Java Client Security

COMP620

Java Security

- The Java platform was designed with a strong emphasis on security
- There are language features to improve security, such as strong typing and range checking on arrays
- There are many system features to support security of Java programs running as applets, applications and server programs

Bytecode Verifier

Upon loading a class, the JVM checks

- The class file is in the correct format
- Final classes and methods are not overridden
- No prohibited casting
- No stack overflows or underflows

Basic Security Architecture

- cryptography
- public key infrastructure
- authentication
- secure communication
- access control.

Platform Security Implementation

- Implementation independence
  - Applications can request security services from the Java platform
  - Security services are implemented in providers
- Implementation interoperability
- Algorithm extensibility
  - Platform includes a number of built-in providers
  - Additional services can be added

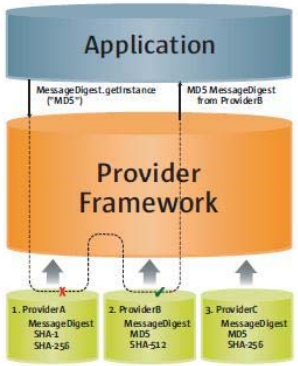
Providers

- The Java system provides a list of vendor providers for security services
- The standard JVM includes a number of default providers
- The **`java.security.Provider`** class encapsulates the list of providers
- The **`getInstance`** method can be used by a program to get a provider

Provider Search

Consider a request for any provider that has message digest MD5

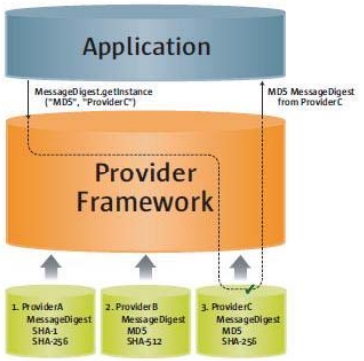
```
MessageDigest md = MessageDigest.getInstance("MD5");
```



Specified Provider

Consider a request for a specific provider that has message digest MD5

```
MessageDigest md =
      MessageDigest.getInstance("MD5", "ProviderC");
```



Secure Communications

- Java provides APIs for several different encrypted communication protocols including
  - Secure Socket Layer (SSL)
  - Transport Layer Security (TLS)
- The Java platform also defines basic Kerberos classes

Access Control

- The Java platform protects access to sensitive resources (for example, local files) or sensitive application code (for example, methods in a class)
- All access control decisions are mediated by a security manager
- A SecurityManager must be installed into the Java runtime in order to activate the access control checks

Security Manager

- By default, there is no security installed when you execute a Java application from the command line
- You can load a security manager programmatically or by using the **`-Djava.security.manager`** argument on the command line

Java Permissions

- Permissions are represented by **`java.security.Permission`** objects
- Permissions are based on
  - Where the code was loaded from
  - Who signed the code (if anyone)
  - Default permissions granted to the code

```
SecurityManager sm =
  System.getSecurityManager();
if (sm != null) {
  sm.checkPermission(perm); }
```

Java Policies

- The basic responsibility of a Policy object is to determine whether access to a protected resource is permitted
- Java encapsulates a security policy in the java.security.Policy class
- Policies are kept in a .policy file, either the default or a user created .policy file
- Policy files can be created using the GUI **policytool** utility

Access Control Example

- Consider a simple Java application that reads files from **`/mydir`** and from **`/otherdir`**
- When run with no security manager, the program can read both files
- When run with the security manager and default policies, neither file can be read
- Access to a specific directory can be allowed by creating a policy file and specifying it on the java command line

Java Command Line

- java -Djava.security.manager
  -Djava.security.policy=MyPolicy.policy
  -jar limited.jar

Java Security Files

- The default .policy and .security files are located in the Java Runtime Environment (JRE) directory in the subdirectory /lib/security