);
        Principal p = sessionContext.getCallerPrincipal();
        System.out.println("StatefulSessionBean.echo, callerPrincipal="+p);
        return arg;
    }
    public void noop() {
        System.out.println("StatefulSessionBean.noop");
        Principal p = sessionContext.getCallerPrincipal();
        System.out.println("StatefulSessionBean.noop, callerPrincipal="+p);
    }
}
```

**Example client code**

```java
public static void main(String args[]) throws Exception {
    if( args.length != 3 ) throw new IllegalArgumentException("Usage: username password example");
    String name = args[0];
    char[] password = args[1].toCharArray();
    String example = args[2];
    try {
        AppCallbackHandler handler = new AppCallbackHandler(name, password);
        LoginContext lc = new LoginContext("TestClient", handler);
        lc.login();
    } catch (LoginException le) {
        System.out.println("Login failed");
        System.exit(1);
    }
    try {
        InitialContext iniContext = new InitialContext();
        SessionHome home = (SessionHome) iniContext.lookup(example+"/StatelessSession");
        SessionBean home_create = home.create();
        System.out.println("Bean.echo('Hello') -> " + home_create.echo('Hello'));
        home.remove();
    } catch (Exception e) { System.exit(1) }
}
```
Example client code, cont.

```java
static class AppCallbackHandler implements CallbackHandler {
    private String username;
    private char[] password;
    public AppCallbackHandler(String username, char[] password) {
        this.username = username;
        this.password = password;
    }
    public void handle(Callback[] callbacks) throws 
        java.io.IOException, UnsupportedCallbackException {
        for (int i = 0; i < callbacks.length; i++) {
            if (callbacks[i] instanceof NameCallback) {
                NameCallback nc = (NameCallback)callbacks[i];
                nc.setName(username);
            } else if (callbacks[i] instanceof PasswordCallback) {
                PasswordCallback pc = (PasswordCallback)callbacks[i];
                pc.setPassword(password);
            } else {
                throw new UnsupportedCallbackException(callbacks[i],
                                "Unrecognized Callback");
            }
        }
    }
}
```

User authentication in JBoss: JBossSX framework

- `javax.ejb.EJBContext` defines the security context in the EJB container
  - `public boolean isCallerInRole(String roleName)` checks if the caller has the specified role
  - `public java.security.principal getCallerPrincipal()` returns the Principal object that identifies the caller
Lecture 9: Authentication

**Security proxies in JBoss**

- `org.jboss.security.SecurityProxy`
  - `void invoke(Method m, Object [] args, Object bean)`
  - ...
  - *An additional layer between clients and beans*
  - *Convenient for custom security checks*