Consiglio Nazionale delle Ricerche

Tecnologie per l'esecuzione di codice distribuito attraverso World Wide Web

M. Casati

IIT B4-02/2002

Nota Interna

Luglio 2002

Istituto di Informatica e Telematica

Tecnologie per l'esecuzione di codice distribuito attraverso World Wide Web

Dott. Marco Casati
CNR - Area della Ricerca di Pisa
Istituto di Informatica e Telematica

16 luglio 2002

Indice

Capitolo 1

Esecuzione di codice distribuito attraverso World Wide Web

1.1 Benefici e rischi di sicurezza

Negli ultimi anni le pagine Web non sono piú considerate solo come un semplice insieme di documenti, contenenti testo ed immagini, eventualmente uniti da collegamenti ipertestuali ma, come un mezzo per la diffusione e lo sviluppo di nuove applicazioni. Le tecnologie attuali consentono di inserire al loro interno dei veri e propri programmi che, una volta scaricati da server remoti, possono essere direttamente eseguiti sulla macchina locale.

L'impiego di pagine dal contenuto eseguibile comporta un vantaggio per l'utente in quanto gli permette un approccio molto piú dinamico e interattivo con la rete ma, al tempo stesso introduce notevoli problemi, soprattutto dal lato della sicurezza. L'esecuzione in locale di un programma scaricato dalla rete ha in sé tutte le potenzialitá per arrecare danno all'utente e il rischio é tanto piú grande quanto maggiore é la capacitá del programma di accedere alle funzionalitá della macchina.

Sono diverse le tecnologie che consentono di realizzare pagine Web dai contenuti eseguibili tra le quali i linguaggi macro, i JavaScripts e i VBScripts, ma le piú diffuse e complete sono le Java Applet e i Controlli ActiveX. Questi due ultimi strumenti se da un lato sono concettualmente simili, in quanto rappresentano dei piccoli programmi eseguibili sulla macchina client, sono altresí ispirati a filosofie completamente diverse per quanto riguarda il problema della sicurezza. Java privilegia l'aspetto della sicurezza dell'utente, implementando un meccanismo di difesa che impedisce alle applet di eseguire operazioni illecite. ActiveX invece, privilegia la potenzialitá operativa lasciando ai controlli la massima libertá di esecuzione e

demandando all'utente il compito di gestire la sicurezza. Quindi a prima vista si puó asserire che Java é piú protettivo rispetto ad ActiveX ma offre minor capacitá operativa anche se, come vedremo in seguito, lo scenario si é sostanzialmente modificato.

1.2 Il modello Sand-box

L'ambiente di sviluppo Java comprende tre componenti:

1. Un linguaggio di programmazione che compila il codice in un formato intermedio e indipendente dall'architettura, chiamato *bytecode*.
2. La *Java Virtual Machine* (JVM) che esegue il bytecode generato.
3. Un ambiente di esecuzione nella JVM che fornisce alcune funzionalitá di base, utili per costruire applicazioni.

Il modello originale di sicurezza fornito dalla piattaforma Java1.0, noto con il termine *Sand-box*, permette di realizzare un ambiente di esecuzione molto ristretto e di conseguenza molto sicuro, in cui eseguire il codice scaricato dalla rete. In questo modo, il codice locale ha pieno accesso alle risorse vitali della macchina (ad esempio il filesystem) mentre il codice remoto (nel caso di Java le applet) puó solamente accedere alle risorse esportate nella sand-box.

La sand-box é composta da tre componenti che lavorano insieme:

- **Bytecode Verifier**: assicura che solo i legittimi bytecode siano eseguiti.
- **Class Loader**: rende Java dinamico allocando a run-time le classi necessarie all'esecuzione.
- **Security Manager**: controlla l'accesso alle risorse cruciali del sistema.

La figura 1.1 mostra come le parti che costituiscono la sand-box interagiscono per garantire l'esecuzione di codice remoto in modalitá sicura. Il codice sorgente Java é compilato in un bytecode che é trasmesso attraverso il Web al browser che lo richiede; l'HTML in una pagina Web, specifica quale codice deve essere prelevato dal server. Quando l'applet in esecuzione richiede una operazione pericolosa, viene consultato il SM che, effettuati i controlli, concede o nega l'autorizzazione a proseguire l'elaborazione richiesta.

Bytecode Verifier

Quando un file Java viene compilato, si ottiene come risultato un codice (bytecode) indipendente dalla piattaforma utilizzata: é il bytecode che garantisce la portabilitá dei programmi scritti nel linguaggio Java. Il bytecode é poi memorizzato in un file binario in uno specifico formato (*class* file): é compito del Bytecode Verifier verificare che il file cosí generato sia corretto. Il processo di verifica é articolato in due passi:

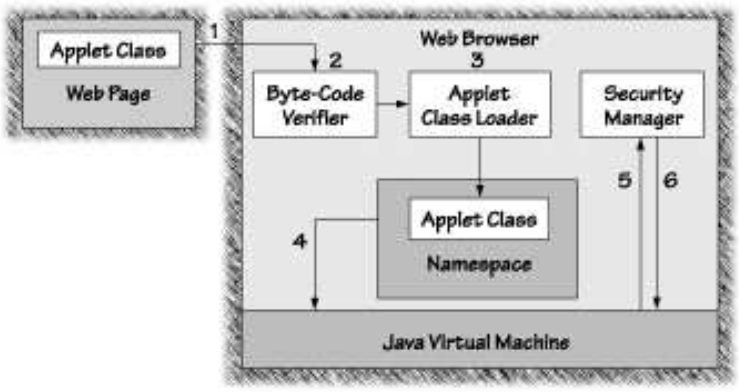

Figura 1.1: Componenti che costituiscono il modello di sicurezza sand-box di Java

1. **Internal check:** controlla che il formato del class file sia della forma corretta e successivamente verifica il bytecode analizzando il flusso dei dati.

2. **Runtime check:** conferma l'esistenza e la compatibilitá di classi, campi e metodi riferiti simbolicamente. Solo il codice che passa entrambe le verifiche é eseguito.

Class Loader

In Java tutto il codice, proveniente sia da disco che dalla rete, é caricato in memoria per mezzo del Class Loader. Quest'ultimo determina quando e come nuove classi possono essere inserite nell'ambiente di esecuzione run-time e assicura che parti fondamentali dell'ambiente di esecuzione non siano rimpiazzati da codici impostori. Il Class Loader svolge inoltre le due seguenti funzioni:

1. Ricerca del bytecode associato ad una determinata classe.

2. Definizione e gestione del *namespace*.

Security Manager

Il Security Manager é l'oggetto Java che esegue i controlli durante l'esecuzione dei metodi pericolosi. Il codice nella libreria Java consulta il Security Manager ogni volta che é richiesta una procedura potenzialmente rischiosa. Il SM prende la decisione finale se l'operazione deve essere permessa o respinta; in quest'ultimo caso una *Exception Security* é generata. Le decisioni del SM sono prese considerando l'origine della classe che ha richiesto l'operazione. In questo modo vengono incapsulate e protette le risorse che potrebbero essere usate illecitamente dal codice mobile.

1.3 Esecuzione di Controllo ActiveX

L'architettura OLE - COM - ActiveX é composta da un insieme di tecnologie software che, sostanzialmente, hanno tutte il medesimo scopo di consentire una maggiore interazione tra programmi software differenti residenti, o meno, sulla stessa macchina, in modo da consentire un sempre miglior utilizzo da parte dell'utente finale. La tecnologia ActiveX, ultima nata tra quelle basate sul modello COM (Component Object Model), é esplicitamente orientata verso il mondo Internet, introducendo un insieme di nuove caratteristiche che in alcuni casi sostituiscono o integrano le tecnologie precedenti, allo scopo di renderle piú adatte all'utilizzo attraverso la rete. Lo scopo principale di ActiveX é quello, come suggerisce il termine stesso, di rendere dinamiche le pagine Web, obiettivo che in un certo senso la accomuna e al tempo stesso contrappone alle applet Java.

Un Controllo ActiveX é un oggetto, che incorpora metodi, variabili, ed altri oggetti generici. Dato che questa tecnologia é presente sul mercato da alcuni anni, il programmatore ha a disposizione una vasta collezione di controlli preesistenti fra cui scegliere per utilizzarli secondo le proprie esigenze. Nel caso in cui si decida di creare un nuovo controllo é possibile utilizzare un ampio insieme di linguaggi di programmazione tra i quali Visual Basic, C++, Delphi.

1.3.1 Utilizzo dei Controlli ActiveX

Per utilizzare un controllo ActiveX attraverso Internet sono richieste le due seguenti operazioni:

- Posizionare il controllo all'interno di una pagina HTML.
- Aggiungere dell'ulteriore codice all'interno della pagina per la gestione del controllo.

Per essere riconosciuto dal browser un controllo deve essere racchiuso tra i tag <OBJECT></OBJECT>. All'interno di questa sezione devono essere forniti al browser tutti gli estremi per l'identificazione del controllo e tutte le specifiche per una corretta inizializzazione dei parametri. Per le operazioni di gestione dei controlli tipicamente si utilizzano i linguaggi di script.

Quando un client richiede una pagina in cui é presente un controllo, il browser controlla l'esistenza del controllo sulla macchina locale e, in caso contrario lo carica e lo installa. A questo punto i parametri del controllo sono inizializzati in conformitá alle specifiche inserite nella pagina e ha inizio l'esecuzione. Durante l'esecuzione il controllo é gestito dal browser mediante le istruzioni rese disponibili dai linguaggi di script, che permettono di vedere gli ActiveX come comuni oggetti a cui é possibile accedere secondo le regole della programmazione ad oggetti. Al termine dell'esecuzione del controllo, la connessione con l'oggetto é chiusa ma, il controllo rimane installato permanentemente sulla macchina locale.

1.3.2 Problemi legati alla sicurezza

L'autoinstallazione del software sulla macchina del client unitamente all'accesso indiscriminato a tutte le risorse del sistema rendono l'utilizzo degli ActiveX rischioso. Per sopperire a questa importante lacuna é stata inserita una tecnica di certificazione, basata su firma elettronica, denominata CodeSign. In questo modo é possibile registrare i controlli in modo da identificarli prima del caricamento. É compito dell'utente, informato preventivamente sulle origini del controllo, valutare se fidarsi o meno, accettando o rifiutando l'installazione del controllo in memoria.

1.4 Esecuzione di Applet Java

L'esecuzione di una Java Applet avviene attraverso i seguenti passi:

- l'utente richiede una o piú applet ad un Web server.
- il codice é interpretato dalla Java Virtual Machine (JVM) e convertito in linguaggio macchina.
- i risultati sono visualizzati attraverso una Graphical User Interface (GUI).

Nella stesura del linguaggio, i progettisti della Sun hanno sacrificato alcune funzionalitá a vantaggio del controllo completo delle applet. Il modello sand-box, descritto nella sezione precedente, prevede una serie di barriere successive, che sono in grado di proteggere totalmente l'utente. La sicurezza offerta da questa soluzione é legata al modo in cui la sand-box é stata progettata e realizzata e di conseguenza ogni eventuale buco diventa una possibile via di accesso al sistema.

Le applet che girano in un browser hanno limitate capacitá di accesso alla macchina client; per esempio non possono accedere al file system e sono solamente in grado di leggere alcune proprietá di sistema invocando il metodo:

System.getProperty (String key)

Una applet non puó aprire connessioni di rete se non con l'host dal quale é stato scaricato e con quello specificato nel parametro *codebase* nell'applet tag. Inoltre, poiché Java non consente l'utilizzo dei puntatori non é possibile realizzare un'accesso diretto alla memoria, sia esso accidentale o intenzionale. Infine, la forte tipizzazione del linguaggio permette uno stretto controllo in fase di compilazione sulla correttezza del tipo dei dati.

Le applet possono essere caricate dalla rete ma possono anche risiedere direttamente sul file system locale: questa distinzione permette di distinguere quello che possono o non possono fare. Nel secondo caso, infatti, l'applet non é sottoposta alle restrizioni sopra descritte imposte dal security manager.

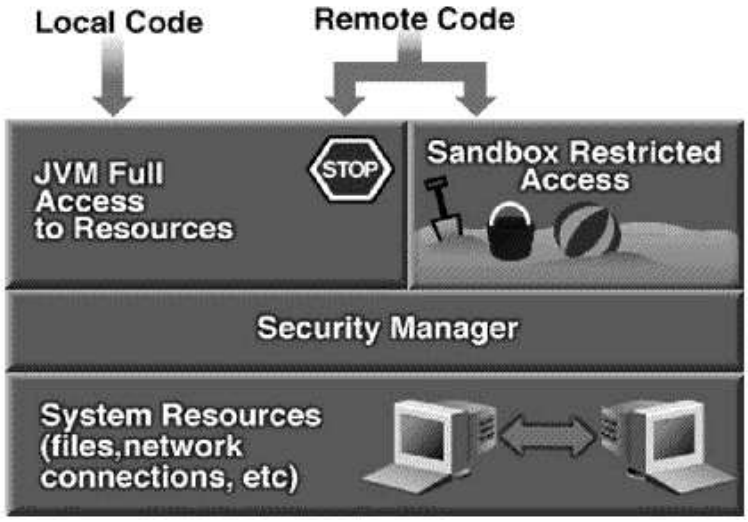

Figura 1.2: Il modello di sicurezza Java1.0

1.4.1 Modello Sand-box di Java1

La versione del JDK1.1 uscita nel 1997 ha apportato notevoli miglioramenti e cambiamenti al modello di sicurezza di base introdotto con la versione precedente. La novitá piú significativa é stata l'inserimento delle tecniche di autenticazione e dei meccanismi di controllo degli accessi alle risorse del sistema, basate sull'utilizzo della crittografia. La possibilitá di firmare digitalmente il bytecode generato, ed in particolare i file inclusi negli archivi Java (jar file), consente di aumentare la fiducia posta in un codice eseguibile. Il browser é in grado di verificare la firma e di garantire che il codice che si sta eseguendo é stato prodotto da una determinata persona o organizzazione. In questo modo é possibile istruire il browser ad accettare solamente le applet firmate dalle entitá che l'utente ritiene fidate. Le applet correttamente firmate sono trattate come codice locale e quindi possono accedere alle risorse del sistema; le applet non firmate continuano invece ad essere eseguite nella sandbox.

Il modello di sicurezza offerto dall'ambiente Java1.1 risulta essere meno restrittivo rispetto al modello sandbox ma non permette di attribuire alle applet accessi selettivi alle risorse: se la firma apposta al codice é verificata, l'applet acquisisce pieni poteri altrimenti, continua a non poter fare nulla. Questa forte limitazione é stata superata con il modello di sicurezza presente nella versione Java1.2, descritto nella sezione successiva.

1.4.2 Evoluzione del modello Sand-box in Java2

In scenari sempre piú orientati all'interconnettivitá e alla gestione di architetture distribuite, poter disporre di un sistema flessibile, e configurabile dinamicamente, é un'esigenza sempre piú diffusa e sentita. Il modello di sicurezza di Java2 risponde proprio a questa mancanza

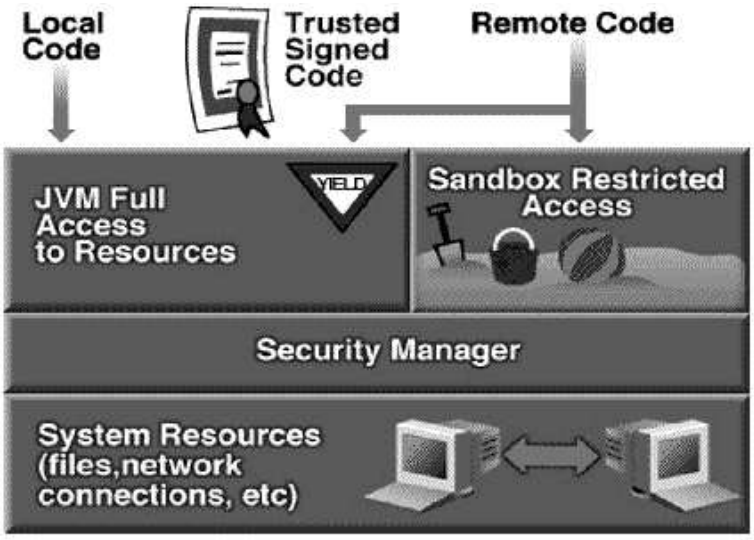

Figura 1.3: Il modello di sicurezza Java1.1

offrendo la possibilitá di creare varie tipologie di permessi a seconda delle esigenze, potendo controllare nei minimi dettagli tutti gli aspetti coinvolti. Risulta utile ed efficiente poter distinguere il tipo di operazione che una applet é in grado di eseguire in base alla provenienza (da dove la si é scaricata), oppure in base al tipo di operazione (su quale risorsa intende lavorare).

La descrizione del nuovo modello di sicurezza parte dal lato client, dove é possibile specificare, in base al proprietario dell'applet, il tipo di operazione che sará possibile eseguire in locale, e su quali risorse. Non si tratta quindi di un meccanismo on-off, ma della possibilitá di creare una politica di permessi molto dettagliata.

In questo modello al fianco del SM si trova l'**Access Controller** (AC), che offre la possibilitá di personalizzare gli accessi, permettendo ad un codice di fare gradualmente dei passi fuori dai vincoli imposti dalla sandbox, specificando gli adeguati permessi. Il SM continua ad essere presente per mantenere la compatibilitá con le applicazioni basate su Java1.1; in questo caso, opera da interfaccia per l'AC che effettua in background tutti i controlli. La possibilitá di modificare la gestione dei permessi consente di cambiare completamente la politica della sicurezza semplicemente modificando un file di configurazione (operazione possibile anche a run-time).

Il funzionamento dell'AC si basa sui seguenti concetti:

Provenienza del codice: specifica da dove il codice é stato scaricato, se si tratta di un host remoto, o caricato se proviene dal file system locale.

Permesso: un insieme di permessi che sono associati alle singole operazione eseguibili sulla macchina.

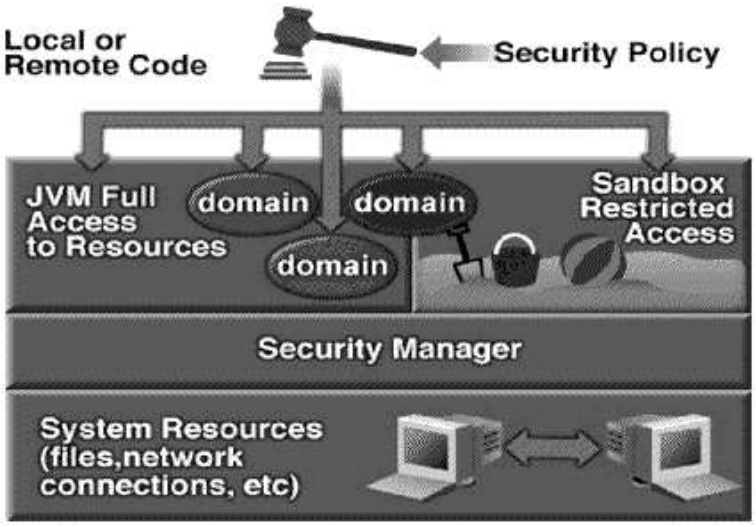

Figura 1.4: Il modello di sicurezza Java1.2

Politica di sicurezza: un insieme di associazioni che specificano le operazioni che possono essere eseguite e da chi.

Dominio di protezione: un insieme di privilegi associati ad un insieme di programmi Java (applet o applicazioni) con la stessa provenienza, e firmati digitalmente da utenti appartenenti allo stesso gruppo.

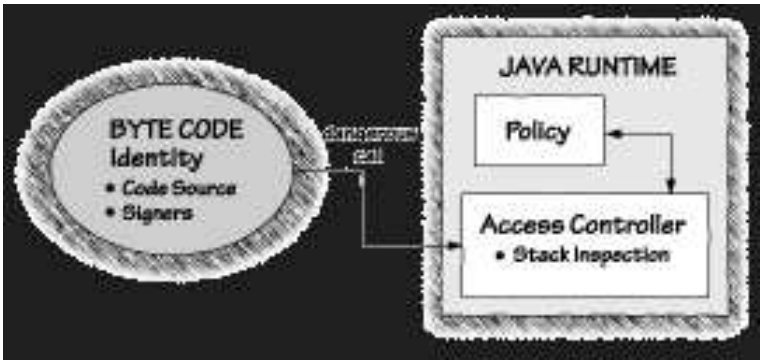

Figura 1.5: Il funzionamento dell'Access Controller in Java2

I permessi

L'elemento base su cui l'AC basa il suo funzionamento é l'oggetto *Permesso*, che corrisponde alla classe *java.security.Permission*. A questa classe sono associati due significati distinti:

quando un oggetto Permission é associato ad una classe, rappresenta l'insieme delle operazioni garantite dalla classe stessa; altrimenti consente di conoscere se il soggetto possiede i permessi per eseguire una data operazione. Il possedere una classe di permesso su di un file, non significa che é possibile accedere realmente al file ma solo se questa operazione é lecita.

Una classe Permission ha tre proprietá fondamentali:

Tipo: identifica il tipo di permesso che si intende specificare (accesso a file, socket).

Nome: il nome associato alla classe.

Azioni: in funzione del tipo di permesso, é possibile specificare una serie di operazioni che possono essere eseguite.

La politica

Dopo aver definito un insieme di permessi, si deve consentire all'AC di dedurre quale di questi devono essere applicati e su quali codici: la coppia AC+policy svolge le funzioni che precedentemente erano affidate al Security Manager. La politica é impostata in un file di configurazione che viene caricato durante il setup della Java Virtual Machine. Esiste per default un singolo file della politica del sistema e un singolo file della politica definita dall'utente. Il primo é localizzato in:

```
Solaris:  {java.home}/lib/security/java.policy
Windows:  {java.home}\lib\security\java.policy
```

dove *java.home* é una proprietá del sistema che identifica la directory in cui la versione del JDK é stata installata. Il secondo, in:

```
Solaris:  {user.home}/.java.policy
Windows:  {user.home}\.java.policy
```

dove *user.home* é una proprietá del sistema che specifica la home-directory dell'utente. Se nessuna politica viene trovata, si utilizza quella di default, ovvero quella che rappresenta il modello di sicurezza sandbox.

L'identitá del codice é confrontata con tutte le entry dell'oggetto Policy per determinare i permessi da assegnare al codice. Se entrambi i valori, origine e firma, sono soddisfatti, si ottiene una corrispondenza 1:1 tra l'identitá del codice e una entry dell'oggetto Policy. Questo significa che in termini di firma, l'URL che definisce l'origine del codice deve essere identica ad uno dei valori di una entry dell'oggetto Policy mentre, la chiave pubblica (associata alla chiave privata legata alla firma del codice) deve essere uguale alla chiave pubblica di un firmatario nella politica del sistema. Un codice puó essere firmato con una firma multipla. Nel caso in cui le firme, che un codice porta con se, hanno corrispondenze multiple con le entry nella politica, tutte le entry sono applicate in modo additivo. Questo significa che al codice sono assegnati tutti i permessi, dati dall'unione dei singoli permessi.

I domini di protezione

Un dominio di protezione rappresenta l'insieme di tutti i permessi che una certa classe (programma client sulla JVM) puó eseguire. Un dominio é rappresentato dalla classe *java.security.ProtectionDomain*, il cui costruttore é:

public ProtectionDomain(CodeSource cs, Permission p)

Associare un ProtectionDomain ad una certa classe, permette di specificarne il sito di provenienza (rappresentato dal nome della sorgente cs), la firma (sempre specificata in cs), ed il set di regole (p) alle quali deve sottostare. Java runtime mantiene una associazione tra codice e domini di protezione.

La politica di sicurezza puó specificare quali domini di protezione devono essere creati e che tipo di permessi devono essere assegnati. Esiste un dominio speciale, il dominio di sistema (system protection domain) che include tutto il codice del sistema caricato con il *Primordial Class Loader*. Questo dominio include le classi nella CLASSPATH e ha permessi privilegiati.

1.4.3 Ambiente di run-time Java: dalla VMJ1 al Java Plug-In

Le Java applet si sono diffuse molto rapidamente soprattutto grazie alla possibilitá di poter essere eseguite attraverso la JVM integrata nei browser. Questa caratteristica se da un lato consente di sviluppare e distribuire codice in modo rapido e semplice, al tempo stesso comporta alcuni problemi, tra i quali:

- Una minor fiducia nell'esecuzione di codice scaricato dalla rete.
- Un ridotto accesso alle opzioni di esecuzione della JVM, poiché quest'ultima é incapsulata nel browser.
- Insufficiente aggiornamento della JVM del browser.

Il primo problema é stato superato aggiungendo al modello della sand-box la possibilitá di firmare il codice eseguibile. L'utilizzo della JVM presente all'interno del browser rende piú difficile eseguire le applet con apposite policy, in quanto quest'ultime non possono essere specificate come parametro ma devono essere necessariamente incluse nei file di installazione dell'ambiente JRE (Java Run-time Environment). Il terzo problema, ovvero la mancanza di aggiornamento della JVM con le nuove caratteristiche di sicurezza del JRE, é stato risolto utilizzando il **Java plug-in**. I plug-in sono degli strumenti che permettono di estendere le funzionalitá di un programma. In modo particolare il Java plug-in permette di eseguire le applet nell'ambiente Java2 JRE e non in quello di default del browser. Le funzionalitá aggiuntive messe ha disposizione dal plug-in comprendono:

- Supporto completo per Java2 SDK.

- Installazione libera e gratuita: quando ad un browser é richiesto il caricamento di una pagina web che richiede il Java plug-in, é possibile scaricare ed installare liberamente i file necessari con un minimo intervento da parte dell'utente.
- Convertitore HTML: un semplice strumento che permette di convertire le pagine HTML che caricano applet in modo da istruire il browser a non utilizzare la sua JVM ma quella fornita dal plug-in.

Il Java plug-in fornisce quindi un ambiente di esecuzione per le applet Java che risulta indipendente dal browser (IE o NN) e dalla versione installata.

I Java plug-in e la sicurezza

Il Java plug-in supporta l'ambiente standard Java2 SDK e quindi le applet sono eseguite sotto il controllo del security manager, AC+policy, che previene l'esecuzione di possibili operazioni pericolose. Inoltre il plug-in consente di sviluppare applet firmate con algoritmi a chiave pubblica RSA e di eseguire lo stesso codice indipendentemente dal browser utilizzato. Questa operazione risulta essere notevolmente vantaggiosa in quanto fino all'introduzione della versione JDK1.2.2 era necessario sviluppare codice firmato in modalitá differenti a seconda del browser poi utilizzato.

I tools di sicurezza dell'ambiente SDK

I tre tool di sicurezza presenti all'interno dell'ambiente di sviluppo Java2 SDK sono:

1. **keytool:** gestisce un deposito per chiavi e certificati.
2. **jarsigner:** gestisce le operazioni di firma e verifica di un archivio jar.
3. **policytool:** gestisce la creazione di una policy attraverso una GUI.

Tutti e tre gli strumenti utilizzano il file *.keystore* che permette di memorizzare e successivamente recuperare chiavi e certificati attraverso un accesso nominativo per mezzo di *aliases*.

keytool

Permette di eseguire una serie di operazioni sul deposito delle chiavi tra le quali:

- Creazione di una coppia di chiavi pubblica/privata.
- Creazione di una richiesta di certificato verso una determinata Certification Authority (CA).
- Importare la risposta da parte della CA.

Keytool gestisce certificati nel formato X509 ma permette di utilizzare altri formati, tra i quali *pkcs#12*, specificando l'apposito provider. Questa operazione non é piú necessaria con la versione JDK1.4 in quanto l'ambiente JSSE (Java Secure Socket Extension) é giá presente nel kit di sviluppo. L'utente ha inoltre la possibilitá di specificare le tipologie di algoritmi che intende utilizzare per le operazioni di generazione di una coppia di chiavi e per la firma di un file.

jarsinger

Accede al deposito delle chiavi per recuperare la chiave privata e la catena dei certificati dell'utente che deve eseguire l'operazione di firma di un archivio Java. Per sicurezza sia il deposito che la singola entry riguardante la chiave privata sono protette tramite password.

policytool

Nell'implementazione di riferimento la policy puó essere configurata attraverso la specifica di uno o piú file di configurazione. Questi file contengono la definizione dei permessi che devono essere associati ad un particolare codice eseguito attraverso la rete o caricato dal file system. Policytool permette di creare e configurare la policy attraverso una interfaccia grafica. La policy é inizializzata la prima volta che viene eseguito il metodo *getPermissions* e successivamente ogni volta che é richiamato il metodo *refresh*. Questa operazione comporta il parsing dei file di configurazione e l'istanziazione dell'oggetto Policy.

Inserimento di nuova entry nel file di policy

Per inserire una nuova entry nel file di policy sono previste due possibili alternative:

1. Aggiungere il riferimento ad un nuovo file di privilegi, precedentemente creato attraverso lo strumento grafico *policytool*, assegnando ad una nuova proprietá il path del file. Questa operazione comporta l'aggiornameto del file *java.security* secondo il modello seguente:

   ```
   Esempio: policy.url.n= URL
   n: un numero.
   URL: il cammino di un file oppure un indirizzo web (http://).
   ```

2. Modificare direttamente il file *java.policy* aggiungendo una nuova concessione (entry grant) che permette di specificare uno o piú permessi (permission entry), secondo la seguente sintassi:

   ```
   grant signedBy "signer_names", codeBase "URL" {
      permission permission_class_name "target_name", "action";
   ```

signedBy: indica l'alias per un certificato inserito nel deposito *.keystore*. La chiave pubblica di questo certificato é utilizzata per verificare la firma digitale apposta al codice al quale si intende assegnare i privilegi. Questo attributo é opzionale; nel caso in cui non é specificato si intende qualsiasi firmatario.

codeBase: indica il luogo di provenienza del codice. Anche questo attributo é opzionale; se é omesso assegna i privilegi a qualsiasi codice indipendentemente dalla sorgente.

permission_class_name: indica il tipo di permesso che si intende assegnare al codice (FilePermission, SocketPermission).

target_name: indica la risorsa (oggetto destinazione) sulla quale i privilegi devono essere assegnati.

action indica il tipo di operazione che é concessa eseguire sulla risorsa. Questo attributo é utilizzato solo per alcuni tipi di privilegi (tipicamente FilePermission).

1.5 Confronto tra il modello di sicurezza di Applet Java e Controlli ActiveX

Come descritto nelle sezioni precedenti, sebbene le applet Java e i controlli ActiveX permettano entrambi di realizzare pagine Web dai contenuti eseguibili, in termini di sicurezza, gli approcci sono sostanzialmente differenti. I controlli ActiveX basano la loro sicurezza interamente sul giudizio dell'utente: una volta verificata la firma il browser é in grado di risalire al firmatario del programma e domanda se si intende accettare o meno il codice. L'utente ha solamente due alternative: accettare o rifiutare. Per compiere la sua scelta, l'utente puó basarsi sia sulla conoscenza diretta dello sviluppatore del controllo, che sulla catena dei certificati: se si fida della CA che ha emesso il certificato allo sviluppatore, si fida di conseguenza anche di lui. Se accetta, il controllo sará installato ed avrá un completo accesso alla macchina locale. Le situazioni piú critiche si verificano quando il programma che si intende eseguire é firmato da una persona sconosciuta o il certificato della CA non é installato nel repository del browser; in questi casi accettare il codice potrebbe provocare dei danni irreparabili. Le applet Java offrono un modello di sicurezza sostanzialmente diverso: analizziamo le possibili situazioni in riferimento all'ambiente Java2.0:

- Applet non firmata e policy di default: in questo caso si applica il modello di sicurezza della sand-box: se l'applet tenta di eseguire un'operazione non esportata dalla sand-box l'ambiente run-time genera una *SecurityException*.

- Applet non firmata e user policy: in questo caso la policy definita dall'utente puó abilitare una determinata applet, ad esempio scaricata da un particolare URL, ad uscire dai vincoli della sand-box.

- Applet firmata e policy di default: se la firma dello sviluppatore viene verificata e il certificato della CA é inserito nel repository del browser tra le CA attendibili, il Java plug-in visualizza una dialog per consentire all'utente di accettare o meno l'applet. In caso di consenso, l'applet ottiene *AllPermission*.

- Applet firmata e user policy: se la firma dello sviluppatore viene verificata e il certificato della CA é inserito nel repository del browser tra le CA attendibili, il Java plug-in non visualizza piú la dialog di sicurezza e l'applet viene eseguito con i soli privilegi accordati dalla policy definita dall'utente.

Capitolo 2

Creazione di una applet distribuita in Java

2.1 Specifica della applicazione di esempio

L'applet sviluppata permette di eseguire le operazioni di cifratura e decifratura asimmetrica di documenti secondo gli standard S/MIME-PKCS#7.

2.1.1 PKCS#7: Cryptographic Message Syntax Standard

Lo standard PKCS#7 descrive una sintassi generale per la rappresentazione dei dati da sottoporre a operazioni di cifratura e firma digitale. L'obiettivo dello standard é definire una struttura dati generica per incapsulare una serie di oggetti; questa struttura é ricorsiva e quindi in essa possono essere inserite piú struttura PKCS#7 dello stesso tipo.

2.1.2 S/MIME: Secure MIME

S/MIME definisce come inviare e ricevere dati MIME (Multipurpose Internet Mail Extension) sicuri, garantendo i seguenti servizi di sicurezza:

- Autenticazione.
- Integritá.
- Non ripudio.
- Privatezza.

In questo caso, i dati da cifrare, sono costituiti da oggetti MIME che vengono processati secondo le specifiche PKCS#7 e successivamente inseriti nuovamente in oggetti MIME.

2.1.3 Descrizione dell'operazione di cifratura

La cifratura dei documenti, PKCS#7 e S/MIME, viene eseguita secondo il seguente schema:

1. Generazione di una chiave simmetrica con cui il documento sará crittografato. Questa chiave, é anche chiamata chiave di sessione in quanto viene generata in modo casuale ogni volta che si deve creare un nuovo oggetto cifrato. Se il documento é destinato a piú persone, vengono fatte tante copie della chiave di sessione tanti quanti sono i possibili destinatari.

2. Ogni copia viene critttografata con la chiave pubblica associata al destinatario.

3. Cifratura del documento con la chiave di sessione.

4. Creazione della **busta crittografica** contenente il documento cifrato e le informazioni riguardanti i destinatari.

2.2 I rischi di sicurezza della applicazione di esempio

L'applicazione di esempio é stata firmata perché durante la sua esecuzione richiede dei privilegi che normalmente il modello di sicurezza sand-box non concede alle applet. Prima di tutto l'applicazione adotta i meccanismi di cifratura messi a disposizione della specifica SUN JCE (Java Cryptography Extension 1.2) ed in particolare il sistema dei Security Provider. Questo sistema permette ad una applicazione che necessiti di servizi crittografici, di utilizzare i servizi esportati da un provider semplicemente registrando nel proprio ambiente di esecuzione il riferimento a tale fornitore. L'applicazione accede direttamente al servizio attraverso il nome, senza doversi preoccupare di ricercare il provider che lo implementa. La registrazione di un nuovo provider é un'operazione privilegiata e di conseguenza per essere eseguita in una applet, senza generare eccezioni di sicurezza, richiede che il codice venga firmato.

Le operazioni di cifratura e decifratura richiedono accesso al file system locale, per leggere i file da crittografare, i certificati e scrivere i file da decifrare. La sand-box nega alle applet non firmate questo tipo di operazioni e di conseguenza anche la nostra applicazione richiede di essere firmata per essere eseguita correttamente.

2.3 Modalitá di generazione della applet firmata

La procedura per la generazione di una applet firmata che utilizzi le proprietá dei Java plug-in, richiede la disponibilitá dei seguenti strumenti:

- Uno strumento per la generazione della firma.
- Una coppia di chiavi pubblica/privata e una catena di certificati per garantire l'autenticitá della chiave pubblica.
- La classe java dell'applet ed eventuali altri file .class, inclusi in un archivio Java (file jar).

2.3.1 Strumenti di firma

Per firmare una applet tramite un algoritmo RSA, i Java plug-in supportano l'utilizzo di due strumenti:

1. **Jarsigner:** strumento inserito nell'ambiente Java2 SDK. Il comando é: *jarsigner*
2. **Netscape Signing Tool:** strumento fornito da Netscape. Il comando é: *signtool*

L'applicazione di esempio é stata firmata utilizzando lo strumento messo a disposizione da Sun; nel seguito sará descritta in dettaglio l'operazione di firma tramite questo tool. Per informazioni riguardanti signtool [2].

2.3.2 Certificati RSA

I certificati RSA possono essere ottenuti acquistandoli da una Certification Authority (CA). Nel nostro caso, il certificato é stato ottenuto dalla CA sviluppata presso lo IIT. A questo punto é necessario esportare dai certificati del browser il file con la chiave privata della persona che intende firmare il codice; per garantire la sicurezza della chiave al momento dell'esportazione deve essere inserita una password. Il formato standard di esportazione é un file PKCS#12 [6].

2.3.3 Creazione dell'archivio

Per utilizzare lo strumento *jarsigner* per firmare una applet tramite dei certificati RSA, l'applet deve necessariamente essere incapsulato in un archivio (file jar). Questa operazione puó essere eseguita utilizzando il comando *jar* ....

```
Esempio:
jar cvf C:\TestApplet.jar .
```

Il comando di esempio crea un file jar C:\TestApplet.jar, che contiene tutti i file inclusi nella directory corrente e nelle sue sottodirectory.

2.3.4 Firma dell'applet

Per firmare l'applet utilizzando *jarsigner* é necessario eseguire le seguenti istruzioni:

1. Utilizzare il comando *jarsigner* fornendo come credenziali la propria chiave privata. La chiave privata deve essere localizzata nel file *.keystore* ed é accessibile attraverso l'alias creato. Nell'applicazione di esempio, si é utilizzato un repository capace di contenere dati in formato PKCS#12.

   ```
   jarsigner -storetype pkcs12 -keystore nomefile.p12
   nomefile.jar "id_certificato"
   Enter keystore password:
   ```

   Per accedere alla chiave privata é necessario inserire la password di protezione.

2. Per verificare la corretta esecuzione della firma dell'applet é sufficiente eseguire il comando:

   ```
   jarsigner -verify -verbose -certs nomefile.jar
   ```

   Se la firma é correttamente verificata é possibile distribuire l'applet.

2.4 Modalitá di esecuzione all'interno del Java plug-in

Per eseguire correttamente l'applicazione di esempio é necessario utilizzare lo strumento *HTMLConverter*, messo a disposizione dalla Sun, per convertire il codice HTML della pagina che contiene l'applet in modo che durante l'esecuzione non sia utilizzata la JVM presente all'interno del browser di navigazione ma quella fornita insieme ai plug-in. L'HTMLConverter modifica il tag di definizione dell'applet <applet></applet> nei due nuovi tag:

```
Internet Explorer:   <OBJECT></OBJECT>
Netscape Navigator:  <EMBED></EMBED>
```

A questo punto é possibile distribuire l'applet firmata secondo le stesse modalitá con le quali si distribuisce una non firmata; la firma infatti, é nascosta nel file jar dell'applet e non compare nella pagina HTML.

Quando un utente che utilizza il Java plug-in richiede una pagina in cui compare l'applet firmata, il plug-in verifica la correttezza della firma e che la catena dei certificati RSA e il certificato della CA sia valido. Nell'applicazione di esempio, poiché la CA che ha emesso il certificato del firmatario non é una CA riconosciuta, é necessario inserire il certificato della CA tra quelli dei firmatari attendibili. Se questa operazione non viene eseguita, il plug-in

rileva un certificato CA non valido e con alcune versioni del JDK1.3.x é possibile che la la finestra di dialogo di sicurezza, descritta in dettaglio piú avanti, non sia visualizzata e l'applet non inizializzata. Se invece la verifica ha esito positivo, il plug-in visualizza la dialog in figura [5].

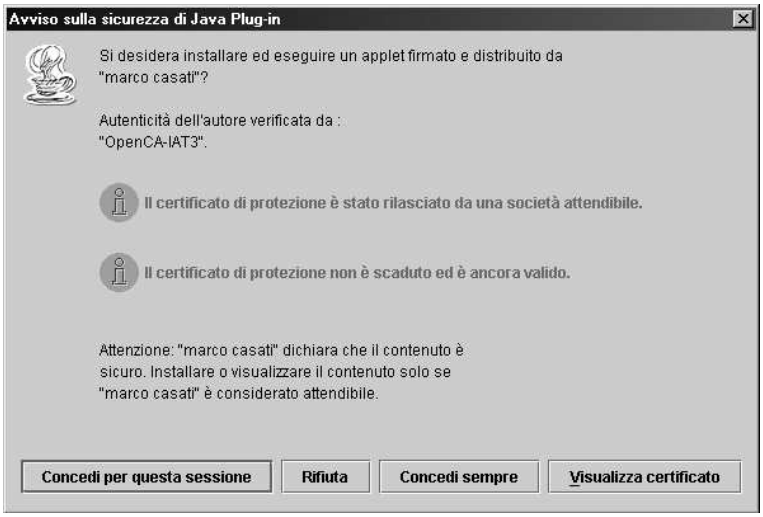

Figura 2.1: Finestra di dialogo per una applet firmata

Nella dialog il plug-in mostra il nominativo della persona che ha firmato l'applet, le informazioni riguardanti l'attendibilitá della CA e le informazioni sulla verifica del certificato del firmatario. L'utente ha a disposizione quattro differenti opzioni:

- **Concedi per questa sessione:** se viene selezionato questo bottone, l'applet riceve *AllPermission*. Ogni altra applet, firmata con il medesimo certificato, che sará caricata durante questa sessione del browser, riceverá automaticamente gli stessi privilegi.
- **Rifiuta:** in questo caso l'applet sará considerata come insicura e quindi alla sua esecuzione verrá applicato il modello di sicurezza della sand-box.
- **Concedi sempre:** l'applet firmata riceve anche in questo caso *AllPermission*. Ogni altra applet firmata con lo stesso certificato sará considerata come sicura e la finestra di dialogo non verrá piú visualizzata. Il certificato del firmatario é inserito nella lista dei certificati presenti all'interno del pannello di controllo del Java plug-in. L'utente ha comunque la possibilitá di eliminare tale certificato dalla lista, e di conseguenza di rivisualizzare la dialog di sicurezza.
- **Visualizza certificato:** se selezionato, permette all'utente di visualizzare gli attributi di ciascun certificato presente nella catena dei certificati contenuti nell'archivio jar.

A seconda della scelta effettuata, l'applet sará eseguita nel rispettivo contesto di sicurezza. É bene sottolineare che questo tipo di decisione, non richiede alcun tipo di configurazione preliminare lasciando completa libertá all'utente. Dal punto di vista della sicurezza, questa situazione non risulta essere sicuramente la migliore, in quanto in ben due casi l'applet riceve completi privilegi. Dato che l'ambiente Java integra altri strumenti (soprattutto le policy) capaci di limitare allo stretto indispensabile le operazioni concesse al codice scaricato dalla rete, é consigliabile utilizzare anche questi tool. Il Java plug-in prevede anche la possibilitá di disabilitare l'esecuzione delle applet firmate RSA, specificando un nuovo permesso, denominato *usePolicy* nel file della policy. Se ad un codice eseguibile é concesso solamente questo permesso, la dialog di sicurezza non sará abilitata e l'applet non verrá inizializzato.

2.4.1 Utilizzo dell'applicazione di esempio

L'applicazione sviluppata é composta da una Graphical User Interface che permette all'utente di accedere alle funzionalitá esportate. La figura 2.2 mostra la GUI caricata dall'applet.

Figura 2.2: Graphical User Interface dell'applet

L'operazione di cifratura avviene secondo il seguente schema:

1. Selezione del file contenente il documento che si intende crittografare. É possibile inserire direttamente nella casella di testo il percorso del file, oppure ricercare il file nel file system attraverso l'apposito bottone.

2. Selezione della modalitá di cifratura tra una di quelle proposte.

3. Selezione dei certificati dei destinatari ai quali il documento cifrato deve essere inviato.

4. Avvio delle operazioni di cifratura attraverso un click sull'apposito bottone.

5. Quando il processo di cifratura termina viene visualizzata una finestra che permette di memorizzare il documento crittografato in un file.

L'operazione di decifratura richiede l'esecuzione dei seguenti passi:

1. Selezione del file contenente il documento crittografato che si intende decifrare.

2. Selezione della modalitá di decifratura.

3. Avvio dell'operazione di decifratura attraverso un click sull'apposito bottone.

4. Selezione del file (.p12) contenente la chiave privata dell'utente, necessaria per la decifratura del documento. Per accedere alla chiave privata, é necessario inserire la corretta password.

5. Se la decifratura avviene con successo, il contenuto decifrato del messaggio é visualizzato in un'apposita dialog.

Capitolo 3

Conclusione e sviluppi futuri

Il modello di sicurezza proposto da Java é sicuramente molto complesso e ricco di caratteristiche che, per essere comprese ed utilizzate in modo realmente sicuro, richiedono competenze ben specifiche. Al tempo stesso, l'architettura per codice mobile proposta dall'ambiente Java, é senza ombra di dubbio la piú completa e affidabile presente sul mercato. Questo non significa che Java garantisce la sicurezza assoluta, cosa che é praticamente impossibile da raggiungere ma che é in grado di limitare i rischi che si accettano quando si collega una macchina alla rete.

Nella sezione 1.5 del primo capitolo sono state confrontate le diverse filosofie che hanno ispirato i modelli di sicurezza su cui si basano i **Controlli ActiveX** e le **Java Applet** e si é dimostrato come il meccanismo di firma del codice, non puó in nessun modo essere considerato uno strumento di protezione, ma puramente una questione di fiducia: una volta accettato il certificato dello sviluppatore, il codice ha completo e libero accesso alle risorse della macchina locale. La firma consente infatti di verificare:

- Integritá
- Autenticitá

ma non protegge da eventuali errori di programmazione e/o operazioni illecite. La specifica di policy in grado di concedere solo a determinate applet privilegi aggiuntivi, consente di superare le restrizioni imposte dalla sand-box e al tempo stesso di limitare l'applet ad eseguire solo le operazioni espressamente concesse. Sicuramente il processo di definizione delle policy non é un'operazione semplice in quanto richiede una conoscenza approfondita dell'ambiente Java. I concorrenti della Sun, Microsoft su tutti, ma anche molti sviluppatori, hanno criticato questo modello di sicurezza ma, questa soluzione risulta essere l'unico modo

per non concedere ad un codice scaricato dalla rete, *AllPermission*, basandosi esclusivamente sul meccanismo dei certificati e della firma digitale.

Lo sviluppo di pagine dal contenuto eseguibile, richiede quindi dal punto di vista della sicurezza, un notevole sforzo progettuale ed implementativo poiché, come descritto, comporta vantaggi e rischi notevoli.

Bibliografia

[1] G.McGraw, E.Felten *Securing JAVA*

[2] http://developer.netscape.com/software/signedobj/jarpack.html

[3] http://telemat.die.unifi.it/book/Networking_os/ActiveX/Controlli_ActiveX/Welcome.html

[4] http://java.sun.com

[5] http://java.sun.com/products/plugin/1.3/docs/nsobjsigning.html

[6] http://www.rsasecurity.com/rsalabs/pkcs/pkcs-12/

[7] http://pkp-ca.iat.cnr.it/