INSA
CENTRE VAL DE LOIRE

POO Avancée : Java

Overview of Java Security

Salwa SOUAF

Outline

Introduction

La sécurité en Java est un vaste sujet qui englobe de nombreux domaines. Certains d'entre eux font partie du langage lui-même, comme les modificateurs d'accès et les chargeurs de classes. En outre, d'autres sont disponibles sous forme de services, notamment le cryptage de données, la communication sécurisée, l'authentification et l'autorisation, entre autres.

Objectif : Voir les bases de la sécurité sur la plate-forme Java. Overview de ce qui est disponible pour écrire des applications sécurisées.

Outline

Caractéristiques du Language

La sécurité en Java commence au niveau des fonctionnalités du langage. Ce qui permet d'écrire du code sécurisé et de bénéficier de nombreuses fonctionnalités de sécurité implicites:

- **Saisie de données statiques:** Java est un langage typé de manière statique, ce qui réduit les possibilités de détection des erreurs liées au type à l'exécution.
- **Modificateurs d'accès:** Java nous permet d'utiliser différents modificateurs d'accès, *public* et *private*, pour contrôler l'accès aux champs, méthodes et classes.
- **Gestion automatique de la mémoire:** Java dispose d'une gestion de la mémoire basée sur le *garbage-collection*, ce qui évite aux développeurs de la gérer manuellement.
- **Vérification du bytecode:** Java est un langage compilé, ce qui signifie qu'il convertit le code en un bytecode indépendant de la plate-forme et que le moteur d'exécution vérifie chaque bytecode chargé pour son exécution.

Ce n'est pas une liste complète des fonctionnalités de sécurité fournies par Java.

Outline

Architecture de sécurité en Java

Les principes fondamentaux de la sécurité en Java reposent sur des implémentations de fournisseurs interopérables et extensibles. Une implémentation particulière de fournisseur peut implémenter tout ou partie des services de sécurité.
Par exemple, certains des services typiques qu'un fournisseur peut implémenter sont:

- Algorithmes cryptographiques (tels que DSA, RSA ou SHA-256)
- Fonctions de génération, de conversion et de gestion de clés (telles que des algorithm-specific keys)

Java est livré avec de nombreux fournisseurs intégrés. En outre, il est possible pour une application de configurer plusieurs fournisseurs avec un ordre de préférence.

Architecture de sécurité en Java

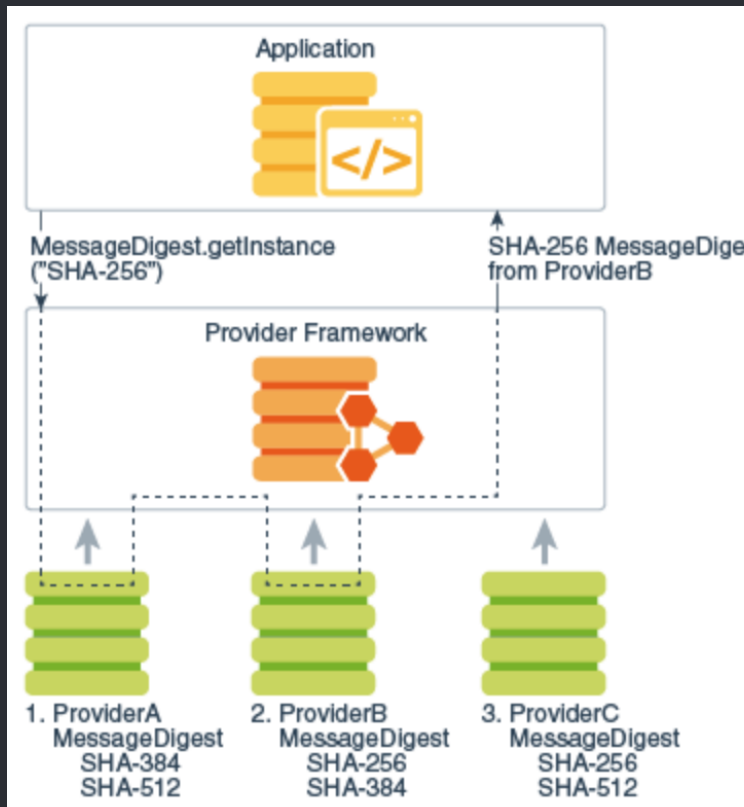

Figure 1: Provider searching

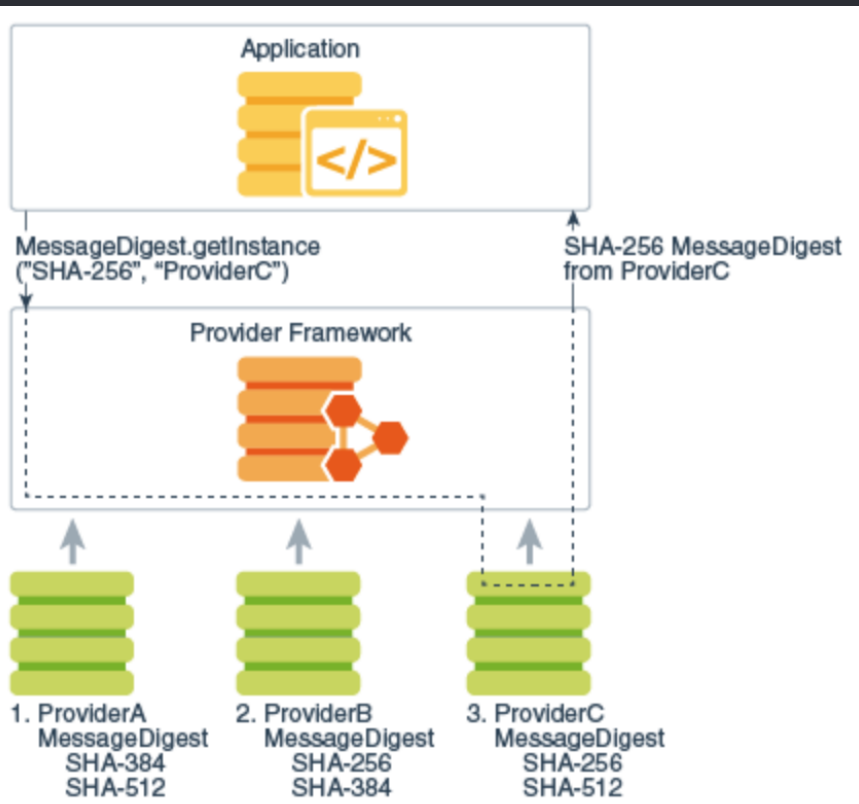

Figure 2: Specific provider requested

Par conséquent, la structure de fournisseur en Java recherche une implémentation spécifique d'un service chez tous les fournisseurs dans l'ordre de préférence défini pour eux.

De plus, il est toujours possible d'implémenter des fournisseurs personnalisés avec des fonctions de sécurité enfichables dans cette architecture.

Outline

Cryptographie

La cryptographie est la pierre angulaire des fonctions de sécurité en général et en Java. Il s'agit d'outils et de techniques de communication sécurisée en présence d'adversaires.

L'architecture cryptographique Java (JCA) fournit un cadre permettant d'accéder aux fonctionnalités cryptographiques en Java et de les implémenter, notamment:

- Digital signatures
- Message digests
- Symmetric and asymmetric ciphers
- Message authentication codes
- Key generators and key factories

Java utilise des implémentations basées sur fournisseur pour les fonctions cryptographiques.

Java inclut des fournisseurs intégrés pour les algorithmes cryptographiques couramment utilisés tels que RSA, DSA et AES, pour n'en nommer que quelques-uns. Nous pouvons utiliser ces algorithmes pour renforcer la sécurité des données en repos, en cours d'utilisation ou en mouvement.

Un cas d'utilisation très courant dans les applications est le stockage de mots de passe d'utilisateurs. Il est évident que le stockage de mots de passe en texte brut compromet la sécurité.
Une solution consiste donc à brouiller les mots de passe de manière à ce que le processus soit répétable, mais à sens unique. Ce processus est connu sous le nom de fonction de hachage cryptographique, et SHA1 est l'un de ces algorithmes populaires.
En Java:

```
1  MessageDigest md = MessageDigest.getInstance("SHA-1");
2  byte[] hashedPassword = md.digest("password".getBytes());
```

MessageDigest est un service cryptographique. La méthode *getInstance()* est utilisé pour demander ce service à n'importe quel fournisseur de sécurité disponible.

Outline

PKI : Public Key Infrustructure

Definition

L'infrastructure de clé publique (PKI) désigne la configuration qui permet l'échange sécurisé d'informations sur le réseau à l'aide du cryptage à clé publique. Cette configuration repose sur la confiance établie entre les parties impliquées dans la communication. Cette confiance est basée sur des certificats numériques émis par une autorité neutre et fiable connue sous le nom d'autorité de certification (CA).

PKI : Public Key Infrustructure

Support PKI en Java

La plate-forme Java a des API pour faciliter la création, le stockage et la validation des certificats numériques:

- KeyStore: Java fournit la classe KeyStore pour le stockage persistant de clés cryptographiques et de certificats de confiance. KeyStore peut ici représenter les fichiers de magasin de clés et de magasins de confiance. Ces fichiers ont un contenu similaire mais leur utilisation varie.
- CertStore: représente un référentiel public de certificats et de listes de révocation potentiellement non fiables. On a besoin d'extraire des certificats et des listes de révocation pour la création de chemins de certificat, entre autres utilisations.
- Java a un trust-store intégré appelé «cacerts» qui contient des certificats pour des autorités de certification bien connues.

Outils Java pour PKI

Java propose des outils très utiles pour faciliter la communication de confiance:

- «keytool» pour créer et gérer le magasin de clés et le magasin de confiance.
- «jarsigner» pour signer et vérifier les fichiers JAR.

PKI : Public Key Infrustructure

Travailler avec des certificats en Java

Une connexion SSL authentifiée mutuellement nécessite deux choses:
Certificat actuel - un certificat valide a présenté à une autre partie dans la communication. Pour cela, il faut charger le fichier de clés contenant nos clés publiques:

```
1  KeyStore keyStore = KeyStore.getInstance(KeyStore.getDefaultType());
2  char[] keyStorePassword = "changeit".toCharArray();
3  try(InputStream keyStoreData = new FileInputStream("keystore.jks")){
4      keyStore.load(keyStoreData, keyStorePassword);
5  }
```

Vérifier le certificat - vérifier le certificat présenté par une autre partie dans la communication. Pour cela, il faut charger le magasin de confiance contenant préalablement des certificats de confiance d'autres parties:

```
1  KeyStore trustStore = KeyStore.getInstance(KeyStore.getDefaultType());
2  // Load the trust-store from filesystem as before
```

Nous avons rarement à le faire par programme et nous passons normalement les paramètres système à Java lors de l'exécution:

```
1  -Djavax.net.ssl.trustStore=truststore.jks
2  -Djavax.net.ssl.keyStore=keystore.jks
```

Outline

Authentification

L'authentification est le processus de vérification de l'identité présentée par un utilisateur ou une machine sur la base de données supplémentaires telles qu'un mot de passe, un jeton ou une variété d'autres informations d'identification disponibles aujourd'hui.
Les API Java utilisent des modules de connexion enfichables pour fournir différents mécanismes d'authentification aux applications, souvent multiples. **LoginContext** fournit cette abstraction, qui à son tour fait référence à la configuration et charge un **LoginModule** approprié.
Alors que plusieurs fournisseurs mettent à disposition leurs modules de connexion, Java en propose quelques-uns par défaut:

- Krb5LoginModule, pour l'authentification basée sur Kerberos
- JndiLoginModule, pour une authentification basée sur un nom d'utilisateur et un mot de passe, sauvegardée par un magasin LDAP
- KeyStoreLoginModule, pour l'authentification par clé cryptographique

Authentification

Exemple : Login

L'un des mécanismes d'authentification les plus courants est le nom d'utilisateur et le mot de passe. Voyons comment utiliser JndiLoginModule.
Ce module est chargé de récupérer le nom d'utilisateur et le mot de passe d'un utilisateur et de les vérifier par rapport à un service d'annuaire configuré dans JNDI:

```
1 LoginContext loginContext = new LoginContext("Sample", new SampleCallbackHandler());
2 loginContext.login();
```

Utilisant une instance de **LoginContext** pour effectuer la connexion. **LoginContext** prend le nom d'une entrée dans la configuration de connexion - dans ce cas, il s'agit de «Sample». En outre, il faut fournir une instance de **CallbackHandler**, à l'aide du module **LoginModule** qui interagit avec l'utilisateur pour obtenir des informations telles que le nom d'utilisateur et le mot de passe.
Configuration de connexion:

```
1 Sample {
2   com.sun.security.auth.module.JndiLoginModule required;
3 };
```

Assez simple, cela suggère que **JndiLoginModule** est utilisé en tant que **LoginModule** obligatoire.

Outline

Communication sécurisée

La communication sur le réseau est vulnérable à de nombreux vecteurs d'attaque. Par exemple, quelqu'un peut accéder au réseau et lire nos paquets de données lors de leur transfert. Au fil des ans, l'industrie a mis en place de nombreux protocoles pour sécuriser cette communication.
Java fournit des API pour sécuriser les communications réseau avec chiffrement, intégrité des messages et authentification client et serveur:

- SSL / TLS: SSL et son successeur, TLS, assurent la sécurité des communications réseau non fiables via le cryptage des données et une infrastructure à clé publique. Java fournit une prise en charge de SSL / TLS via SSLSocket défini dans le package "java.security.ssl".
- SASL: SASL (Simple Authentication and Security Layer) est une norme d'authentification entre client et serveur. Java prend en charge SASL dans le cadre du package «java.security.sasl».
- GGS-API / Kerberos: L'API générique du service de sécurité (GSS-API) offre un accès uniforme aux services de sécurité via divers mécanismes de sécurité tels que Kerberos v5. Java prend en charge GSS-API dans le cadre du package "java.security.jgss".

Communication sécurisée

Ouvrir une connexion sécurisée avec d'autres parties en Java à l'aide de SSLSocket:

```
1  SocketFactory factory = SSLSocketFactory.getDefault();
2  try (Socket connection = factory.createSocket(host, port)) {
3      BufferedReader input = new BufferedReader(
4          new InputStreamReader(connection.getInputStream()));
5      return input.readLine();
6  }
```

I**SSLSocketFactory** est utilisé pour créer **SSLSocket**. Dans ce cadre, on peut définir des paramètres facultatifs tels que les suites de chiffrement et le protocole à utiliser. Pour que cela fonctionne correctement, vous devez avoir créé et défini le magasin de clés et notre magasin de confiance.

Outline

Contrôle d'accès

Le contrôle d'accès fait référence à la protection des ressources sensibles telles qu'un système de fichiers ou une base de code contre les accès non garantis. Ceci est généralement réalisé en limitant l'accès à de telles ressources.
Le contrôle d'accès en Java se fait en utilisant les classes **Policy** et **Permission** via la classe **SecurityManager**. **SecurityManager** fait partie du package "java.lang" et est responsable de l'application des contrôles d'accès en Java.
Lorsque le chargeur de classes charge une classe dans le moteur d'exécution, il accorde automatiquement des autorisations par défaut à la classe encapsulée dans l'objet Permission. Au-delà de ces autorisations par défaut, on peut accorder plus de ressources à une classe par le biais de stratégies de sécurité. Ceux-ci sont représentés par la classe **Policy**.
Au cours de la séquence d'exécution du code, si le moteur d'exécution rencontre une demande pour une ressource protégée, **SecurityManager** vérifie l'autorisation demandée par rapport à la politique installée via la pile d'appels. Par conséquent, il accorde l'autorisation ou lève une **SecurityException**.

Java a une implémentation par défaut de Policy qui lit les données d'autorisation à partir du fichier de propriétés. Toutefois, les entrées de stratégie dans ces fichiers de stratégie doivent être dans un format spécifique.
Java est livré avec «policytool», un utilitaire graphique permettant de composer des fichiers de règles.

Contrôle d'accès

Exemple : Access Control

Limitant l'accès à une ressource telle qu'un fichier en Java:

```
SecurityManager securityManager = System.getSecurityManager();
if (securityManager != null) {
    securityManager.checkPermission(
      new FilePermission("/var/logs", "read"));
}
```

On utilise SecurityManager pour valider la demande de lecture pour un fichier, encapsulée dans FilePermission.
Cependant, SecurityManager délègue cette demande à AccessController.
AccessController utilise en interne la stratégie installée pour prendre une décision.
Un exemple du fichier de politique:

```
grant {
  permission
    java.security.FilePermission
      <<ALL FILES>>, "read";
};
```

On accorde essentiellement la permission de lecture à tous les fichiers pour tout le monde. Cependant, on peut fournir un contrôle beaucoup plus fin par le biais de politiques de sécurité.

Exemple : Access Control

Il est à noter qu'un SecurityManager peut ne pas être installé par défaut en Java. On peut s'en assurer en démarrant toujours Java avec le paramètre:

```
1  -Djava.security.manager -Djava.security.policy=/path/to/sample.policy
```

Outline

Signature XML

Les signatures XML sont utiles pour sécuriser les données et assurer leur intégrité. Le W3C fournit des recommandations pour la gouvernance de la signature XML. Nous pouvons utiliser la signature XML pour sécuriser des données de tout type, telles que des données binaires.
L'API Java prend en charge la génération et la validation des signatures XML conformément aux instructions recommandées. L'API de signature numérique Java XML est encapsulée dans le package "java.xml.crypto".
La signature elle-même est juste un document XML. Les signatures XML peuvent être de trois types:

- Détaché (Detached): ce type de signature recouvre les données externes à l'élément Signature
- Enveloppement (Enveloping): Ce type de signature recouvre les données internes à l'élément Signature.
- Enveloppé (Enveloped): Ce type de signature recouvre les données contenant l'élément Signature lui-même.

Java prend en charge la création et la vérification de tous les types de signatures XML ci-dessus.

Signature XML

Exemple : Création d'une signature XML

Générant une signature XML pour nos données. Par exemple, on peut être sur le point d'envoyer un document XML sur le réseau. On veut donc que le destinataire puisse vérifier son intégrité.

```
1  XMLSignatureFactory xmlSignatureFactory = XMLSignatureFactory.getInstance("DOM");
2  DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory.newInstance();
3  documentBuilderFactory.setNamespaceAware(true);
4
5  Document document = documentBuilderFactory
6    .newDocumentBuilder().parse(new FileInputStream("data.xml"));
7
8  DOMSignContext domSignContext = new DOMSignContext(
9    keyEntry.getPrivateKey(), document.getDocumentElement());
10
11 XMLSignature xmlSignature = xmlSignatureFactory.newXMLSignature(signedInfo, keyInfo);
12 xmlSignature.sign(domSignContext);
```

Signature XML

Exemple : Création d'une signature XML

Pour clarifier, générant une signature XML pour les données présentes dans le fichier «data.xml». En attendant, il y a quelques choses à noter à propos de ce morceau de code:

- Tout d'abord, XMLSignatureFactory est la classe fabrique pour générer des signatures XML.
- XMLSigntaure nécessite un objet SignedInfo sur lequel il calcule la signature
- XMLSigntaure a également besoin de KeyInfo, qui encapsule la clé de signature et le certificat.
- Enfin, XMLSignature signe le document en utilisant la clé privée encapsulée en tant que DOMSignContext.

Par conséquent, le document XML contient désormais l'élément Signature, qui peut être utilisé pour vérifier son intégrité.

Outline

Java Frameworks

Java fournit beaucoup des fonctionnalités nécessaires pour écrire des applications sécurisées. Cependant, ces derniers sont parfois de bas niveau et ne sont pas directement applicables, par exemple, au mécanisme de sécurité standard sur le Web.
On ne veut généralement pas lire l'intégralité du RFC OAuth et l'implémenter soit-mêmes. On a souvent besoin de moyens plus rapides et de plus haut niveau pour atteindre la sécurité. C'est là que les cadres d'application entrent en scène - ils aident à atteindre un bon niveau de sécurité avec beaucoup moins de code passe-partout.
Sur Java, cela signifie généralement Spring Security. Le framework fait partie de l'écosystème Spring, mais il peut être utilisé en dehors de l'application Spring pure. En termes simples, il est utile d'obtenir l'authentification, l'autorisation et d'autres fonctionnalités de sécurité de manière simple, déclarative et de haut niveau.

Outline

Conclusion

Pour résumer, on donne juste un aperçu des fonctionnalités de sécurité de Java. Par conséquent, chacun des domaines abordés mérite une exploration plus approfondie.