Lab Notes

CMPU4021 Distributed Systems

Java Security

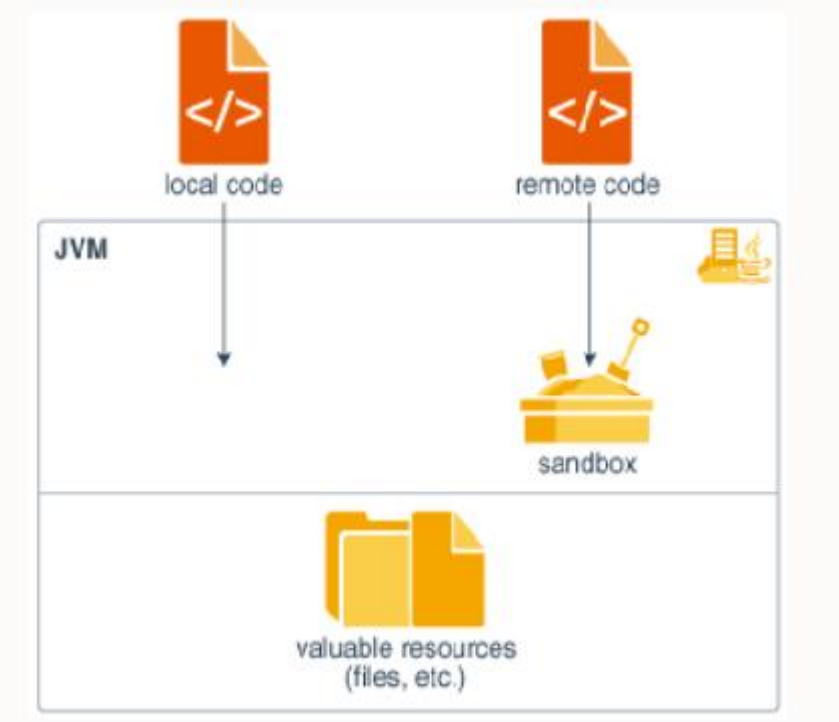
Java Security

- Includes a large set of APIs, tools, and implementations of commonly used security algorithms, mechanisms, and protocols.

- Strong emphasis on core security
  - Type-safe and provides automatic garbage collection
    - enhancing the robustness of application code
  - A secure class loading and verification mechanism
    - ensures that only legitimate Java code is executed

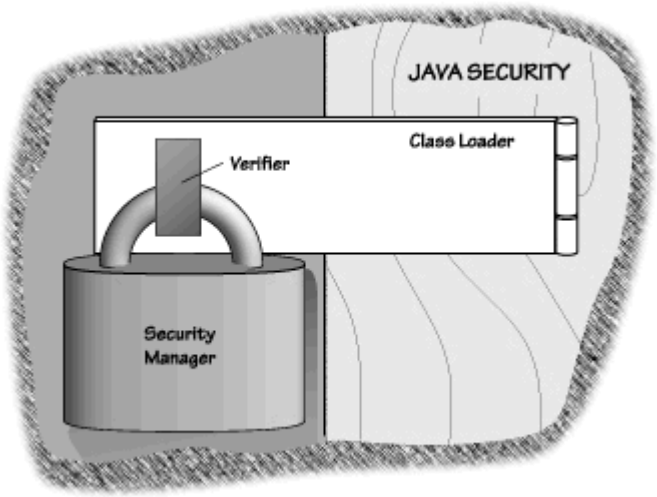
Original Java Platform Security Model

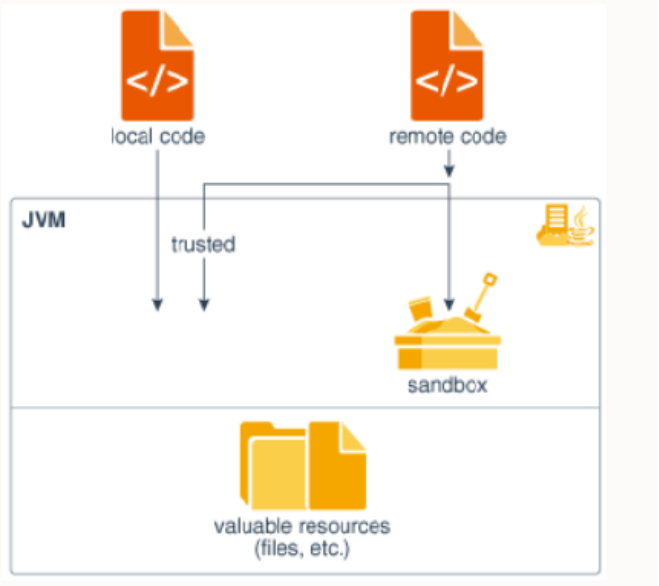
- Original Sandbox Model
  - Code is executed in the Java Virtual Machine (JVM).
    - JVM simulates execution of Java Byte Code.
  - Sandbox model allows code to run in a very restricted environment.
  - Local code has full access to valuable system resources.



Original Java Sandbox

- The default sandbox is made of three interrelated parts:
  - The *Verifier*
    - helps ensure type safety.
  - The *Class Loader*
    - loads and unloads classes dynamically from the Java runtime environment.
  - The *Security Manager*
    - acts as a security gatekeeper guarding potentially dangerous functionality.





Evolving the Sandbox Model: Java Standard Edition (SE) Platform Security Model

- Fine-grained access control
  - Previously, the application writer had to do substantial programming
    - e.g., by subclassing and customizing the `SecurityManager` and `ClassLoader` classes.
  - The new architecture makes this simpler and safer

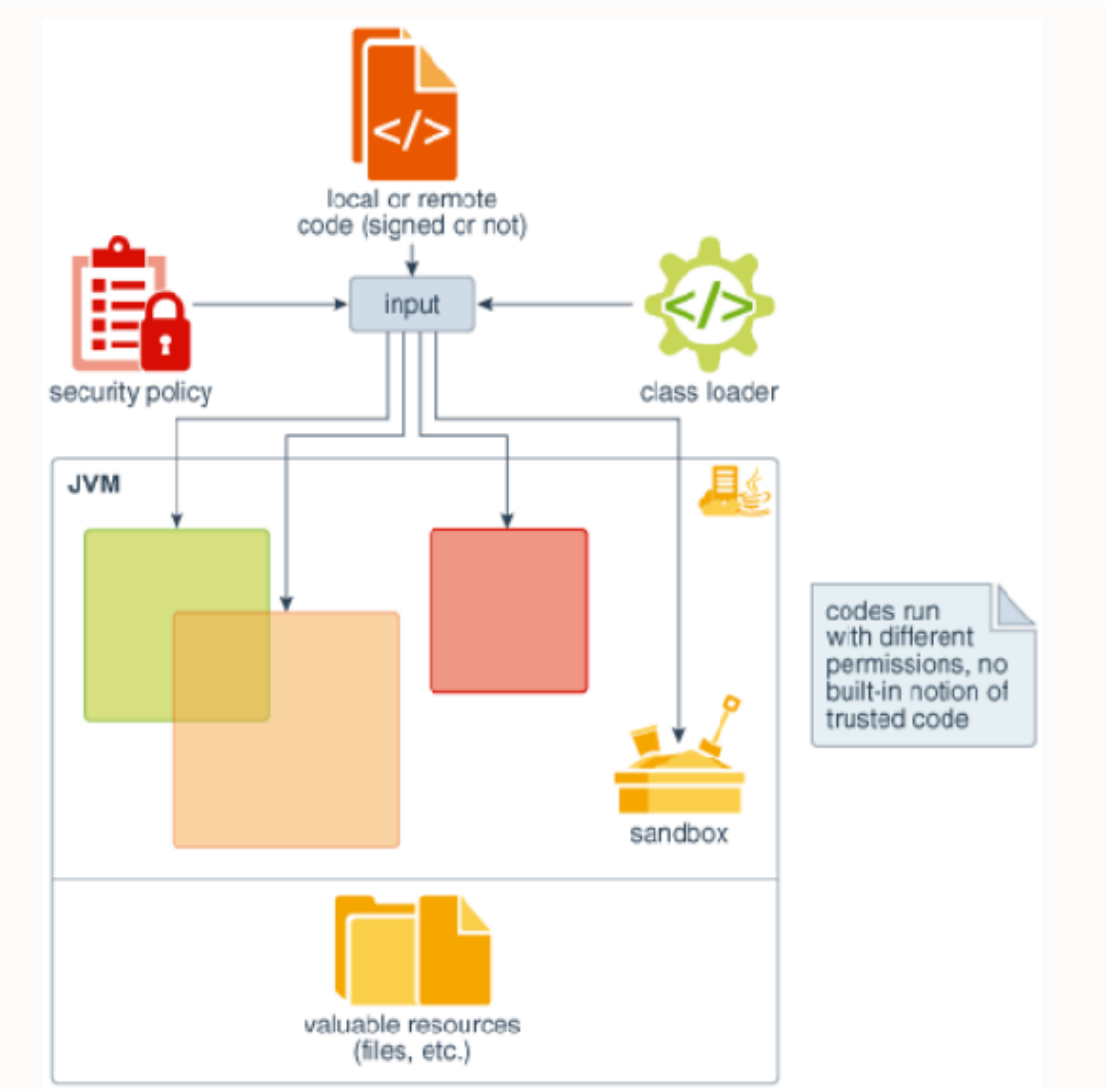
- Easily configurable security policy
  - Allows application builders and users to configure security policies without having to program

- Easily extensible access control structure
  - The new architecture allows typed permissions (each representing an access to a system resource) and automatic handling of all permissions (including yet-to-be-defined permissions) of the correct type.

Java SE Platform Security Model

- Extension of security checks to all Java programs
- There is no longer a built-in concept that all local code is trusted
- Local code (e.g., non-system code, application packages installed on the local file system)
  - is subjected to the same security control as remote

Java SE Security Architecture

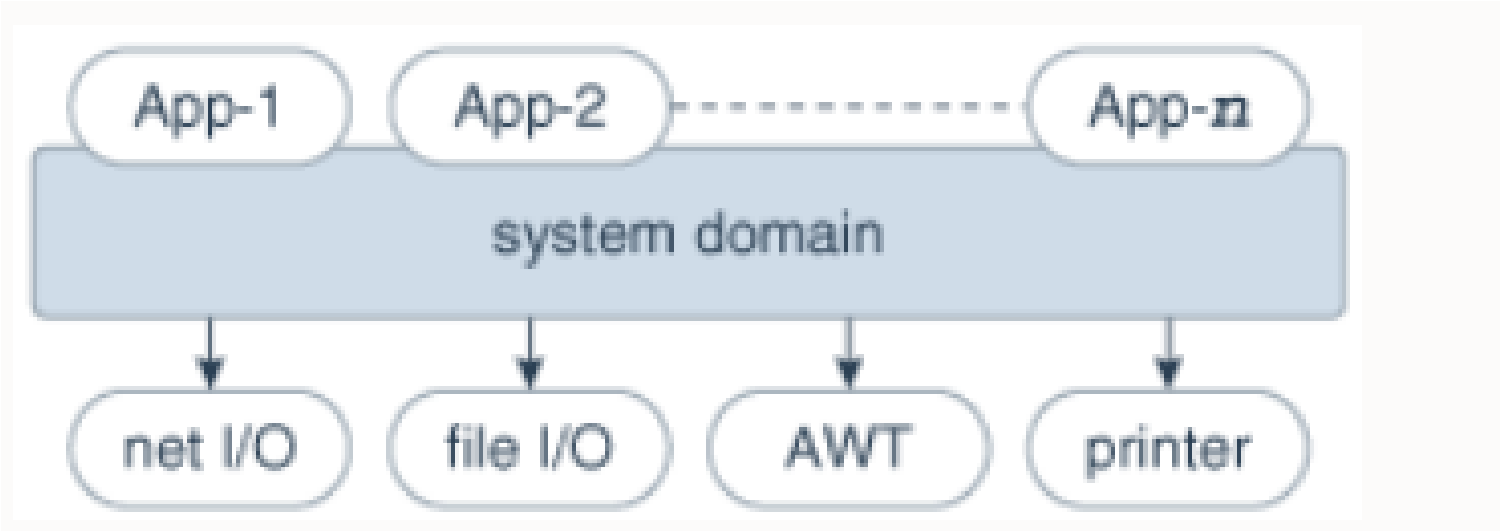


Java SE Platform Security: Protection Domains

- Protection Domains
  - Set of objects that are currently directly accessible by a principal.
  - **Principal**
    - An entity in the computer system to which permissions are granted.
  - Serves to group and to isolate between units of protection.
  - Protection domains are either system domains or application domains.

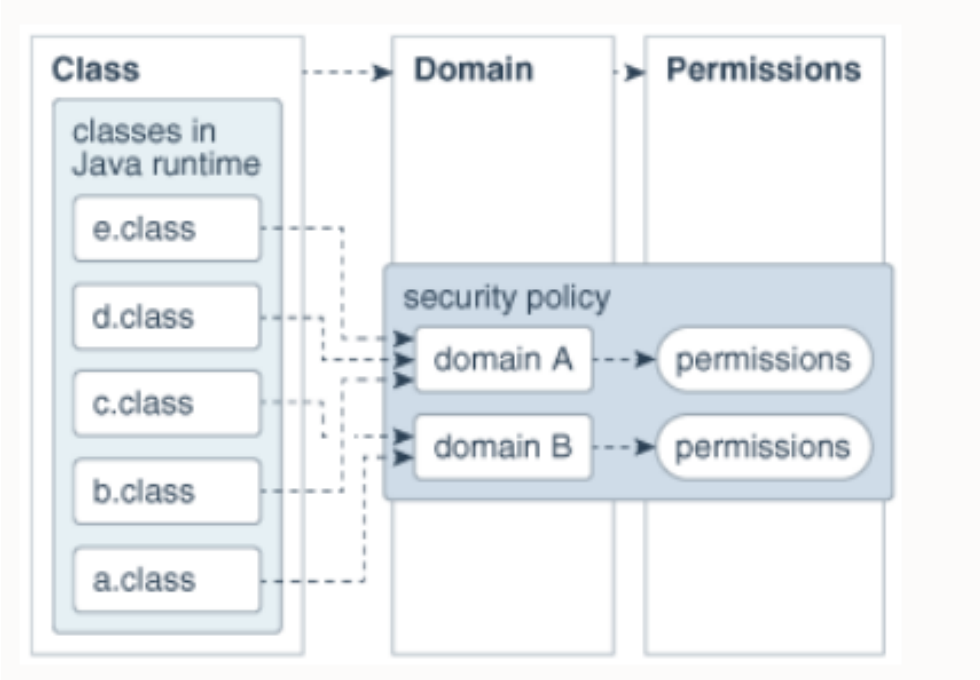
Java SE Platform Security: Protection Domains

- Protection domains generally fall into two distinct categories:
  - system domain
  - application domain.
- It is important that all protected external resources, such as the file system, the networking facility, and the screen and keyboard, be accessible only via system domains.



Mapping from Code to Domains and to Permissions

- A domain conceptually encloses a set of classes whose instances are granted the same set of permissions.
- Protection domains are determined by the policy currently in effect.
- The Java application environment maintains a mapping from code (classes and instances) to their protection domains and then to their permissions.



Protection Domains

- Java thread can completely occur within a single protection domain.
- Can also involve application domain and system domain.
- Examples:
  - Application prints out a message.
  - Needs to interact with system domain that is the access point to an output stream.
  - AWT system domain calls an applet's paint method to display it.
- Important:
  - A less "powerful" domain can NOT gain additional permissions as a result of calling or being called by a more powerful domain.

Protection Domains: Rules

- Normal rule:
  - The permission set of an execution thread is the intersection of the permissions of all protection domains traversed by the execution thread.
  - Exception: doPrivileged call
    - Enables a piece of trusted code to temporarily enable access to more resources than are available directly to the application that called it.
  - Example:
    - Application may not be allowed direct access to files that contain files, but the system utility displaying those fonts needs to obtain them on behalf of the user.

Protection Domains

- When access to a critical system resource (such as file I/O and network I/O) is requested:
  - the resource-handling code invokes a special AccessController class method
    - Evaluates the request
    - Decides if the request should be granted or denied.

Protection Domains

- Each domain needs to implement additional protection of internal resources.

- Example:
  - Banking application needs to maintain internal concepts of
    - checking accounts
    - deposits
    - withdrawals

Permissions and Security Policy

- The permission classes represent access to system resources.
- `java.security.Permission` class is an abstract class and is subclassed, as appropriate, to represent specific accesses.

- As an example of a permission, the following code can be used to produce a permission to read the file named "abc" in the /tmp directory:

```
perm = new java.io.FilePermission("/tmp/abc", "read");
```

`java.security.ProtectionDomain`

- `ProtectionDomain` class encapsulates the characteristics of a domain.

- Such a domain encloses a set of classes whose instances are granted a set of permissions when being executed on behalf of a given set of `Principals`.

Java SE Platform Security Why:

- Original Problem:
  - Users download programs that contain viruses and worms (even in commercial software).
  - Java machines executes downloaded codes, which make the problem worse.
  - Early work focuses on this issue:
    - Java programs are secure because they cannot install, run, or propagate viruses.

- Java SE
  The changes incorporated at this stage were to make the security solutions on the Java platform easy to use and more robust, thereby correcting limitations of earlier platform versions.

JAVA PUBLIC KEY INFRASTRUCTURE (PKI)

JAVA PKI

- The Java platform includes APIs and provider support for X.509 digital certificates and Certificate Revocation Lists (CRLs

- The classes related to PKI are located in the `java.security` and `java.security.cert` packages.

Java PKI Tools

Two built-in tools for working with keys, certificates, and key stores:

- `keytool`
  - Creates and manages key stores.
  - Use it to perform the following tasks:
    - Create public/private key pairs
    - Display, import, and export X.509 v1, v2, and v3 certificates stored as files
    - Create X.509 certificates
    - Issue certificate (PKCS#10) requests to be sent to CAs
    - Create certificates based on certificate requests
    - Import certificate replies (obtained from the CAs sent certificate requests)
    - Designate public key certificates as trusted
    - Accept a password and store it securely as a secret key

- `jarsigner`
  - Signs JAR files and verifies signatures on signed JAR files.

Secure File Exchange
Code and Document Security

- Digital documents are easy to generate, copy and alter.

- If you electronically send someone an important document (or documents), or application to run, the recipient needs a way to verify that the document or code came from you and was not modified in transit (for example, by a malicious user intercepting it).

- Digital signatures, certificates, and `keystores` all help ensure the security of the files you send.

Secure Code & File Exchange (1)

- Use `keytool` to generate or import appropriate keys and certificates into your key store (if they are not there already).
- Use the `jar` tool to package the code in a JAR file.
- You "sign" the document or code using one of your private keys.That is, you generate a digital signature for the document or code, using the `jarsigner` tool or (or the `jdk.security.jarsigner` API).
- You send to the other person, the "receiver," the document or code and the signature.
- You also supply the receiver with the public key corresponding to the private key used to generate the signature, if the receiver doesn't already have it.

Secure Code & File Exchange (2)

- The receiver uses the public key to verify the authenticity of the signature and the integrity of the document/code.
- A receiver needs to ensure that the public key itself is authentic before reliably using it to check the signature's authenticity.
- It is more typical to supply a certificate containing the public key rather than just the public key itself.

Secure File Exchange: Digital Certificates

A certificate contains:

1. A public key.

2. The "distinguished-name" information of the entity (i.e., the certificate subject or owner).

   Includes attributes such as: the entity's name, organisational unit, organization, city or locality, state or province, and country code.

3. A digital signature.

   A certificate is signed by one entity, the issuer, to vouch for the fact that the enclosed public key is the actual public key of another entity, the owner.

4.The distinguished-name information for the signer (issuer).

Secure File Exchange: Digital Certificates … cont'd

Validating Certificates

1. A recipient check if a certificate is valid by verifying its digital signature, using the issuer's (signer's) public key.

2. That key can be stored within another certificate whose signature can also be verified by using the public key of that next certificate's issuer, and that key may also be stored in yet another certificate, and so on.

   You can stop checking when you reach a public key that you already trust and use it to verify the signature on the corresponding certificate.

3. If a recipient cannot establish a trust chain, then he/she can calculate the certificate fingerprint(s), using the `keytool -import or -printcert` command.

A fingerprint is a relatively short number (hash value of the certificate information) that uniquely and reliably identifies the certificate. If the fingerprints are the same, the certificates are the same.

Signing Code and Granting Permissions –Sending side

Susan:

1. Creates an application

2. Create a JAR File containing the class file, using the `jar` tool.

3. Generate keys (if they don't already exist), using the `keytool -genkey` command.

4. Sign the JAR file, using the `jarsigner` tool and the private key.

5. Export the public key certificate, using the `keytool -export` command. Then supply the signed JAR file and the certificate to the receiver Ray.

Signing Code and Granting Permissions - Receiving side

Ray:

1. Tries to run the jar code received from Susan and gets:

```
Exception in thread "main"
java.security.AccessControlException:
```

2. Import the certificate as a trusted certificate, using the `keytool -import` command, and give it the alias susan.

3. Set up a **policy** file to grant the required permission to permit classes signed by susan to read the specified file.

4. See the policy file effects, that is, see how the application can now read the file.

References

- **https://docs.oracle.com/en/java/javase/19/security/java-security-overview1.html**

- https://docs.oracle.com/en/java/javase/19/security/java-se-platform-security-architecture.html#GUID-D6C53B30-01F9-49F1-9F61-35815558422B