BEVEILIGINGSRISICO'S VAN JAVA APPLETS

Jan van den Berg
Mark Kilsdonk
Ed Ridderbeekx

1. INLEIDING

1.1 Algemeen

Een organisatie kan op vele mogelijke manieren profiteren van het Internet. De aanwezigheid van nieuwe, door het Internet gefaciliteerde, communicatiediensten biedt mogelijkheden om bepaalde bedrijfsprocessen verder te stroomlijnen. Daarbij moet de organisatie zich er wel rekenschap geven dat het gebruik van Internet bepaalde beveiligingsrisico's met zich meebrengt. In een eerdere bijdrage [ribe_98] is een model geïntroduceerd dat gebruikt kan worden om deze risico's inzichtelijk te maken en te systematiseren, waarna vervolgens een afgewogen verzameling maatregelen kan worden gedefinieerd om deze risico's terug te brengen tot op het door de organisatie gewenste niveau. In dit artikel zal dit model toegepast worden om de beveiligingsimplicaties van het gebruik van Java-applets te analyseren.

In het vervolg van hoofdstuk 1 wordt ingegaan op enkele karakteristieken van Java. In hoofdstuk 2 zal aandacht worden besteed aan het uitvoeringstraject van een Java-applet. Het beveiligingsmodel van Java wordt in hoofdstuk 3 toegelicht, waarna in hoofdstuk 4 de risico's van Java worden besproken. De maatregelen die genomen kunnen worden om deze risico's te mitigeren komen in hoofdstuk 5 aan de orde. In aparte kaders is een tweetal *cases* opgenomen, die een indruk beogen te geven van het mogelijk gebruik van Java-applets en de verschillende niveaus van beveiliging die van toepassing zijn bij dit gebruik.

1.2 Java

In één van de eerste introducties door Sun Microsystems werd Java aangekondigd als [graw_96]:

***Java**: A simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded and dynamic language.*

Java is een programmeertaal, die qua ideeën en basis-syntax sterk lijkt op C++. Net als C++ is Java een object-georiënteerde programmeertaal. Java wordt vaak gezien als een gebruiksvriendelijke versie van C++. Er is echter een aantal essentiële verschillen. De twee belangrijkste verschillen worden in de navolgende paragrafen kort toegelicht. Het gaat daarbij om platform-onafhankelijkheid en netwerkmobiliteit.

1.3 Platformonafhankelijkheid

Een van de belangrijkste verschillen met C++ is Java's platform-onafhankelijkheid. Deze eigenschap wordt ook wel omschreven met de karakteristiek "write once, run anywhere". Bij het programmeren in Java hoeft niet, zoals bij C++, al bij compilatie een keuze te worden gemaakt voor een bepaald besturingssysteem of hardware-platform waarop de gecompileerde programmatuur zal draaien. Java-broncode kan op ieder platform worden gecompileerd en gecompileerde Java-code draait zonder wijziging op ieder platform. Het is duidelijk dat dit een enorme besparing kan betekenen in de kosten van ontwikkeling en beheer van programmatuur die bedoeld is om op verschillende machines en besturingssystemen te kunnen functioneren. Deze platformonafhankelijkheid wordt bereikt door:

- een universele *application programming interface* (API);
- een universele specificatie van de Java Virtual Machine (JVM).

Java API
De Java API is een consistente, machine- en platformonafhankelijke specificatie van de "bouwstenen" waarmee een Java-programma kan worden gecreëerd. De Java API bestaat onder andere uit:

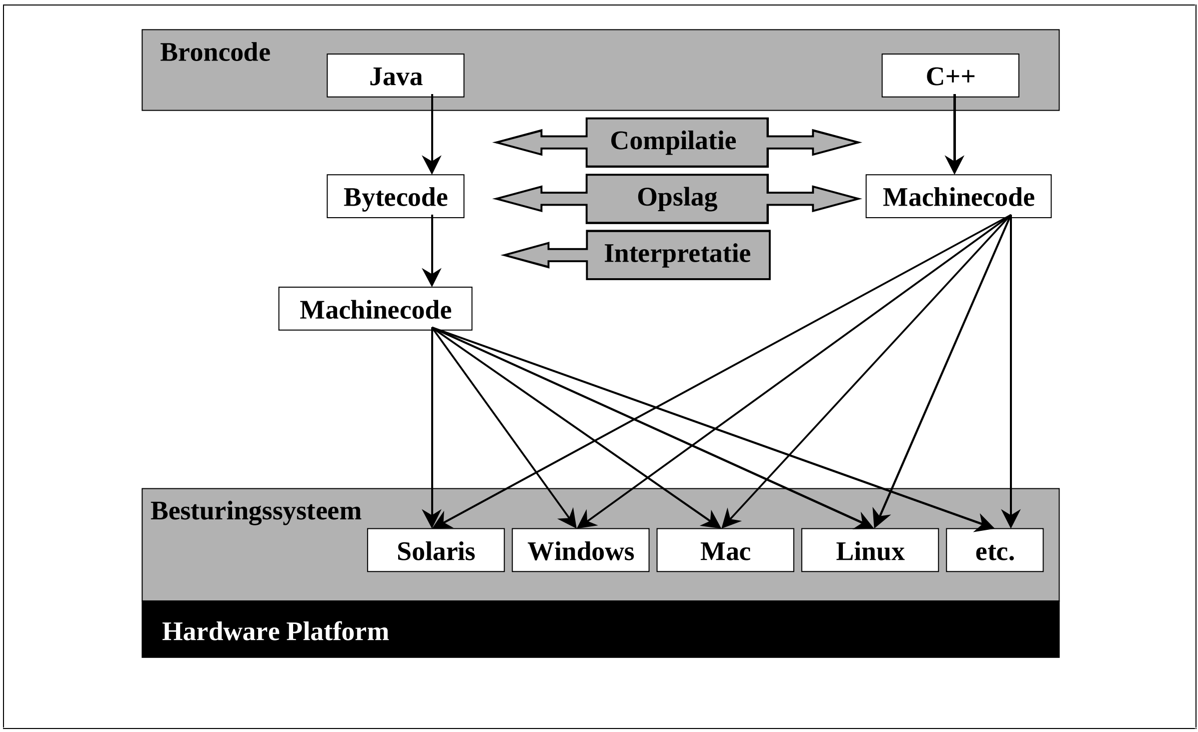
- de Java-taal zelf;
- Java-utilities
- I/O en netwerkfunctionaliteiten;
- ondersteuning voor het omgaan met windows en applets;
- security-functionaliteiten.

Deze API wordt geïmplementeerd door software libraries, gebundeld in de zogenaamde *Java Developers Kit* (JDK). Ontwerpers die hiervan gebruik maken hebben zekerheid over het feit dat de door hun gebouwde applets op ieder platform dat Java ondersteunt zullen kunnen draaien.

Java Virtual Machine
Gewoonlijk zetten compilers de broncode van een programmeertaal om naar instructies die specifiek zijn afgestemd op de eisen van de processor van de machine waarop het programma uitgevoerd moet worden. Om redenen van machine-onafhankelijkheid gaat de Java-compiler hiermee anders om. Hij transformeert de Java-broncode naar een *bytecode*-formaat ofwel *J-code*. Deze bytecode, die wordt opgeslagen in een CLASS-bestand, moet worden gezien als de machinetaal van een softwarematige computer: de Java Virtual Machine (JVM). De JVM omvat onder andere een run-time interpreter, die de bytecode omzet naar de machinetaal van de onderliggende processor. Dit kan eveneens plaatsvinden door de zogenaamde JIT (*just-in-time*)-compiler. Deze interpreter en JIT-compiler zijn derhalve *wel* machine-afhankelijk.

1.4 Netwerkmobiliteit

De netwerkmobiliteit van Java kan worden toegelicht aan de hand van het client-server paradigma. Onder de server wordt het proces verstaan dat een service via een netwerk ter beschikking stelt; een client is het proces dat het verzoek om een service via het netwerk aan een server verstuurt en op een respons wacht.
Er zijn twee typen Java-programma's: applicaties en applets. Java-applicaties zijn conventionele programma's en worden derhalve uitgevoerd op dezelfde locatie als waar ze zijn opgeslagen (hetzij aan de client-, hetzij aan de serverzijde). Een Java-applet daarentegen is een applicatie die op de machine aan de serverzijde opgeslagen ligt, en op verzoek van een client via het netwerk naar de clientzijde wordt getransporteerd en direct op de machine aan de clientzijde wordt uitgevoerd. Deze uitvoering (*execution*) van een Java-applet aan de clientzijde wordt verzorgd door een implementatie van de JVM, zoals die bijvoorbeeld voorhanden is in een Java-enabled webbrowser als de Microsoft Internet Explorer of Netscape's Communicator.
In het algemeen wordt een Java-applet aangeduid als *executable content*. Executable content is een uitvoerbaar programma dat door de server wordt verstuurd naar de client. Dit programma wordt vervolgens door de machine aan de clientzijde uitgevoerd. Java is momenteel veruit de meest populaire implementatie van executable content. Concurrenten van Java zijn JavaScript, Safe-TCL, Telescript, Postscript en het opkomende ActiveX van Microsoft.



Figuur 1. Ontwikkeling van broncode tot uitvoering van Java en C++

2. UITVOERINGSTRAJECT

De kenmerkende netwerkmobiliteit van een Java-applet kan nader worden beschreven aan de hand van een uitvoeringstraject. Dit uitvoeringstraject vormt een preciezere uitwerking van het eerder behandelde client/servermodel in combinatie met Java-applets, waarbij we uitgaan van de volgende punten:

- de server is een webserver;
- de client is een Java-enabled webbrowser;
- de aanroep van het applet is ingebed in een op aan de serverzijde opgeslagen HTML-pagina, waaraan door de client wordt gerefereerd.

De zeven stappen in het uitvoeringstraject van een Java-applet zijn dan:

1. ontwikkeling van het applet
De eerste stap in het uitvoeringstraject van een Java-applet is de ontwikkeling van het Java-applet zelf en een web-pagina die dit applet aanroept. De ontwikkeling van het Java-applet vindt plaats door het schrijven van de broncode en vervolgens de compilatie van deze broncode naar een CLASS-bestand.

2. opslag aan de serverzijde
Vervolgens worden het applet (ofwel het CLASS-bestand) en de webpagina met applet-aanroep opgeslagen op een machine aan de serverzijde. De webserver van deze machine kan de bestanden over het Internet distribueren naar verschillende clients.

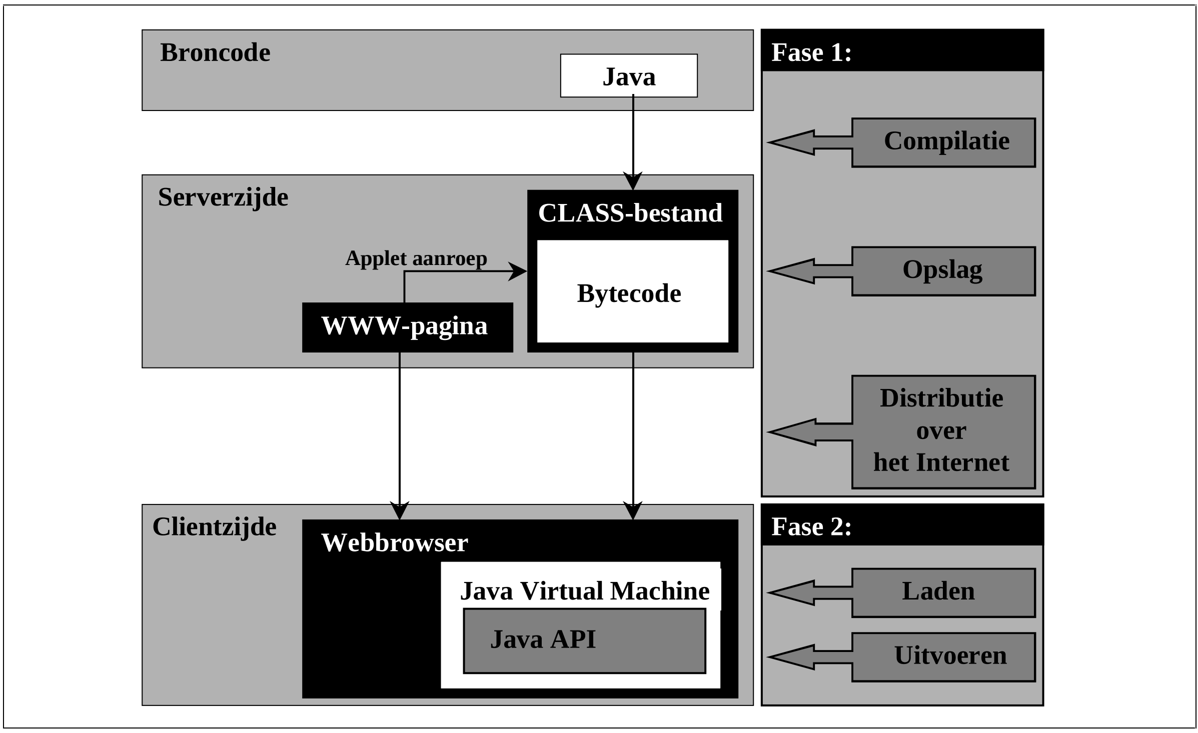
3. opvragen applet
De volgende stap in het uitvoeringstraject van een Java-applet vindt plaats op het moment dat een gebruiker aan de clientzijde de webpagina opvraagt. De Java-enabled webbrowser waarmee deze gebruiker werkt, verzoekt de met deze webpagina corresponderende webserver de pagina te verzenden.

4. versturen webpagina
De webserver ontvangt dit verzoek en verstuurt de webpagina naar de machine aan de clientzijde.

5. opvragen applet
De Java-enabled webbrowser aan de clientzijde ontvangt de webpagina en zal deze pagina laden. Bij het laden van de webpagina herkent de webbrowser de applet-aanroep in de HTML-code. De webbrowser zal de webserver verzoeken het betreffende Java-applet te versturen. Na het laden van de webpagina zal de webbrowser deze pagina tonen aan de gebruiker.

6. versturen applet
De webserver ontvangt het verzoek en verstuurt het Java-applet (het CLASS-bestand) naar de machine aan de clientzijde.

7. uitvoeren applet
De webbrowser aan de clientzijde ontvangt het Java-applet. De JVM laadt het Java-applet en voert het vervolgens uit.



Figuur 2. Het uitvoeringstraject van een Java-applet

In figuur 2 is het uitvoeringstraject van een Java-applet opgedeeld in twee fasen. De eerste fase beslaat de ontwikkeling van de broncode tot en met het versturen van het Java-applet naar een machine aan de clientzijde. Alle facetten die worden uitgevoerd aan de clientzijde, de ontvangst, het laden en het uitvoeren van het Java-applet, vormen de tweede fase van het traject van een Java-applet.

3. JAVA-BEVEILIGINGSMODEL

Het gebruik van executable content in het algemeen en Java-applets in het bijzonder kan beveiligingsrisico's met zich meebrengen. In het algemeen is dit het gevolg van het feit dat het applet gecreëerd is in een andere omgeving dan die waar het applet wordt uitgevoerd. Aan de clientzijde bestaat er derhalve veelal geen precies inzicht in wat het applet aan functionaliteit bezit. Het is heel wel denkbaar dat die functionaliteit de clientzijde negatief beïnvloedt. De risico's van een Java-applet lijken daarmee in grote mate op de risico's van een computervirus. Verderop in dit artikel zullen deze risico's nauwkeuriger worden benoemd. Op dit moment volstaan we met de opmerking, dat ook Sun Microsystems zich bij de ontwikkeling van Java bewust is geweest van de beveiligingsrisico's.
Om deze te beperken is een beveiligingsmodel ontwikkeld, dat kan worden gezien als een verzameling technische beveiligingsmaatregelen in de taal en in de *runtime*-omgeving. Het Java-beveiligingsmodel moet voor en tijdens uitvoering van een Java-applet de machine aan de clientzijde (en daarmee in feite de organisatie die van applets gebruik maakt) beveiligen tegen de risico's van vijandige Java-applets.
Het Java-beveiligingsmodel specificeert een zogenaamde *sandbox* om een Java-applet. Een sandbox is als het ware een virtuele muur om een Java-applet in uitvoering. De sandbox moet voorkomen dat een Java-applet tijdens zijn uitvoering activiteiten verricht buiten de grenzen van de sandbox. Deze grenzen worden gedefinieerd door de beveiligingsmaatregelen in het Java-beveiligingsmodel. De beveiliging aan de clientzijde is hiermee afhankelijk geworden van de kracht van de beveiliging van de sandbox. Eventuele "lekken" in de sandbox doorbreken deze beveiliging. Om te verzekeren dat deze lekken zich niet voordoen, heeft Sun destijds het Java-beveiligingsmodel zodanig breed opgezet dat het elk aspect van de Java-architectuur beïnvloedt.

De fundamentele onderdelen voor Java's Sandbox zijn:

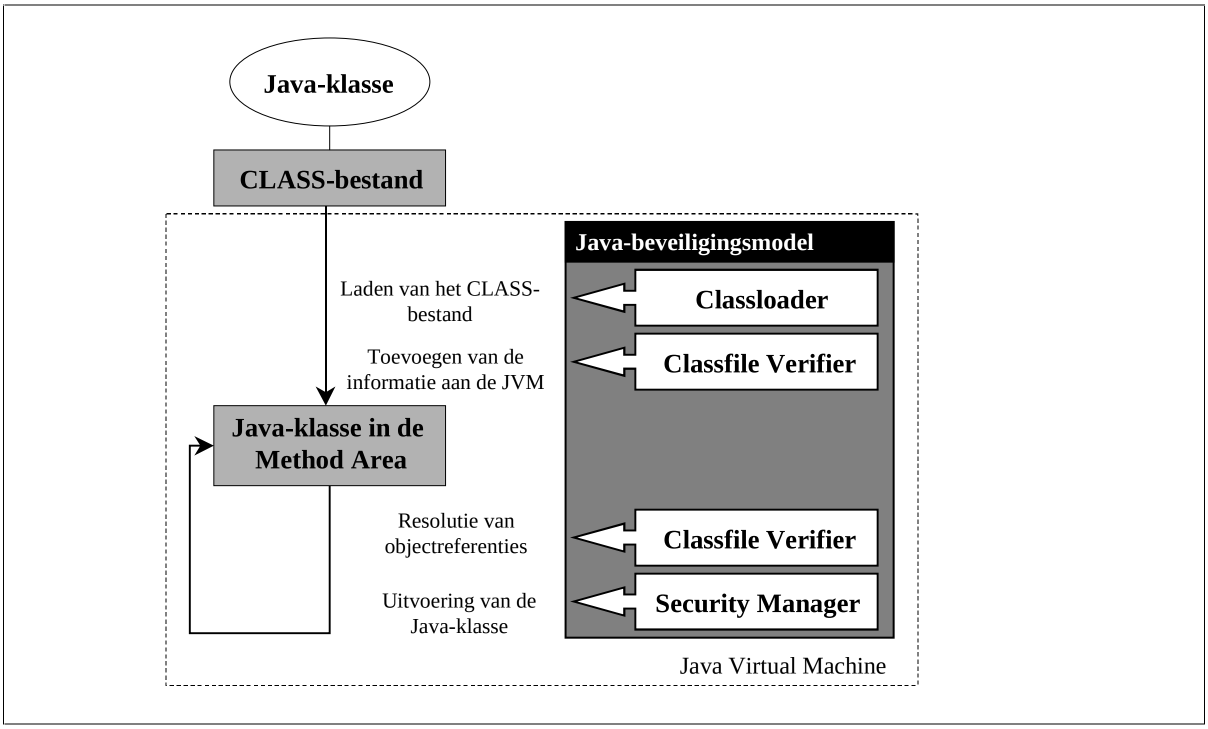
Fase 1 (serverzijde):
De beveiligingsmaatregelen van de programmeertaal Java.

Fase 2 (clientzijde):
De Classloader.
De Classfile Verifier.
De Security Manager.

De bovenstaande onderdelen zijn verdeeld naar de twee fasen van het uitvoeringstraject. Het eerste onderdeel, de beveiligingsmaatregelen van de programmeertaal Java, wordt geïmplementeerd door de Java-compiler. Bij compilatie van de broncode controleert de Java-compiler de broncode op foutieve constructies die een risico kunnen vormen bij uitvoering. De controle op deze beveiligingsmaatregelen vindt daarmee plaats aan de serverzijde.

De implementatie van de overige beveiligingsmaatregelen wordt aan de clientzijde uitgevoerd. De onderdelen, die deze controles uitvoeren, worden op deze plaats kort toegelicht aan de hand van uitvoering van een Java-applet. Voor een uitgebreidere beschrijving van deze onderdelen wordt verwezen naar [graw_96] en [kils_97].

Na de ontvangst van een Java-applet in de vorm van een CLASS-bestand aan de clientzijde, wordt de informatie in dit bestand door de Classloader geladen. De Classloader zal, tijdens het laden, het Java-applet toewijzen aan een geheugengebied waar het applet geen invloed kan uitoefenen op andere applets of het besturingssysteem van de machine aan de clientzijde. Nadat het applet geladen is wordt het door middel van *linking* aan de JVM toegevoegd. *Linking* is het proces van het toevoegen van de informatie in een CLASS-bestand aan de JVM. Bij deze toevoeging zal de Classfile Verifier controleren of de informatie in een dergelijk bestand juist is qua inhoud en structuur. Laden en toevoegen van de informatie in een CLASS-bestand zijn twee stappen die direct na elkaar plaats vinden en dientengevolge voeren de Classloader en de Classfile Verifier hun werk ook direct na elkaar uit. Tenslotte controleert de Security Manager tijdens uitvoering de operaties van een Java-applet. De Security Manager beschermt de systeem-resources aan de clientzijde tegen misbruik tijdens de uitvoering van een Java-applet. De Security Manager bewaakt de grenzen van de Sandbox.



Figuur 3. De beveiligingsmaatregelen aan de clientzijde

Het Java-beveiligingsmodel is grotendeels afhankelijk van een correcte implementatie van de JVM aan de clientzijde. De eis van een correcte implementatie van de JVM wordt gecompliceerd door het feit dat de rechten om een JVM te implementeren en te distribueren zijn verleend aan verscheidene bedrijven (o.a. Microsoft, Netscape, Symantec en Borland). De praktijk laat zien (zie paragraaf 4.4) dat veel implementaties fouten vertonen en op die wijze op verschillende plaatsen gebreken in de beveiliging creëren.

Case 1. Het Java-beveiligingsmodel is (bijna) toereikend.
Een adviesbureau besluit na overleg voor al zijn medewerkers vanaf het hoofdkantoor toegang tot het Internet te realiseren. Het gaat hierbij met name om de adviseurs mogelijkheden te bieden tot het verzenden en ontvangen van e-mail, maar eveneens om het raadplegen van relevante nieuwsgroepen. Voornaamste reden is echter de toenemende wens van medewerkers om algemeen vaktechnische en beroepsgerelateerde informatie op het World Wide Web te kunnen benaderen.
Het bureau realiseert een vaste verbinding tussen zijn bestaande interne netwerk en het Internet via een Internet Service Provider. Een firewall moet afdwingen dat onbevoegden zich geen toegang kunnen verschaffen tot het interne netwerk van het bureau. In de discussies rondom de beveiliging komen

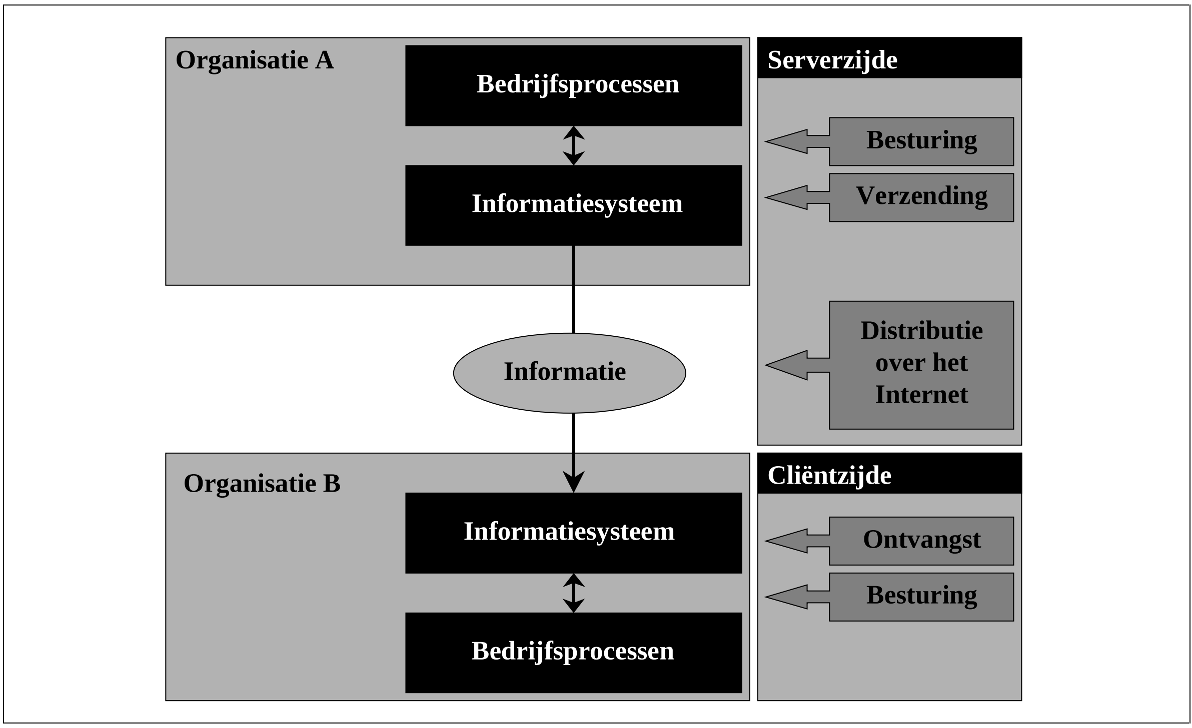
op een gegeven moment *executable content* en in het bijzonder Java-applets aan de orde. Het bedrijf heeft enkele vervelende ervaringen met virussen achter de rug en pleit in eerste instantie voor blokkering van deze applets door de firewall. Na advies ingewonnen te hebben over zowel de voordelen als de risico's van applets besluit men echter dat het sandbox-beveiligingsmodel van Java toereikend is voor het soort gebruik dat van Internet wordt gemaakt. Wel wordt besloten de browsers van medewerkers initieel zo te configureren, dat applets niet door de browser worden uitgevoerd. Bovendien wordt in een algemene brochure gewezen op de beveiligingsrisico's van het werken met het world wide web.

4. RISICO-ANALYSE

Ten einde de risico's van het gebruik van Java-applets te analyseren, wordt in dit hoofdstuk het Internet Risico Model gepresenteerd [ribe_98]. In het licht van het uitvoeringstraject van Java-applets wordt dit risicomodel nader gespecificeerd tot het Java-applets Risico Model. Aan de hand van dit laatste model kan het standaard Java-beveiligingsmodel uit hoofdstuk 3 getoetst worden. Vervolgens zal - gebruikmakend van enige voorbeelden uit de praktijk - getracht worden het beeld van de risico's, die worden gelopen ten gevolge van het toepassen van Java-applets, compleet te maken.

4.1 Internet Risico Model

Informatie-uitwisseling tussen organisaties via het Internet kan op betrekkelijk eenvoudige wijze worden gemodelleerd. Daartoe onderscheiden we binnen elke organisatie enerzijds de (primaire en secundaire) bedrijfsprocessen en anderzijds het ondersteunende informatiesysteem. Voorts veronderstellen we dat de informatiesystemen van de diverse organisaties onderling gegevens(bestanden) uitwisselen via een extern netwerk, hier het Internet.



Figuur 4. Communicatiemodel

Het informatiesysteem van een organisatie kan worden gedefinieerd als *het geheel van methoden, faciliteiten en activiteiten waarmee deze organisatie in haar informatiebehoeften tracht te voor-*

zien [bemel_94]. Als we deze definitie toepassen op de geautomatiseerde informatiesystemen, kunnen we deze systemen opgebouwd denken uit een drietal componenten zijnde *hardware* (bestaande uit computerapparatuur inclusief computernetwerken), *software* (programmatuur en data) en *gebruik* (alle menselijke handelingen m.b.t. de bediening van het informatiesysteem). Hierbij zij aangetekend dat in de situaties die wij beschouwen het externe netwerk (het Internet) niet binnen de directe invloedssfeer en beheersverantwoordelijkheid ligt van de onderscheiden organisaties. Het neemt hierdoor een bijzondere positie in. De funktie van het Internet is ook beperkt: het is in essentie louter een faciliteit waarmee wereldwijd multimediale gegevens (in ons geval Java-applets en de daarmee samenhangende informatie) kunnen worden uitgewisseld. Om deze redenen zal hier het externe netwerk steeds apart worden genoemd en onderscheiden worden van de overige, interne componenten van een bedrijfsinformatiesysteem. Ook gaan we stilzwijgend uit van een permanente beschikbaarheid van het WWW.
Het Internet Risico Model [ribe_98] (IRM) werd geïntroduceerd met als doel in staat te zijn om op systematische wijze alle *extra* risico's te analyseren die het gebruik van het Internet voor een organisatie met zich meebrengt. Op grond van de zojuist genoemde uitgangspunten kunnen deze risico's worden gecategoriseerd in een tweetal typen te weten:

(A) de negatieve beïnvloeding van de kwaliteit van via het Internet uitgewisselde informatie.
(B) de negatieve beïnvloeding van het interne informatiesysteem van de organisatie.

Bij categorie (A) gaat het in essentie om bedreigingen rond:
- vertrouwelijkheid: data moeten veelal ontoegankelijk zijn voor derden,
- integriteit: data moeten niet gewijzigd kunnen worden door derden,
- authenticiteit: de herkomst van data moet ondubbelzinnig vastgesteld kunnen worden,
- onweerlegbaarheid: de verzender mag niet in staat zijn te ontkennen daadwerkelijk door hem verstuurde data verzonden te hebben.

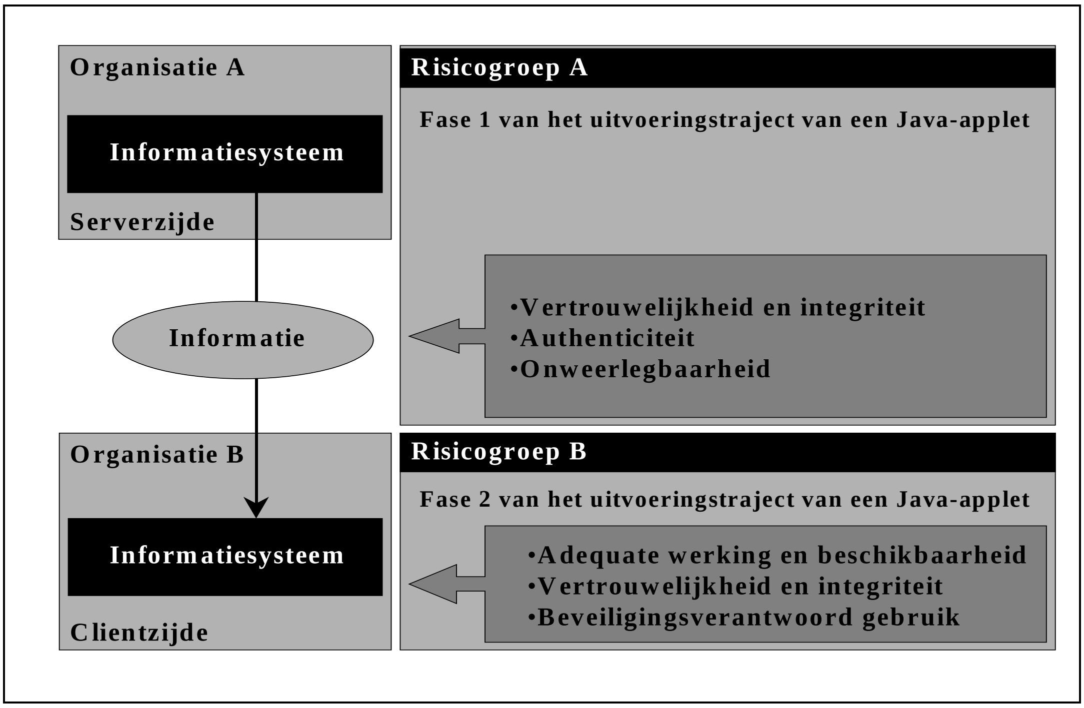
Bij categorie (B) gaat het in essentie gaat het om bedreigingen met betrekking tot:
- hardware en software: het interne automatiseringssysteem moet onverminderd beschikbaar zijn voor geautoriseerde gebruikers en adequaat werken,
- software: de vertrouwelijkheid en integriteit[1] van de aanwezige programmatuur en de opgeslagen gegevens moeten gewaarborgd zijn,
- verantwoord gebruik: gebruikers dienen op beveiligingsverantwoorde wijze met het informatiesysteem om te gaan.

Op hun beurt houden deze negatieve bedreigingen zekere gevaren in voor de binnen de organisatie operationele bedrijfsprocessen. Hoe met deze gevaren om te gaan zal echter pas in hoofdstuk 5 nader worden beschouwd.

4.2 Java-applets Risico Model

Omdat het in dit artikel gaat om een analyse van de risico's ten gevolge van het gebruik van Java-applets, zullen we de hiervoor onderscheiden risicotypen (A) en (B) beschouwen aan de hand van de twee fasen van het uitvoeringstraject van een applet (zie hoofdstuk 2). Dit levert het volgende Java-applets Risico Model (JRM):

[1] Een impliciete veronderstelling is dat de vertrouwelijkheid en integriteit van de hardware niet bedreigd (kunnen) worden.



Figuur 5. Het Java-applets Risico Model

In termen van het uitvoeringstraject van een Java-applet (zie figuur 2) vindt fase 1 zowel plaats binnen het informatiesysteem van organisatie A als tijdens de distributie ervan over het Internet. Fase 2 van het uitvoeringstraject vindt plaats binnen het informatiesysteem van organisatie B. Als we veronderstellen dat het betreffende Java-applet een correct, foutvrij programma is dat ook correct is geïnstalleerd bij de distribuerende organisatie, dan kunnen de risico's van het gebruik van Java-applets als volgt worden samengevat:

> De risico's van het gebruik van Java-applets vallen onder categorie (A) wat betreft de distributie ervan over het Internet en onder categorie (B) wat betreft de uitvoering ervan aan zowel server- als clientzijde.

4.3 Toetsing van het standaard Java-beveiligingsmodel

Nu de potentiële risico's van Java-applets in kaart gebracht zijn, kunnen we vervolgens proberen in detail te analyseren in hoeverre het standaard Java-beveiligingsmodel deze risico's afdekt. Hierbij zullen we alleen kijken naar de fouten en/of gebreken in de specificatie van het Java-beveiligingsmodel. Fouten in de implementatie vallen - logischerwijs - buiten de invloedssfeer van dit model vallen[2].

4.3.1 Bedreigingen tijdens distributie over het Internet

Het Java-beveiligingsmodel voorziet niet in de waarborging van de vertrouwelijkheid van een Java-applet als passieve informatie (het class-bestand). Een dergelijk bestand wordt blijkbaar geacht geen gevoelige informatie te bevatten. Ook de informatie, die een Java-

[2] Praktijkvoorbeelden (zie b.v. par. 4.4 en [comp_98]) laten overigens zien dat juist implementatiefouten ten grondslag liggen aan veel security-problemen. Helaas is dit type fouten moeilijk algemeen analyseerbaar.

applet eventueel zal terugsturen naar de webserver vanwaar het applet is geladen, is niet beveiligd tegen kennisname door derden. Vertrouwelijkheid zal dus – indien gewenst - op andere wijze moeten worden gegarandeerd, bijvoorbeeld door toepassing van cryptografische technieken voorafgaand aan verzending van het applet: de JDK biedt daartoe mogelijkheden.

Na het laden van het applet aan de clientzijde zal de Classfile Verifier controleren of het CLASS-bestand voldoet aan de indeling die een Java-compiler creëert bij de opslag van een dergelijk bestand. Daarnaast zal de JVM tijdens uitvoering controleren of voldaan is aan de beveiligingsmaatregelen van Java zoals *type safety* en gestructureerde geheugentoegang. Er zijn echter in het beveiligingsmodel geen specifieke maatregelen genomen om de integriteit van een applet te waarborgen. Dit zal dus desgewenst ook op andere wijze moeten worden gegarandeerd, bijvoorbeeld door toepassing van cryptografische technieken.

Door het ophalen en laden van een WWW-pagina met een Java-applet geeft een gebruiker toestemming om dit applet te laden en uit te voeren. Het applet wordt hierdoor een geautoriseerd proces binnen het informatiesysteem aan de client-zijde. De JDK geeft door middel van digitale handtekeningen de ontwikkelaar van het applet de mogelijkheid om zijn identiteit aan een Java-applet mee te geven en daarmee de authenticiteit te waarborgen. Een Java-applet zonder digitale handtekening wordt door het Java-beveiligingsmodel onbetrouwbaar geacht en wordt op grond hiervan (terecht) slechts geautoriseerd tot een beperkte verzameling van operaties.

Een gebruiker aan de client-zijde heeft binnen het huidige Java-beveiligingsmodel geen mogelijkheden om de herkomst van een Java-applet ondubbelzinnig vast te stellen. De onweerlegbaarheid van binnengekomen Java-applets (hier beschouwd als passieve informatie) kan niet zonder meer worden vastgesteld.

4.3.2 *Bedreigingen voor het informatiesysteem*

De uitvoering van een Java-applet leidt altijd tot het gebruik van *system resources* van het informatiesysteem aan de client-zijde. Het Java-beveiligingsmodel moet voorkomen dat een Java-applet als geautoriseerde gebruiker ongeautoriseerde activiteiten uitvoert. De Security Manager van het Java-beveiligingsmodel beperkt de toegang tot de resources. Zo heeft een Java-applet in uitvoering geen toegang tot bestanden op de lokale schijf (bestanden kunnen noch aangemaakt, noch gewijzigd, noch verwijderd worden). Ook systeeminstellingen kunnen niet gewijzigd worden, noch kunnen applicaties gestart of beëindigd worden. Ook de interne geheugenlokaties van andere programma's zijn beveiligd. Echter, ten aanzien van een aantal componenten zijn de beveiligingsmaatregelen beperkt. De voorbeelden uit paragraaf 4.4 zullen tonen dat processortijd, geheugenverbruik en het opzetten van netwerkconnecties op vrijwel onbeperkte schaal kunnen plaatshebben, hetgeen de beschikbaarheid en adequate werking van het informatiesysteem mogelijkerwijs in gevaar brengt.

Vertrouwelijkheid en integriteit van het informatiesysteem betreft de op het systeem opgeslagen programma's en data. De hierboven genoemde werkzaamheden van de Security Manager zijn hier van belang. Daarnaast gebruikt de JVM voor de opslag van applets en voor gecreëerde objecten een apart geheugengebied. Er lijkt (nogmaals, in termen van de specificatie van het beveiligingsmodel) geen enkele mogelijkheid te bestaan dat door het uitvoering van het applet per abuis andere geheugenplaatsen worden gelezen of overschreven. Vertrouwelijkheid respectievelijk integriteit van het informatiesysteem zijn dus gewaarborgd.

Het Java-beveiligingsmodel bestaat louter uit een verzameling van technische maatregelen en kan dus per definitie niet voorzien in de waarborging van een beveiligingsverantwoord gebruik. Dat dit risico's met zich meebrengt blijkt mede uit de volgende voorbeelden.

Case 2. Het Java-beveiligingsmodel vereist additionele beveiligingsmaatregelen.
Een industriële onderneming maakt intensief gebruik van een geïntegreerd financieel en logistiek softwarepakket, dat operationeel is op een cluster minicomputers. De leverancier van de software, die voor meerdere hardware-platforms op de markt wordt gezet, brengt regelmatig nieuwe releases van de software uit; bovendien is men recent gestart met de mogelijkheid om tussentijdse software-patches te *downloaden* van de web-site van de leverancier. Gelijktijdig met de nieuwe software wordt daarbij tevens een aantal Java-applets naar de client-omgeving getransporteerd, die de nodige intelligentie leveren om de nieuwe software correct te installeren en die bovendien een aantal controles doen op de integriteit van de software-installatie.
De industriële onderneming wil graag gebruik maken van deze functionaliteit, maar besluit dit pas te doen als men met zekerheid kan vaststellen dat de applets afkomstig zijn van de betreffende software-leverancier. Deze eis wordt nog sterker als duidelijk wordt dat de betreffende applets -om hun taak te kunnen uitvoeren- lees- en schrijfbevoegdheid op het *file system* van de computers van de industriële onderneming nodig zullen hebben. Het probleem (waarop al geanticipeerd is door de software-leverancier) wordt opgelost door gebruik te maken van *signed applets*, waardoor op basis van een 'electronische handtekening' en een 'certificaat' de authenticiteit en integriteit van het applet zeker wordt gesteld. Dergelijke *trusted applets* krijgen rechten om lees- en schrijfacties op de computer van de industriële onderneming uit te voeren.

4.4 Enige voorbeelden uit de praktijk

Onderstaande voorbeelden vormen slechts een zeer beknopte selectie uit een grote verzameling van bekende praktijk-voorbeelden. Meer informatie over dit onderwerp kan worden gevonden in [kils_97] en in de daarbij opgesomde referenties.

We beginnen met enige tekortkomingen van het Java-beveiligingsmodel. De eerste categorie voorbeelden betreft risico's door fouten in de specificatie van het Java-beveiligingsmodel. Een Java-applet blijkt meerdere *threads* (parallelle functies binnen een programma) te kunnen opstarten zonder verplicht te zijn deze ook weer te stoppen. Op het Internet is een groot aantal Java-applets van dit type verschenen. Bovendien is het aantal threads dat kan worden opgestart niet begrensd tot een bepaald maximum. Duidelijk is dat hierdoor gemakkelijk een vijandig applet kan worden gemaakt dat bij uitvoering een bedreiging vormt voor de adequate werking of zelfs beschikbaarheid van het informatiesysteem, bijvoorbeeld doordat betreffende threads bij voortduring de CPU opeisen voor de uitvoering van tijdsintensieve instructies.

Er zijn ook allerlei voorbeelden bekend van implementatiefouten van het Java-beveiligingsmodel. Een aantal hiervan hebben betrekking op de mogelijkheden van wijziging van de gebruikte IP-adressen, waardoor netwerkconnecties met derden kunnen worden opgezet. Op deze wijze kunnen malafide personen waardevolle informatie in handen krijgen door te interfereren in de door de gebruiker en het applet gecreerde client-server-verbindingen. Ook dreigt gevaar voor de adequate werking en beschikbaarheid van computersystemen omdat niet-geautoriseerde toegang kan worden verkregen. Een andere bug betrof de naamgeving van bepaalde packages die bij het laden van de applet worden meegeladen van de locale schijf. Het bleek mogelijk om vijandige Java-applets niet-veilige klassen van de locale schijf te laten laden. Dit zijn slechts twee voorbeelden. Vele andere zijn bekend op het Internet. Betreffende implementatiefouten zijn veelal zeer technisch van aard.

Ook kunnen (vijandige) Java-applets worden ontworpen die de gebruiker verlokken tot risicovol gedrag. Een tamelijk botte is een applet dat de gebruiker vraagt om zijn gebruikersnaam en password (weer eens) in te vullen, hetgeen een argeloze gebruiker zonder meer zal doen. Een vijandige hacker heeft daarna ”vrij spel” in het interne informatiesysteem van deze gebruiker.

Het blijkt in een aantal gevallen ook mogelijk te zijn om veranderingen aan te brengen in de byte-code van het door de Java-compiler gegenereerde class-bestand. Virusachtige problemen zijn zelfs mogelijk doordat wijzigingen in de bytecode kunnen worden aangebracht bij de uitvoering van andere Java-programma's. Dit soort risico's zijn nog betrekkelijk recent van aard: de eerste voorbeelden stammen uit 1997. Gezien de relatief eenvoudige opbouw van class-bestanden, valt te verwachten dat in de toekomst meer voorbeelden van dit type bedreiging zullen worden ontdekt.

De hier beschreven voorbeelden laten zien dat de eerdergevonden tekortkomingen in het standaard Java-beveiligingsmodel niet louter theoretisch van aard zijn, maar dat er ten gevolge van die tekortkomingen daadwerkelijk security-problemen kunnen rijzen door het gebruik van Java-applets. Naast bedreigingen van structurele aard (ten gevolge van fouten in de specificatie van het beveiligingsmodel) kunnen fouten in de implementatie van Java-applets zeer specifieke, lastig te traceren bedreigingen mee zich mee brengen. Ook het onverantwoord handelen door gebruikers van Java-applets vereist speciale aandacht. Hoe met al deze bedreigingen om te gaan zal in de volgende sectie worden besproken.

5. MAATREGELEN

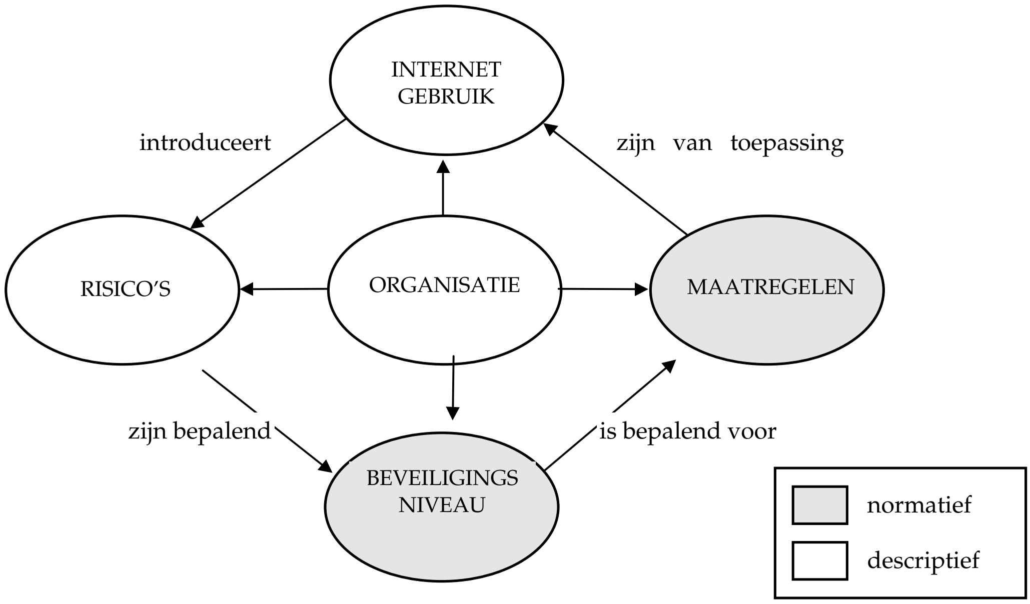
5.1. Inleiding

Uit het voorgaande is gebleken dat het standaard Java-beveiligingsmodel geen volledige beveiliging biedt tegen de risico's die gebruik van Java-applets met zich meebrengt. Een 100%-beveiliging is ook niet mogelijk. In dit verband is het dan ook beter te praten over een *gewenst beveiligingsniveau*, zijnde het niveau van beveiliging dat een organisatie nodig acht in de context van haar, door gebruik van Internet ondersteunde, bedrijfsprocessen. Alvorens maatregelen te kunnen vaststellen moet dan ook eerst worden vastgesteld wat het gewenste beveiligingsniveau is. Dit is – vanzelfsprekend – sterk afhankelijk van de toepassing en van de organisatie die het betreft: als een bank een nieuw product op de markt brengt inzake telebankieren via het WWW, zal ze veel hogere eisen aan de beveiliging van de bijbehorende applet-communicatie stellen dan twee personen zullen doen, die met behulp van eigengemaakt applet een spelletje schaak of iets dergelijks spelen onder gebruikmaking van het WWW.

Men zou kunnen zeggen dat het Java-beveiligingsmodel een standaard-beveiligingsniveau definieert. Als dit niveau in een concrete situatie onvoldoende geacht wordt, zijn additionele maatregelen gewenst. Daarbij is het van belang onderscheid te maken tussen verschillende typen maatregelen. Preventieve maatregelen beogen te voorkomen dat bepaalde risico's gelopen worden, terwijl repressieve maatregelen proberen de schade, die is opgelopen ten gevolge van de uitvoering van een vijandig Java-applet, zo snel mogelijk te detecteren, zoveel mogelijk te beperken en, indien mogelijk, ook te herstellen. Een ander relevant onderscheid (dat hierboven ook al aan de orde kwam) is dat tussen technische en organisatorische maatregelen. Alvorens concrete maatregelen van genoemde typen te bespreken, zullen we eerst een algemene methodiek bespreken voor de realisatie van een permanent verantwoord Internet-gebruik.

5.2 Dynamisch Structuurmodel

De betreffende methodiek wordt geformuleerd aan de hand van het zogenoemde Dynamisch Structuurmodel [ribe_98].



Figuur 6. Het Dynamisch Struktuurmodel

Centraal staat de organisatie die van het Internet gebruik maakt, bijvoorbeeld door de distributie of ontvangst van Java-applets. De organisatie onderkent dat er risico's verbonden zijn aan het toepassen van het Internet, die nauw samenhangen met de karakteristieken van haar bedrijfsprocessen. Op basis hiervan definieert zij een voor haar noodzakelijk en door haar gewenst beveiligingsniveau. Door uit te gaan van dit gewenste beveiligingsniveau kan worden bereikt dat de omvang en diepgang van beveiligingsmaatregelen zich op een goede manier verhouden tot de specifieke kenmerken van de organisatie en de risico's die ze loopt ten aanzien van Internetgebruik.

Deze abstractie is *dynamisch*: zodra in een van de elementen een verandering optreedt moet de cyclus opnieuw worden doorlopen. Veranderingen zijn in een Internet-context meer regel dan uitzondering, hetgeen betekent dat een organisatie die van Internet gebruik maakt bovenstaande cyclus wellicht veelvuldig zal dienen te doorlopen. Voor meer details over dit model zij verwezen naar [ribe_98].

5.3 Technische maatregelen

Zoals reeds vermeld kunnen de maatregelen die inherent zijn aan het Java beveiligingsmodel worden aangevuld met additionele maatregelen om het door de organisatie gewenste beveiligingsniveau te bereiken. In het onderstaande worden kort enkele mogelijke technische maatregelen besproken, waarbij een onderscheid is gemaakt naar preventieve en repressieve maatregelen.

Preventieve maatregelen

Gebruik van aanvullende faciliteiten uit de Java-API
Alhoewel strikt genomen geen deel van het Java-beveiligingsmodel, maakt een aantal beveiligingsgerelateerde faciliteiten deel uit van de Java-API. Daarmee staan deze faciliteiten ter beschikking van de bouwers van Java-applets om het standaard beveiligingsniveau van Java-applets verder aan te vullen. Het gaat hierbij onder andere om (zie ook "case 2"):

- integriteits- en authenticatiecontrole van applets op basis van hash-functies en digitale handtekeningen;
- site-authenticatie op basis van certificaten.

Voorts is het gebruik van Access Control Lists voorzien, waarmee verder kan worden gedifferentieerd in de systeembronnen die wel of niet voor een Java-applet toegankelijk zijn.

Firewalls
Een firewall is een verzameling van hardware- en softwarecomponenten , inclusief daaromheen gedefinieerde procedures, die is geplaatst op het koppelvlak van netwerken om de risico's van die koppeling te beperken conform het gewenste beveiligingsniveau. Het gaat daarbij bijvoorbeeld om:
- het filteren van Internetservices, waarbij ongewenste services worden geblokkeerd;
- het beperken van communicatiemogelijkheden van interne systemen met het Internet en van het Internet met interne systemen;
- het verborgen houden van informatie over de structuur en samenstelling van het interne netwerk.

Firewalls zijn veelal in staat Java-applets (c.q. class-bestanden) te herkennen en deze te blokkeren. Daarmee wordt voorkomen dat de applets aan de clientzijde geladen en uitgevoerd worden.

Geheugenresidente programmatuur
Een nadeel van filtering door de firewall is dat deze doorgaans niet gedifferentieerd omgaat met applets: ze worden hetzij doorgelaten, hetzij geblokkeerd. Inmiddels is er geheugenresidente programmatuur verkrijgbaar, die hiermee gedifferentieerder omgaat. Dergelijke programma's zijn opgeslagen aan de clientzijde. Zodra een applet op de client machine wordt geladen, raadpleegt de programmatuur een database met gegevens over bekende vijandige applets. In die zin zijn de programma's vergelijkbaar met anti-virus programmatuur. Bijkomend voordeel is dat sommige varianten van deze programma's tevens faciliteiten bieden om de operaties van een Java-applet te registreren.

Repressieve maatregelen

Logging en alarmering
Logging is het vastleggen van informatie over relevante gebeurtenissen binnen een informatiesysteem. Op basis van gelogde gegevens is het mogelijk te reconstrueren welke gebeurtenissen op het systeem hebben plaatsgevonden. Dit is uit beveiligingsoogpunt wezenlijk om (pogingen tot) ongeoorloofde acties te kunnen traceren en de daarmee eventueel aangerichte schade te kunnen herstellen. Alarmering is gericht op het herkennen van vooraf gedefinieerde situaties *op het moment* dat deze zich voordoen, en het ondernemen van vooraf bepaalde acties als reactie op deze situaties. Logging en alarmering ten aanzien van applets

kan plaatsvinden door de firewall, door de eerder beschreven geheugenresidente programmatuur, of door een specifiek daarvoor gebouwde applicatie.

5.4 Organisatorische maatregelen

Organisatorische maatregelen hebben met name betrekking op het laatste item van categorie (b), zijnde `verantwoord gebruik´. Het zal duidelijk zijn dat het Java-beveiligingsmodel geen rol van betekenis speelt op het gebied van organisatorische maatregelen. Dit is geheel en al een verantwoordelijkheid van de organisatie die van Java gebruik maakt.

Preventieve maatregelen

Bevorderen beveiligingsbewustzijn
Een gebruiker heeft een voorname rol gekregen in het geheel van de beveiliging. In gedecentraliseerde en gedistribueerde systemen beheert hij zijn eigen IT-omgeving (zijn PC, software, vaste schijf, randapparaten, netwerkaansluiting), en bovendien zijn andere participanten in het netwerk afhankelijk geworden van de mate van beveiliging die hij toepast. Het is dan ook van groot belang dat gebruikers in een organisatie beveiligingsbewust handelen. Strakke regels en harde sancties zijn een mogelijkheid om gewenst beveiligingsgedrag af te dwingen, maar het is sterker de gebruiker te wijzen op de risico's van applets en te overtuigen van het belang van een veilig gebruik van Internet en Java-applets. Het zal in het algemeen, afhankelijk van het noodzakelijke beveiligingsniveau, nodig zijn om specifieke maatregelen te nemen om het beveiligingsbewustzijn bij individuele gebruikers te vergroten. Daarbij kan tevens een plaats worden ingeruimd voor de problematiek van *social engineering*, en voor het omgaan met programmatuur en bestanden die van het Internet worden gehaald. Verschillende middelen en methoden zijn voorhanden ter verhoging van het beveiligingsbewustzijn van gebruikers; verwezen wordt naar [ngi_95].

Formulering standaarden en gedragsregels voor de omgang met Java-applets
Als een organisatie onderkent dat risico's gemoeid zijn met het gebruik van Java-applets kan zij gedragsregels en standaarden opstellen voor haar gebruikers. Hierbij kan gedacht worden aan standaarden met betrekking tot de parametrisering van web-browsers, aan richtlijnen op het gebied van digitale handtekeningen, maar ook aan een meldplicht van onverwachte of vreemde systeemgedragingen bij het gebruik van Internet en Java-applets.

Op peil houden van kennis en volgen van relevante ontwikkelingen
De beveiligingsproblematiek rondom Java-applets vormt een complexe materie. Kennis en deskundigheid op dit gebied is noodzakelijk voor een organisatie die van Java-applets gebruik maakt. Door de verdere ontwikkeling van de programmeertaal Java en het optreden van fouten in de specificatie en implementatie van het Java-beveiligingsmodel is deze kennis veranderlijk. Dit maakt het continue op peil houden van deskundigheid op het gebied van Java in het algemeen en Java-applets in het bijzonder een belangrijke additionele beveiligingsmaatregel.

Repressieve maatregelen

Afhandeling beveiligingsincidenten
Een belangrijke repressieve organisatorische beheermaatregel is het opstellen van procedures en richtlijnen die gevolgd moeten worden op het moment dat (het vermoeden bestaat dat) een beveiligingsincident als gevolg van het gebruik van Java-applets heeft plaatsgevonden. Het kan daarbij zowel gaan om het aangeven van een centraal meldpunt als om het

definiëren van te ondernemen acties in termen van vastlegging, analyse, en oplossing. Doelstelling hiervan is om de schade als gevolg van beveiligingsincidenten zo snel mogelijk te ontdekken en zoveel mogelijk te beperken. Bovendien stelt een analyse van een opgetreden incident de organisatie wellicht in staat het stelsel van preventieve maatregelen structureel te uit te breiden en te verbeteren.

6. CONCLUSIES

Java-applets bieden interessante toepassingsmogelijkheden, maar kunnen tegelijkertijd op beveiligingsgebied een bron van zorgen vormen. Het Java-beveiligingsmodel zorgt voor een basis-beveiligingsniveau, waarvan de toereikendheid echter niet gegarandeerd is. Iedere organisatie die van de functionaliteit van Java-applets gebruik wenst te maken, dient na te gaan of aanvullende maatregelen noodzakelijk zijn, gezien de aard en het belang van de bedrijfsprocessen die door applet-gebruik zullen worden ondersteund. In dit artikel is door middel van het Java-applets Risico Model getracht de organisatie met een aansluiting op het Internet een handvat te geven. Het model biedt de mogelijkheid om, afhankelijk van de soort organisatie en de bedrijfsprocessen binnen deze organisatie, te bepalen welke risico's men loopt bij het ontvangen en uitvoeren van Java-applets. Tevens zijn een aantal additionele maatregelen besproken die het basis-beveiligingsniveau, die door het Java-beveiligingsmodel wordt gevormd, kunnen verhogen.

Curricula vitae

Drs. Mark Kilsdonk is als consultant werkzaam bij CMG, Trade, Transport & Industry. Hij houdt zich bezig met de implementatie van SAP-systemen. Als afstudeerproject voor zijn studie Bestuurlijke Informatica van de Erasmus Universiteit Rotterdam heeft hij een onderzoek gedaan naar de beveiligingsproblematiek rondom Java-applets.

Dr.Ir. Jan van den Berg is als universitair docent werkzaam bij de vakgroep Informatica van de Economische Faculteit van de Erasmus Universiteit Rotterdam. Zijn onderwijs concentreert zich op de vakken computersystemen en computernetwerken. Binnen zijn onderzoek staan onderwerpen uit de kunstmatige intelligentie centraal, in het bijzonder neurale netwerken, fuzzy systems en genetische algoritmen.

Drs. Ed Ridderbeekx Re RI is als hoofd IT Audit werkzaam bij de Interne Accountantsdienst van Generale Bank Nederland N.V.. Hij houdt zich bezig met de beoordeling van en advisering over kwaliteit van toepassing van informatietechnologie. Daarbij hebben netwerken en internetworking zijn bijzondere interesse en aandacht. Hij schrijft dit artikel op persoonlijke titel.

Literatuur

[kils_97]	Kilsdonk, M.M., "Java-applets risico's en beveiliging". *Doctoraalscriptie* Erasmus Universiteit Rotterdam, 1997. (http://www.few.eur.nl/few/people/jvandenberg/masters.html).
[ribe_98]	Ridderbeekx, E.J.M., en J. van den Berg, "Internetbeveiliging: een beheerperspectief". In: *Informatie*, april 1998.
[graw_96]	Mcgraw, G., en E. Felten. *Java Security*. Wiley Computer Publishing, 1996
[bemel_94]	Bemelmans, T.M.A., *Bestuurlijke informatiesystemen en automatisering*. Kluwer Bedrijfswetenschappen 1994.
[ngi_95]	"Beveiligingsbewustzijn bij gegevensbescherming. Hoe dit ten goede te beïnvloeden". *Rapport* van de afdeling beveiliging van het Nederlands Genootschap voor Informatica. Kluwer 1995.
[comp_98]	Themanummer over "Network Security" van het *IEEE*-tijdschrift *Computer*, vol. 31, nr. 9, sept. 1998.

In [kils_97] kunnen veel aanvullende literatuurverwijzingen worden gevonden.