Java Reflection and Keystores

FIT5003 Software Security week 5

Faculty of Information Technology, Monash University

1 Security with Java Reflection API

Java reflection API is used to examine or modify the behavior of attributes, methods, classes or interfaces at runtime. Through Java reflection, we can manipulate the private members of a class, which can be used to break the security of a system.

Here is the sample code used to invoke a method after authenticating the user with a security token. The method invocation would be successful, only if the user provides the correct access token.

```
import java.util.UUID;

public class APIConnection {

        private String token;

        public APIConnection() {

                // Assume the token is obtained via a secure service
                token =  UUID.randomUUID().toString();
        }

        // A public method that requires authentication first.
        public void doSomething(String tokenGuess) {
                if (token.equals(tokenGuess)) {
                        doSomethingSecret();
                } else {
                        System.out.println("Invalid Token");
                }
        }

        //This method is supposed to be protected
        private void doSomethingSecret() {
           System.out.println("You are authenticated!
                     Now let's do some sensitive stuff.");
        }
}
```

Question 1

Task 1: Overwriting the values of private attributes via Java reflection

Write a program named HackToken.java where you overwrite the value of the private attribute 'token' in APIConnection and then invoke the doSomething method with your own token value. Show how hava reflection can be used to examine and modify the values of private attributes.

Question 2

Task 2: Invoking private methods of a class via Java reflection

Write a program named HackMethod.java where you invoke the private method doSomethingSecret directly. Show how Java reflection can be used to invoke private methods of a Java class.

Question 3

Task 3: Enable Java Security Manager for protecting

Write a Java security policy for blocking the access to the private members of a Java class via Java Reflection API. Enable the Java Security Manager and show that your security policy blocks access to the private members of a class. Use your programs written for Task1 and Task2 for testing the security policy.

Info: Additional Reading:

Java Reflection API Tutorial with Example
https://www.guru99.com/java-reflection-api.html

Java Reflection And Security Management
https://rupalchatterjee.wordpress.com/2014/01/07/java-reflection-and-security-management/

2 Java Keytool and Java Keystores

2.1 Introduction

Java Keytool is a key and certificate management tool that is used to manipulate Java Keystores, and is included with Java. The Java KeyStore is a database that can contain keys. It is represented by the KeyStore (java.security.KeyStore) class. It is a container for authorization certificates or public key certificates, and is often used by Java-based applications for encryption, authentication, and serving over HTTPS. Its entries are protected by a keystore password. A keystore entry is identified by an alias, and it consists of keys and certificates that form a trust chain.

2.2 Keytool Installation

Keytool come with Java and is in the bin/ directory of the Java installation. To verify it is installed by checking the help output:

```
keytool -help
```

The output should look like below:

```
Key and Certificate Management Tool

Commands:

 -certreq            Generates a certificate request
 -changealias        Changes an entry's alias
 -delete             Deletes an entry
 -exportcert         Exports certificate
 -genkeypair         Generates a key pair
 -genseckey          Generates a secret key
```

```
 -gencert            Generates certificate from a certificate request
 -importcert         Imports a certificate or a certificate chain
 -importpass         Imports a password
 -importkeystore     Imports one or all entries from another keystore
 -keypasswd          Changes the key password of an entry
 -list               Lists entries in a keystore
 -printcert          Prints the content of a certificate
 -printcertreq       Prints the content of a certificate request
 -printcrl           Prints the content of a CRL file
 -storepasswd        Changes the store password of a keystore
 -showinfo           Displays security-related information

Use "keytool -?, -h, or --help" for this help message
Use "keytool -command_name --help" for usage of command_name.
Use the -conf <url> option to specify a pre-configured options file.
```

2.3 Java Keystore

A KeyStore can hold the following types of keys:

- Private keys
- Public keys + certificates
- Secret keys

Private and public keys are used in asymmetric encryption. A public key can have an associated certificate. A certificate is a document that verifies the identity of the person, organization or device claiming to own the public key. A certificate is typically digitally signed by the verifying party as proof. Secret keys are used in symmetric encryption. In many cases symmetric keys are negotiated when a secure connection is set up. Therefore you will more often be storing public and private keys in a KeyStore than secret keys.

2.3.1 Creating a KeyStore

You can create a Java KeyStore instance by calling its getInstance() method. Here is an example of creating a KeyStore instance:

```
KeyStore keyStore = KeyStore.getInstance(KeyStore.getDefaultType());
```

This example creates a KeyStore instance of Java's default type. It is also possible to create other types of KeyStore instance by passing a different parameter to the getInstance() method. For instance, here is an example that creates a PKCS12 type KeyStore:

```
KeyStore keyStore = KeyStore.getInstance("PKCS12");
```

2.3.2 Loading the KeyStore

Before a KeyStore instance can be used, it must be loaded. KeyStore instances are often written to disk or other kinds of storage for later use. That is why the KeyStore class assumes that you must read its data in before you can use it. However, it is possible to initialize an empty KeyStore instance with no data, as you will see later. Loading the KeyStore data from a file or other storage is done by calling the KeyStore load() method. The load() takes two parameters:

- An InputStream from which to load the KeyStore data.
- A char[] (char array) containing the KeyStore password.

Here is an example of loading a Java KeyStore:

```
char[] keyStorePassword = "123abc".toCharArray();
try(InputStream keyStoreData = new FileInputStream("keystore.ks")){
    keyStore.load(keyStoreData, keyStorePassword); }
```

This example loads the KeyStore file located in the keystore.ks file. If you don't want to load any data into the KeyStore, just pass null for the InputStream parameter. Here is how loading an empty KeyStore looks:

```
keyStore3.load(null, keyStorePassword);
```

You must always load the KeyStore instance, either with data or with null. Otherwise the KeyStore is uninitialized, and all calls to its methods will throw an exception

2.3.3 Getting Keys

You can get the keys of a Java KeyStore instance via its getEntry() method. A KeyStore entry is mapped to an alias which identifies the key, and is protected with a key password. Thus, to access a key you must pass the key alias and password to the getEntry() method. Here is an example of accessing a key entry in a KeyStore instance:

```
char[] keyPassword = "789xyz".toCharArray();
KeyStore.ProtectionParameter entryPassword =
    new KeyStore.PasswordProtection(keyPassword);
KeyStore.Entry keyEntry = keyStore3.getEntry("keyAlias", entryPassword);
```

If you know that the key entry you want to access is a private key, you can cast the KeyStore.Entry instance to a KeyStore.PrivateKeyEntry. Here is how that looks:

```
KeyStore.PrivateKeyEntry privateKeyEntry =
    (KeyStore.PrivateKeyEntry) keyStore3.getEntry("keyAlias", entryPassword);.
```

After casting to a KeyStore.PrivateKeyEntry you can access the private key, certificate and certificate chain via these methods:

- getPrivateKey()
- getCertificate()
- getCertificateChain()

2.3.4 Setting Keys

You can also set keys into a KeyStore instance. Here is an example of setting a secret key (symmetric key) into a KeyStore instance:

```
SecretKey secretKey = getSecretKey();
KeyStore.SecretKeyEntry secretKeyEntry =
        new KeyStore.SecretKeyEntry(secretKey);
keyStore3.setEntry("keyAlias2", secretKeyEntry, entryPassword);
```

2.3.5 Storing the KeyStore

Sometimes you may want to store a KeyStore to some storage (disk, database etc.) so you can load it again another time. You store a KeyStore by calling the store() method. Here is an example of storing a KeyStore

```
Tchar[] keyStorePassword = "123abc".toCharArray();
try (FileOutputStream keyStoreOutputStream =
        new FileOutputStream("data/keystore.ks")) {
    keyStore3.store(keyStoreOutputStream, keyStorePassword); }
```

Explore more Keystore Classes in java.security Package such as:

- Class KeyStore
- Class KeyStore.Builder
- Class KeyStore.CallbackHandlerProtection
- Class KeyStore.PasswordProtection
- Class KeyStore.PrivateKeyEntry

2.4 Sample Code

Example of RSA generation, sign, verify, encryption, decryption and keystores in Java

```
import javax.crypto.Cipher;
import java.io.InputStream;
import java.security.*;
import java.util.Base64;

import static java.nio.charset.StandardCharsets.UTF_8;

public class RsaExample {
  public static KeyPair generateKeyPair() throws Exception {
      KeyPairGenerator generator = KeyPairGenerator.getInstance("RSA");
      generator.initialize(2048, new SecureRandom());
      KeyPair pair = generator.generateKeyPair();

      return pair;
    }

  public static KeyPair getKeyPairFromKeyStore() throws Exception {
//Generated with:
//keytool -genkeypair -alias mykey -storepass s3cr3t -keypass s3cr3t
//    -keyalg RSA -keystore keystore.jks

    InputStream ins = RsaExample.class.getResourceAsStream("/keystore.jks");

    KeyStore keyStore = KeyStore.getInstance("JCEKS");
    keyStore.load(ins, "s3cr3t".toCharArray());   //Keystore password
    KeyStore.PasswordProtection keyPassword =       //Key password
              new KeyStore.PasswordProtection("s3cr3t".toCharArray());

    KeyStore.PrivateKeyEntry privateKeyEntry =
            (KeyStore.PrivateKeyEntry) keyStore.getEntry("mykey", keyPassword);

    java.security.cert.Certificate cert = keyStore.getCertificate("mykey");
    PublicKey publicKey = cert.getPublicKey();
    PrivateKey privateKey = privateKeyEntry.getPrivateKey();

    return new KeyPair(publicKey, privateKey);
  }

  public static String encrypt(String plainText, PublicKey publicKey)
        throws Exception {
    Cipher encryptCipher = Cipher.getInstance("RSA");
    encryptCipher.init(Cipher.ENCRYPT_MODE, publicKey);
```

```
    byte[] cipherText = encryptCipher.doFinal(plainText.getBytes(UTF_8));

    return Base64.getEncoder().encodeToString(cipherText);
  }

  public static String decrypt(String cipherText, PrivateKey privateKey)
    throws Exception {
    byte[] bytes = Base64.getDecoder().decode(cipherText);

    Cipher decriptCipher = Cipher.getInstance("RSA");
    decriptCipher.init(Cipher.DECRYPT_MODE, privateKey);

    return new String(decriptCipher.doFinal(bytes), UTF_8);
  }

  public static String sign(String plainText, PrivateKey privateKey)
        throws Exception {
    Signature privateSignature = Signature.getInstance("SHA256withRSA");
    privateSignature.initSign(privateKey);
    privateSignature.update(plainText.getBytes(UTF_8));

    byte[] signature = privateSignature.sign();

    return Base64.getEncoder().encodeToString(signature);
  }

  public static boolean verify(String plainText, String signature,
    PublicKey publicKey) throws Exception {
    Signature publicSignature = Signature.getInstance("SHA256withRSA");
    publicSignature.initVerify(publicKey);
    publicSignature.update(plainText.getBytes(UTF_8));

    byte[] signatureBytes = Base64.getDecoder().decode(signature);

    return publicSignature.verify(signatureBytes);
  }

  public static void main(String... argv) throws Exception {
        //First generate a public/private key pair
    KeyPair pair = generateKeyPair();
        //KeyPair pair = getKeyPairFromKeyStore();

        //Our secret message
    String message = "the answer to life the universe and everything";

        //Encrypt the message
    String cipherText = encrypt(message, pair.getPublic());

        //Now decrypt it
    String decipheredMessage = decrypt(cipherText, pair.getPrivate());

    System.out.println(decipheredMessage);

        //Let's sign our message
    String signature = sign("foobar", pair.getPrivate());

        //Let's check the signature
```

```
    boolean isCorrect = verify("foobar", signature, pair.getPublic());
    System.out.println("Signature correct: " + isCorrect);
  }
}
```

Question 4

TASKS:

1. Generate 2048 bits RSA public keys, which valid for 360 days.
2. Generate Self-signed SSL certificates..
3. Print certificate information and export into a file.
4. Generate and sign certificate signing request(CSR).

Question 5

TASKS:

1. Write Java Program to retrieve your key pair for your keystore, which you generated in 1.
2. Write Java Program to Listing the Aliases in your Key Store.
3. Write Java Program to encrypt the file using Symmetric Key from your Keystore