Introduzione alla sicurezza in Java

Sicurezza dell'Informazione L-M
AA 2009-2010

Anna Riccioni
anna.riccioni@unibo.it

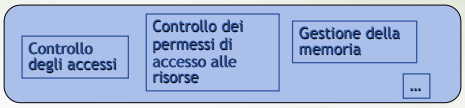
Sicurezza in Java

- Java è una piattaforma *sicura*
  - Linguaggio nato per:
    - applicazioni multipiattaforma
    - applicazioni da distribuire attraverso internet
    - applicazioni da eseguire su diversi dispositivi (cellulari, ...)
  - Sicurezza: obiettivo di progetto già dalle prime release
    - struttura del linguaggio
    - controlli a tempo di complilazione / caricamento / esecuzione
    - security model
    - architettura di supporto alla crittografia

Sicurezza e Crittografia

- Sicurezza (*computer security*)
  - applicazione di misure orientate a far sì che determinate informazioni, nel momento in cui vengono elaborate, archiviate o comunicate siano affidabili ed accessibili alle sole entità autorizzate
- Crittografia (*cryptography*)
  - applicazione di tecniche di codifica dell'informazione che la trasformino in un formato decifrabile solo dal destinatario designato

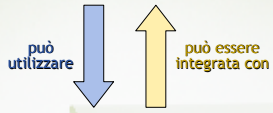
Sicurezza e Crittografia

- Sicurezza (*computer security*)
  - 

- Crittografia (*cryptography*)
  - 

Sicurezza e Crittografia

- Sicurezza (*computer security*)



- Crittografia (*cryptography*)

Sicurezza in Java

- Linguaggio Java
- SDK
  - piattaforma su cui eseguire applicazioni Java in modo sicuro
- API, strumenti e servizi per la sicurezza e la crittografia
  - framework che consente ad amministratori e programmatori di gestire applicazioni in modo sicuro e di scrivere applicazioni che si appoggiano ad algoritmi, meccanismi e protocolli per la sicurezza

Sicurezza in Java

- Meccanismi di controllo per evitare errori causa di comportamenti imprevedibili delle applicazioni
  - linguaggio fortemente tipizzato
  - controllo degli indici negli accessi a stringhe e array
  - gestione automatica della memoria (garbage collection)

Sicurezza in Java

- Livelli di visibilità differenziata applicabili a classi, metodi e campi
  - private
  - protected
  - public
  - package

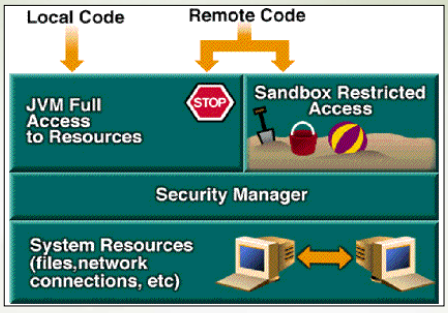
Sicurezza in Java

- Verifica, prima dell'esecuzione, della conformità del bytecode alle Java Language Specification
  - rispetto del formato e vincoli strutturali imposti per i file .class che la JVM deve interpretare
  - assenza di stack underflows / overflows
  - nessuna violazione della memoria
  - nessun cast non permesso

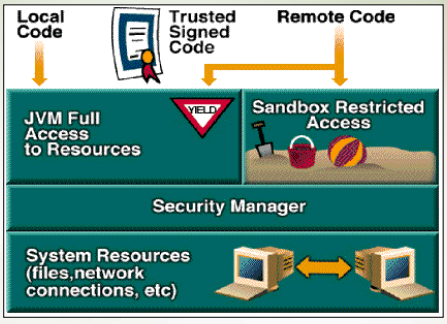
Sicurezza in Java

- Ulteriori meccanismi di controllo a run-time
  - Class Loader: gestisce il caricamento e il linking delle classi coinvolte nell'esecuzione all'interno degli opportuni namespace
  - Security Manager: valida gli accessi alle risorse di sistema mediati dalla JVM
  - modello a SandBox: limita il contesto di esecuzione di codice potenzialmente dannoso

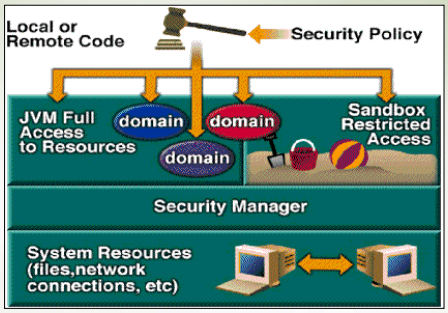
Security Model (Java 1.0)



Security Model (Java 1.1)



Security Model (da Java 1.2)



Security Model (da Java 1.2)

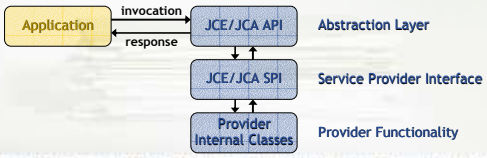
- Evoluzione rispetto al modello precedente
  - superamento del modello ON/OFF: sono possibili diversi domini intermedi tra la SandBox e l'accesso completo alle risorse
  - maggiore granularità a livello di insiemi di permessi
  - maggiore flessibilità
    - differenziazione di permessi in funzione di singole operazioni e singoli utenti
    - aggiornamento dinamico delle politiche di sicurezza

Crittografia e Sicurezza in Java

- Dalla versione 1.4 di Java:
  - integrazione in un'unica Java 2 Security Platform di
    - Java Cryptography Architecture
    - Java Cryptography Extension
    - Java Secure Socket Extension
    - Java Authentication and Authorization Service
  - aggiunta dei package
    - Java Certification Path
    - Java Generic Security Service

Java Cryptography Architecture

- Framework di base per la Crittografia
- Provider-based Architecture
  - indipendenza dall'implementazione
  - interoperabilità (provider e applicazioni non sono necessariamente strettamente legati)
  - estendibilità

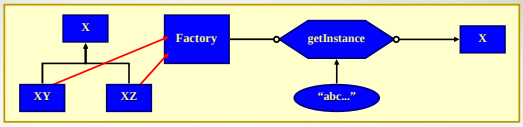


Provider-based Architecture

- Cryptographic Service Provider
  - uno o più package che implementano uno o più servizi crittografici esposti dalla JCA
  - più provider installati, anche se di produttori differenti, possono coesistere
- Responsabilità
  - un'applicazione richiede un oggetto (`MessageDigest`)...
  - ... che implementa uno specifico servizio (hash calcolato in base all'algoritmo `SHA-1`)...
  - ... e riceve un'implementazione fornita da uno dei provider installati (`BouncyCastle`)

Provider-based Architecture

- Pattern Factory



- Implementazione
  - classi astratte di tipo Engine
    - dichiarano le funzionalità offerte
  - classi concrete contenute nel provider
    - implementano le funzionalità dichiarate

Provider-based Architecture

- Concretamente
  - un oggetto viene istanziato
    - non più con la parola chiave new
    - ma grazie al metodo statico `getInstance(String nome)`
- Esempi
  - `MessageDigest hash = MessageDigest.getInstance("SHA-1");`
  - `KeyGenerator kGen = KeyGenerator.getInstance("TripleDES");`

Provider-based Architecture

- Indipendenza dall'implementazione
  - più provider possono coesistere nello stesso JRE
  - provider diversi possono offrire funzionalità diverse
  - provider diversi possono offrire implementazioni diverse delle stesse funzionalità

- Vantaggio
  - scegliere autonomamente il provider più adatto alle specifiche esigenze

Provider-based Architecture

- Scelta consapevole del provider
  - il nome del provider viene specificato come parametro all'interno del metodo `getInstance()` utilizzato per istanziare l'oggetto richiesto
- Esempi
  - `MessageDigest hash = MessageDigest.getInstance("SHA-1", "BC");`
  - `KeyGenerator kGen = KeyGenerator.getInstance("TripleDES", "BC");`

Installazione di un CSP

- Installazione statica
  - inserire l'archivio jar che implementa il provider tra le librerie esterne del JRE
    - *<JAVA_HOME>\jre\lib\ext*
  - modificare il file **java.security** che si trova nella directory *<JAVA_HOME>\jre\lib\security* aggiungendo il nome del provider come ultimo elemento della lista che ha il formato
    - security.provider.*n=provider*
- Esempio
  - `security.provider.7 = org.bouncycastle.jce.provider.BouncyCastleProvider`

Installazione di un CSP

- Installazione dinamica
  - il provider va importato all'interno dell'applicazione
  - il metodo `addProvider(String nome)` della classe `java.security.Security` consente di aggiungere il provider desiderato a run-time
- Esempio
  - `import java.security.Security;`
  - `import org.bouncycastle.jce.provider.BouncyCastleProvider;`
  - `Security.addProvider(new BouncyCastleProvider());`

Provider-based Architecture

- Nessuna preferenza nella scelta del provider
  - il nome del provider non viene specificato come parametro all'interno del metodo `getInstance()` utilizzato per istanziare l'oggetto richiesto

- Ricerca ordinata per preferenze
  - file *java.security*
  - la ricerca continua finché non si trova il primo provider che implementa la funzionalità richiesta

Provider-based Architecture

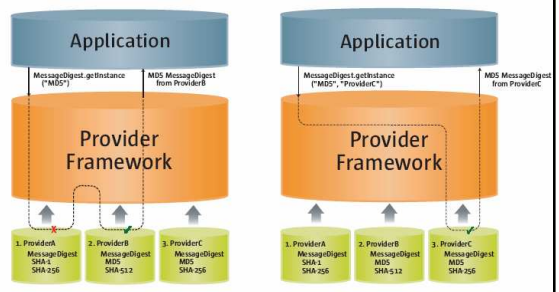


Figure 1 – Provider searching

Figure 2 – Specific provider requested

Provider Sun

- Integrato nel JDK
- Include implementazioni di:
  - algoritmo DSA
    - KeyPairGenerator per DSA
    - AlgorithmParameter per DSA
    - AlgorithmParameterGenerator per DSA
    - KeyFactory per DSA
  - algoritmi MD-5 e SHA-1 (hash)
  - algoritmo SHA1PRNG (SecureRandom)
  - CertificateFactory per X.509 e CRLs
  - KeyStore proprietario JKS

Provider BouncyCastle

- Non integrato nel JDK (va installato)
- Implementa ed estende le tecniche crittografiche definite dalla JCA e dalla JCE
  - implementazione alternativa degli algoritmi di cifratura (AES, DES, TripleDES, PBE, ...)
  - algoritmi di cifratura asimmetrica (RSA, ElGamal)
  - algoritmi di firma digitale
  - padding (None, PKCS1PADDING)
  - facilities per diversi protocolli di KeyAgreement
  - facilities per utilizzare cifrari simmetrici a flusso e a blocchi
  - ...

Architettura Java per la sicurezza