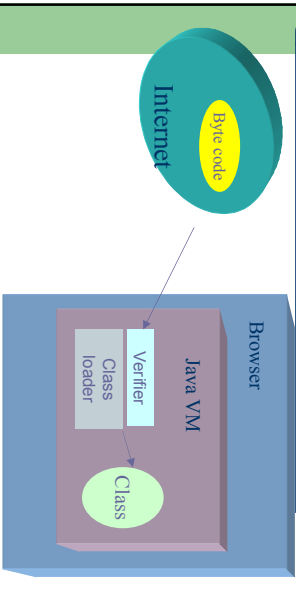


Loading external bytecode



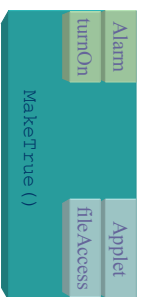
L'architettura base di sicurezza

- La sicurezza di base e' garantita attraverso :
 - progettazione object-oriented;
 - type safety;
 - risoluzione automatica dei compiti "difficili":
 - gestione automatica della memoria;
 - controllo del range per stringhe ed array;
 - garbage collection
 - gestione delle eccezioni

Type safety ↔ Sicurezza

- I programmi non possono effettuare delle operazioni dannose sugli oggetti.

- Esempio:



- Type checking statico vs dinamico

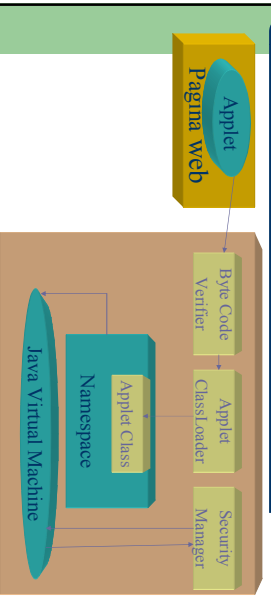
Componenti base di sicurezza

Goal: assicurarsi che le specifiche del linguaggio Java e della JVM siano osservate e implementate correttamente!

Le componenti base del sistema di sicurezza sono:

- Bytecode Verifier
 - Verifica la correttezza del codice prima di essere eseguito
- Class Loader
 - Aggiunge classi esterne all'ambiente Java
- Security manager
 - Controlla l'accesso a risorse protette

Il ciclo di vita di un applet

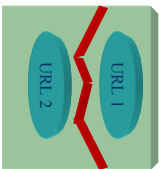


Il Bytecode Verifier

- Verifica il formato e la struttura del byte code
- Un theorem prover (analisi data-flow) assicura che:
 - Non si creino falsi puntatori
 - Non si violino restrizioni di accesso
 - Si acceda ad oggetti del tipo corretto
 - Non ci siano stack overflows
- Sia corretto il numero e il tipo dei parametri nelle chiamate di metodi
- Non ci siano conversioni illegali (intero-> puntatore)

Il Class Loader

- Il Class loader determina il modo in cui nuove classi vengono aggiunte al runtime :
 - trova e carica il byte code
 - definisce *namespaces separati*
Es. unico ns per i files e per ogni network source
- I Class loader possono essere ridefiniti dall'utente



Dynamic Class Loading

- Il primordial class loader ha il compito di fare il bootstrap del sistema
 - usa i meccanismi di accesso ai file forniti dal s.o.
 - Carica `c:\classes.zip` che contiene i le Java API



Il Security Manager

- Il Security Manager controlla l'accesso ad operazioni potenzialmente dannose;
- Prima di consentire l'accesso a tali operazioni le Java API consultano il Security Manager:
 - se il codice non e' trusted viene lanciata una *security exception*
 - altrimenti l'operazione viene eseguita

I compiti del Security Manager

- Il Security Manager :
- Evita che nuovi classloaders siano installati
- Protegge l'accesso reciproco fra thread appartenenti a gruppi diversi
- Controlla l'esecuzione delle applicazioni
- Controlla la possibilità di terminare la VM
- Controlla l'accesso ad altre applicazioni
- Controlla le operazioni sui file system
- Controlla le operazioni sulle connessioni di rete
- Controlla l'accesso a package di sistema

Il modello di Sicurezza in JDK 1.0



La sandbox

- Gli applet non possono:
 - accedere a file locali;
 - aprire connessioni se non all'host di origine;
 - accedere o cambiare alcune proprietà di sistema
 - lanciare programmi in locale
 - creare od accedere a thread di altri gruppi
- Possono accedere alle proprietà:
 - `java.version`, `vendor`, `class.version`, `os.name`, `os.version`, `os.arch`, `file-path-line.separator`

Sicurezza in JDK 1.0: La sandbox

- La Sandbox protegge l'accesso a tutte le risorse del sistema;
- I programmatori di applicazioni possono ridefinire un nuovo SecurityManager per "uscire" dalla sandbox

Manca la flessibilità:
Es. Un applet non può aggiornare le quotazioni delle azioni nel mio file portfolio

Il modello di sicurezza di JDK 1.1



Il modello di sicurezza di JDK 1.1

- La firma del codice può essere usata per consentire maggiori privilegi agli applet
- La politica di sicurezza per gli applet firmati e binaria (o tutto o niente)
- Applicazioni eseguite localmente sono lanciate fuori dalla sandbox, senza possibilità di controllo
- Il codice che si trova sul CLASSPATH è trusted

Il modello di sicurezza di JDK 1.2

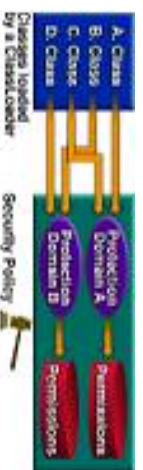


Caratteristiche del modello di sicurezza di JDK 1.2

- API per la sicurezza (JCE/JCA)
- Controllo di accesso fine-grained
 - Il codice è soggetto ad una security policy
- Meccanismi di controllo ben definiti
- Nuovo Security Manager/ Access Controller

Il modello di sicurezza di JDK 1.2

Ogni classe ha un insieme di permessi determinati dal dominio di appartenenza definiti dalla security policy



CodeSource

- Ogni pezzo di codice ha una origine ed una firma che ne definisce l'identità.
- Ogni pezzo di codice è definito da:
 - "The JavaSoft Division Security Group"
 - "Rossi, Paolo"
- Da dove proviene il codice (URL)
 - file:/home/paolo/classes/
 - http://java.sun.com/security/util.jar
- Le informazioni sono mantenute a runtime nell'oggetto `java.security.CodeSource`

Code source & Protection Domain

- I permessi vengono accordati in base a:
 - CodeSource (dove)
 - Code Signer (Chi ha firmato)
- Le diverse politiche riguardano insiemi di classi con lo stesso origine e stesso firmatario
 - Gli insiemi di classi formano un "Protection Domain"
- Ogni classe appartiene ad un singolo ProtectionDomain
- Esistono due categorie di domini:
 - System domain che protegge l'accesso alle risorse del sistema
 - Application domain

Politiche di sicurezza

Il comportamento di JRE e' specificato dalla politica di sicurezza adottata:

- Matrice di controllo di accesso che mappa insiemi di proprietà del codice in esecuzione in insiemi di permessi

java.sun.com	Code	Permessi
Read/write /tmp		
r/w /home/dinamo		
DiaunisaIt signed		

Internamente la politica implementata e' rappresentata da un oggetto che il SM puo' consultare `java.security.Policy`
Esternamente la politica e' rappresentata da un file ASCII (`java.policy`)

Il file java.policy

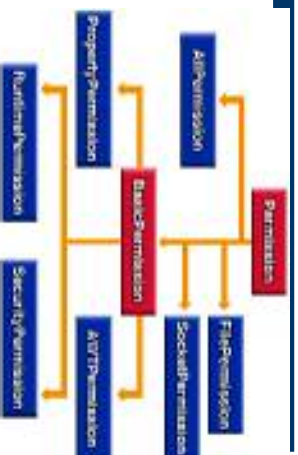
- ```
keystore "keystoreFile";
grant [codebase "<URL>"]
 [signedBy "<alias>"]
 {
 permission [permission class]
 ["target"],
 ["actions"];
 permission ...
 };
```

```
Es. grant CodeBase "http://www.dia.unisa.it/" ,
SignedBy "*"
{ permission java.io.FilePermission
 "read/write"/, "tmp";
 permission java.net.SocketPermission "connect",
 "*.*.unisa.it:"; }
```

## Permessi

- Possono essere solo positivi e vengono accordati su:
  - un target ("`home/schemeri/readme`")
  - una azione ("`read/write`")
- Esp. `new SocketPermission ("www.unisa.it: 1023", "connect")`
- Possono implicare altri permessi
  - Es. `FilePermission ("ALL_FILES"/, "read")` implica `FilePermission ("/tmp/prova.txt"/, "read")`
- Possono essere definiti dall'utente derivando dalle classi già presenti divisi in categorie File Access, System, AWT, Network.
  - Es. `java.io.SocketPermission "*" :1023"/, "connect")`
  - `java.io.FilePermission "/tmp/" , "read, write")`
  - `java.io.lang.RuntimePermission "getClassLoader")`
  - `java.awt.AWTPermission "accessEventQueue")`

## Gerarchia degli oggetti Permission



## JDK1.2 Controllo di Accesso

- L' AccessController controlla che tutti i domini nella catena di chiamate abbiano i giusti permessi



## Uso dell' Access Controller

- In JDK 1.1i controlli usano il Security Manager
- In Java 2 si usa Access Controller:  

```
FilePermission p =
 new FilePermission("/tmp", "read");
AccessController.checkPermission(p);
```
- Per compatibilità, i metodi check\*\*\* del SM sono presenti ma implementati usando l'Access Controller

## Controllo di accesso con più domini

- L'insieme dei permessi di una thread è data dall'intersezione dei permessi di tutti i domini attraversati



## Blocchi Privileged

- Come si fa ad accedere a risorse protette? Il codice di sistema che gestisce la risorsa invoca il metodo doPrivileged() della classe AccessController che permette di ignorare i precedenti chiamanti:

```
void changePasswd() {
 // ...normal code here
 AccessController.doPrivileged(
 new PrivilegedAction() {
 public Object run() {
 // Open file for read/write
 ...return null; } });
}
```

## Controllo di Accesso con un blocco Privileged

- La classe Bar può accedere alla lettura del file, sebbene non abbia i permessi poiché ha un blocco privileged



## Algoritmo di Controllo Accesso

```
void checkPermission(Permission p) {
 foreach (caller) {
 if (the caller doesn't have permission)
 throw new AccessControlException(p);
 if (caller is marked as privileged)
 return;
 }
 // Access Granted
 return;
}
```

## Sicurezza per gli Applet

- Ogni applet è soggetto al security manager per default e l'utente non può cambiare il SM dell'applet. Le applet eseguite all'interno del browser non devono poter accedere a risorse protette. La sicurezza è difficile da garantire perché:
- Il codice scaricato non è fidato
  - Modello sandbox o applet firmati
  - È impossibile usare la linea di comando
  - Non è possibile adattare facilmente ad esempio un policy file
- Non tutte le funzionalità di sicurezza sono implementate
- Si utilizza il Java-plug-in sia per Netscape che per Explorer

## Regole implementate nei Browser

- I browser ridefiniscono proprie Virtual Machine e propri Classloader e Security Manager
  - AppletSecurityManager dal package sun.applet in JDK1.1
  - AppletSecurity dal package sun.applet in JDK1.2
  - AppletSecurity dal package netscape.security in Netscape 4.0
  - AppletSecurity dal package netscape.applet in Netscape 3.0
  - StandardSecurityManager dal package com.ms.security in Internet Explorer
- La sottoclasse del SM definisce la politica di sicurezza. Applets:
  - non possono accedere ai file locali;
  - aprire connessioni se non all'host di origine;
  - possono leggere solo 9 proprietà di sistema (VM version, ...)
  - gli applet caricati con `FILE` fuori dal `CLASSPATH` usano l'Applet Class Loader

## Sicurezza per la Microsoft VM

- Microsoft Security Bulletin MS03-011 (816093)  
*Originally posted: April 09, 2003*  
This new security vulnerability affects the ByteCode Verifier ... the ByteCode verifier does not correctly check for the presence of certain malicious code when a Java applet is being loaded.
- Microsoft Security Bulletin MS02-069 (810030)  
*Originally posted: December 11, 2002*  
Impact of vulnerability: Eight vulnerabilities, the most serious of which would enable an attacker to gain control over another user's system.

## Microsoft e Java

- Microsoft Moves Forward with Removal of Java from Products  
On February 3, 2003, the Fourth Circuit Court of Appeals granted a stay pending appeal of a preliminary injunction order entered by the District Court in Baltimore that required Microsoft to distribute Sun's Java runtime environment (JRE) with the Microsoft® Windows® XP operating system and to halt "separate" distribution of the Microsoft virtual machine (Microsoft VM).
- As part of our January 2001 settlement agreement with Sun, which was intended to resolve disputes concerning Microsoft's development and distribution of the Microsoft virtual machine (Microsoft VM), Microsoft will no longer be authorized by Sun to have the ability to support the Microsoft VM after January 2, 2004. As a result, we will stop including the Microsoft VM in new releases of Windows and other products, and we will stop distributing Microsoft® Visual J++® as well.

## Java Plug-in

- Il plug-in Java supporta il modello di sicurezza di Java 2 SDK
- Tutti gli applet sono eseguiti sotto lo standard applet security manager
- Operazioni pericolose non sono consentite
- Per garantire maggiori permessi l'utente deve configurare il file di certificazioni (keystore)
- Nella versione 1.3, il `sun.plugin.security.PluginClassLoader` supporta la verifica di firme RSA

## API per la sicurezza in Java

- Le API per la sicurezza in Java consentono di incorporare funzionalità per la sicurezza nei programmi Java. Supportano:
- Crittografia
  - Autenticazione ed autorizzazione
  - PKI
  - Comunicazioni sicure
  - Web Services

## API per la sicurezza in Java

Nella release JDK 1.1 viene introdotta la Java Cryptography Architecture (JCA)

- Firme digitali, message digest, generazione di chiavi

Nella release Java 2 vengono introdotti:

- Key factories, creazione e gestione del keystore, creazione e gestione dei parametri per gli algoritmi, certificate factory
- Java Cryptography Extension (JCE)
  - o cifratura (DES, 3DES, AES), scambio di chiavi, MAC (MD5, SHA1), gestione di certificati

## Java 2 release 1.4

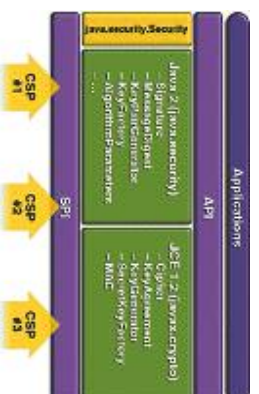
- La JCE viene assorbita nelle API core.
- Vengono integrati anche:
  - Java Secure Socket Extension (JSSE)
    - API per socket SSL v3 e TLS 1.0
  - Java Authentication and Authorization Service (JAAS)
    - Autorizzazione in base a chi esegue i codice, autenticazione (login)
- Vengono introdotti:
  - Java Generic Security Service (JGSS)
    - Kerberos V5.
  - Java Certification Path
    - Gestione di catene di certificati

## API per la sicurezza in Java

Nella release 1.5:

- Miglioramenti alla JCE
- Elliptic Curve Cryptography
- XML Security
  - XML Digital Signature
  - XML Digital encryption

## JCA/JCE



## Tools: keystore

- Il keystore e' un database protetto per memorizzare chiavi e certificati
  - E' implementato tramite un file (.keystore)
  - E' protetto da password
  - ogni entry è protetta ed e' associata a degli alias
- La corrispondente classe fornisce metodi per accedere al database tramite una SPI
  - Cioè c'è una classe astratta corrispondente KeyStoreSpi nel java.security package, che definisce i metodi che i "providers" devono implementare

## Tools: keytool

- Il Keytool gestisce il keystore mettendo a disposizione utility per:
- creare coppie di chiavi
  - creare e gestire entry nel keystore
    - Import ed export di certificati
    - genera richieste di certificazioni per la CA
- Supporta certificati X.509

## Tools: policytool

- Utilità per creare e modificare file policy
  - interfaccia grafica per la creazione di policy entry
  - permette di
    - aggiungere o revocare permissions
    - specificando il tipo e il target
  - specificare azioni
  - specificare le firme autorizzate

## Tools: jarsigner

- Utilità per firmare e verificare file JAR
  - utilizza le informazioni contenute nel keystore
  - usa algoritmi SHA e DSA o MD5 e RSA
  - Il file JAR generato ha due file aggiuntivi:
    - un file manifest con estensione .sf
    - per ogni sorgente in Jar lista lista il nome del file., il nome dell'algoritmo usato e il valore digest
    - un file di firma con estensione .dsa

## Verifica con jarsigner

- Jarsigner -verify example.jar
  - verifica la firma dello stesso file sf
  - verifica ogni entry del file sf con la corrispondente del file manifest
  - calcola il digest per ogni file che ha una entry in sf e verifica i valori

## Bibliografia

- G. Mc Graw, E. W. Felten  
"Securing Java"  
Wiley & Sons
- L. Gong  
"Inside Java 2 Platform Security"  
Addison Wesley