Erschienen in: Fox, D.; Horster, P.: *Datenschutz und Datensicherheit – DuD*, Verlag Vieweg, Braunschweig 1999, S. 192-206.

Sicherheit in Java und ActiveX

Holger Mack

Secorvo Security Consulting GmbH
mack@secorvo.de

Zusammenfassung

In den vergangenen Jahren sind viele Diskussionen über die Sicherheitsrisiken von ausführbarem Inhalt von Web-Seiten z.B. Java Applets und ActiveX Controls geführt worden. Während vor allem die Entwickler solcher Technologien (z.B. Sun, Microsoft) die Benutzer überzeugen wollen, daß solche Techniken sicher angewendet werden können, sind viele Benutzer und Organisationen sehr skeptisch und vorsichtig im Umgang mit solchen Technologien. Dieser Beitrag wird an den Beispielen Java und ActiveX die Risiken solcher Technologien aufzeigen. Nach einer allgemeinen Besprechung der Problematik werden die von den Herstellern angebotenen Sicherheitsansätze diskutiert und ihre Stärken und Schwächen sowie ihr praktischer Nutzen bewertet.

1 Einleitung

Das Aufkommen von Technologien wie Java und ActiveX haben ein neues Problem bei der Nutzung des WWW gebracht. Bis dahin bestanden WWW-Seiten ausschließlich aus passiven Elementen wie z.B. Texten, Grafiken, Video- und Sounddaten. Diese Objekte stellen keine direkte Bedrohung für den Anwender dar, da sie nur vom Browser des Benutzers dargestellt werden, aber nicht aktive Kommandos auf dem Zielrechner ausführen können.[1] CGI-Scripts [2] waren die einzige Möglichkeit WWW-Seiten um interaktive Elemente zu erweitern. Da CGI-Scripts allerdings auf dem Server ausgeführt werden, stellen sie keine Bedrohung für den Benutzer dar.

Java-Applets und ActiveX Controls hingegen haben ihre Stärke gerade darin, daß sie aktive Operationen auf dem Rechner des Benutzers ausführen können. In erster Linie führt dies dazu, daß Entwickler viele neue Möglichkeiten haben, die Darstellung und Funktionalität ihrer Web-Seiten zu erhöhen, sie können sogar verteilte Anwendungen entwickeln. Allerdings bringen solche Programme neue Sicherheitsprobleme mit sich.

An dieser Stelle sollte darauf hingewiesen werden, daß das Herunterladen von Applets/Controls als Teil von Web-Seiten über Netzwerke nicht das einzige Anwendungsgebiet von Java oder ActiveX ist. Dieser Artikel wird sich allerdings auf diese

[1] Gefährdungen können allerdings bei Ausführung der zugehörigen Anwendungen entstehen; das ist aber kein spezifisches Problem von Daten im WWW.

[2] Common Gateway Interface

Anwendungen konzentrieren bei denen Java/ActiveX Programme dynamisch über Netzwerke geladen werden, da diese aus Gründen der Sicherheit von besonderem Interesse sind.

2 Bedrohung

Prinzipiell sind die Bedrohungen, die durch das Laden von Java-Applets oder ActiveX-Controls entstehen können, ähnlich denen, die auch bei normalen Programmen bestehen. Werden sie ohne Beschränkungen auf dem Zielsystem ausgeführt, sind mehrere Gefährdungen möglich:

- Das Programm kann Viren enthalten, die das System infizieren und schädigen.
- Das Programm könnte als „trojanisches Pferd„ unbekannte, schädigenden Nebeneffekte haben die vom Entwickler eingebaut wurden. Ein Beispiel dafür sind die Angriffe auf Nutzer des T-Online-Systems im letzten Jahr[LUC_98].
- Das Programm kann vom Entwickler nicht beabsichtigte Nebeneffekte haben (z.B. durch Programmierfehler).

Die Auswirkungen solcher Gefährdungen können dabei verschiedener Art sein:

- Veränderungen des Systems
- Einsicht in vertrauliche Daten des Benutzers
- Angriffe auf die Verfügbarkeit eines Systems (Denial of Service)

Zum Schutz vor solchen Bedrohungen werden üblicherweise verschiedene Methoden angewendet:

- Programme werden vor dem ersten Ausführen mit Hilfe von Virenscannern nach bekannten Virenmustern abgesucht.
- Programme werden zuerst in einer Test-Umgebung (z.B. einem „Quarantäne-Rechner„) ausgeführt, um zu untersuchen, ob unerwünschte Nebeneffekte auftreten.
- Es werden nur Programme von sogenannten vertrauenswürdigen Quellen (z. B. renommierte Hersteller) auf dem System ausgeführt.

Wenn es sich um traditionelle Programme handelt, die im allgemeinen fest auf dem Zielsystem installiert werden, sind diese Methoden relativ effektiv durchführbar, ohne größere Einbußen in der Funktionalität in Kauf nehmen zu müssen.

Alle drei Ansätze sind allerdings wenig geeignet für Java-Applets oder ActiveX Controls. Das Hauptproblem liegt darin, daß Anwendungen, die auf diesen Technologien beruhen, im Gegensatz zu traditionellen Anwendungen nicht langfristig auf dem Zielsystem installiert und gespeichert werden, sondern jeweils beim Aufrufen der Web-Seite dynamisch neu auf das System geladen werden. Dabei ist es durchaus möglich, daß beim erneuten Aufrufen einer Anwendung geänderter Code (z.B. neue Version, Bug-fixes etc.) geladen wird. Ohne Einbußen bei der Dynamik und Funktionalität ist es deshalb nicht möglich, die Applets/Controls erst in einer geschützten Umgebung auszuführen. Auch ist es ohne zusätzliche Maßnahmen nicht möglich, die Herkunft eines geladenen Programmteils eindeutig festzustellen und nur Applets/Controls aus vertrauenswürdigen Quellen auszuführen.

Das Problem von Virenscannern ist, daß sie grundsätzlich nur solche Viren identifizieren können, die bereits bekannt sind. Es besteht also immer die Gefahr, daß neue Viren in das System gelangen, die von der Virenscanner-Software nicht erkannt werden. Die Hersteller von Virenscannern haben zunehmend Schwierigkeiten, mit der ständig wachsenden Anzahl der Viren und Viren-Varianten Schritt zu halten; deshalb wird auch hier nach anderen Wegen gesucht.

Eines der Hauptmerkmale von Java ist die Plattformunabhängigkeit, ActiveX Controls hingegen sind plattformabhängig (derzeit nur für Win32 Plattformen verfügbar). Java-Applets können schon heute ohne Veränderungen auf einer großen Anzahl verschiedener Betriebssysteme ausgeführt werden (z.B. MS-Windows, MAC-OS, UNIX). Plattformunabhängigkeit an sich stellt kein Sicherheitsproblem dar, allerdings kann es die Folgen und die Verbreitung eines erfolgreichen Angriffs erhöhen. Bis heute hat sich die Vielfalt der am Internet angeschlossenen Plattformen aus sicherheitstechnischer Sicht als eine „natürliche Barriere„ ausgewirkt. Dadurch waren Sicherheitsprobleme oft auf eine Plattform beschränkt.[3] Ein Angriff, der mit Java realisiert würde, würde hingegen nahezu alle gängigen Plattformen gefährden. Welche Folgen das haben kann, ist am Beispiel der rasanten Verbreitung von Macro-Viren zu sehen.

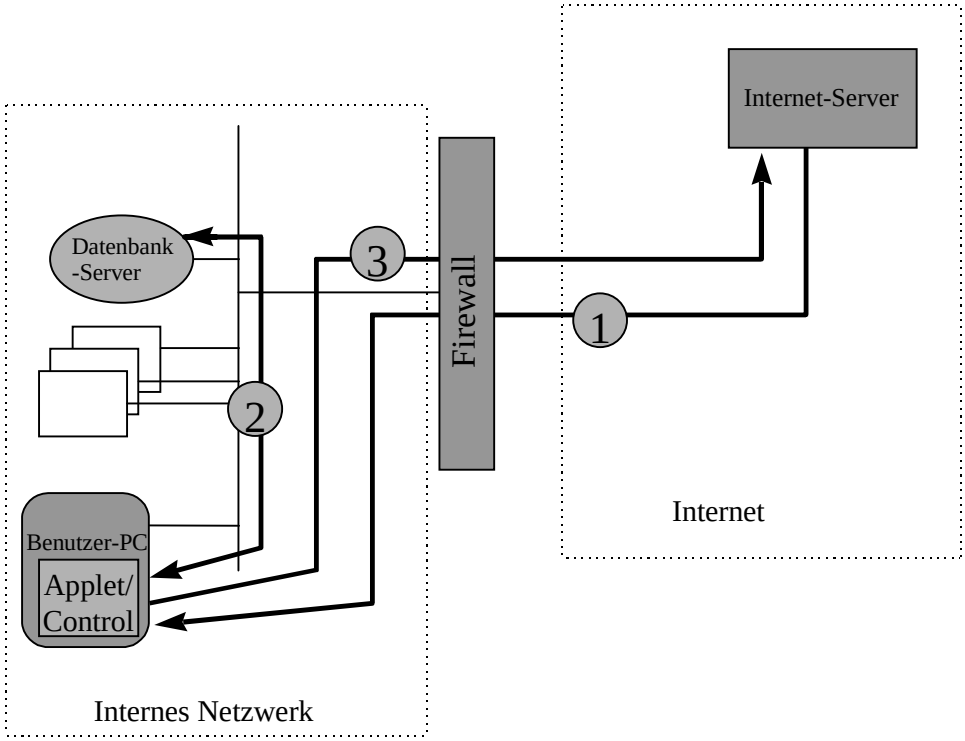

Abb. 1: Angriffsszenario

Java Applets oder ActiveX Controls stellen aber nicht nur eine Gefahr für den Rechner dar auf dem sie ausgeführt werden. Die Tatsache, daß solche Programme auf dem Zielsystem ausgeführt werden, kann dazu führen, daß ein Rechner, auf dem ein solches Programm unkontrolliert ausgeführt wird, als Ausgangspunkt für komplexere Angriffe verwendet werden kann (z.B. durch Versenden gefälschter E-Mail oder durch Ausnutzung von

[3] Beispiele dafür sind DOS-Viren oder der Internet-Wurm [EIRO_88].

Vertrauensbeziehungen in internen Netzen z. B. bei UNIX- oder Windows NT-Systemen). Das Applet/Control kann dabei ausnutzen, daß bei vielen Netzwerken die Sicherheitsmechanismen auf die Firewall beschränkt sind. Da Java-Applets und ActiveX Controls im internen Netzwerk ausgeführt werden, sind diese Kontrollen wirkungslos. Ein Programm könnte z.B. versuchen auf einen Datenbank-Server auf dem internen Netz zuzugreifen (siehe Abb. 1). Der Datenbankserver genehmigt den Zugriff da er von einer Maschine des internen Netzwerks erfolgt. Anschließend könnte das Applet die gewonnen Daten an den Server auf dem Internet senden.

3 ActiveX

Die ActiveX Technologie wurde von Microsoft nicht speziell für den Einsatz auf dem Internet entwickelt, sondern ist eine Art Nebenprodukt der Component Objekt Model (COM) Technologie, die eine wichtige Komponente der Microsoft Windows Architektur geworden ist [CHAP97]. Teile dieser Technologie (z.B. OLE) wird von den meisten Windows-Anwendern unwissentlich fast täglich eingesetzt. OLE erlaubt es z.B. Objekte in Microsoft Office Dokumente einzufügen (z.B. ein Excel-Sheet in ein Word-Dokument). Die ActiveX Technologie wird derzeit nahezu ausschließlich von Microsoft Windows Plattformen in Verbindung mit dem Microsoft Internet Explorer unterstützt. Auf der ActiveX Technologie beruhende Programme werden in Form sogenannter Controls vom Internet geladen. Durch geeignete Plug-Ins können ActiveX-Controls inzwischen auch mit Netscape Browsern benutzt werden. Microsoft strebt außerdem danach die COM Technologie und damit auch ActiveX auf andere Plattformen (z.B. MAC-OS) zu erweitern.

ActiveX Controls können prinzipiell in jeder beliebigen Programmiersprache programmiert sein (am häufigsten C/C++, Visual Basic) und werden in maschinenabhängiger Form auf dem Internet bereitgestellt und heruntergeladen. Zur Ausführung auf dem Zielsystem benötigen ActiveX Controls einen sogenannten Container. In unserem Fall ist das ein ActiveX fähiger Browser. Ansonsten werden ActiveX Controls wie traditionelle Anwendungen auf dem Zielsystem ausgeführt. Dies hat zur Folge, daß ActiveX Controls wie traditionelle Applikationen direkt auf das Betriebssystem und die Systemressourcen zugreifen (siehe Abb. 2) können.

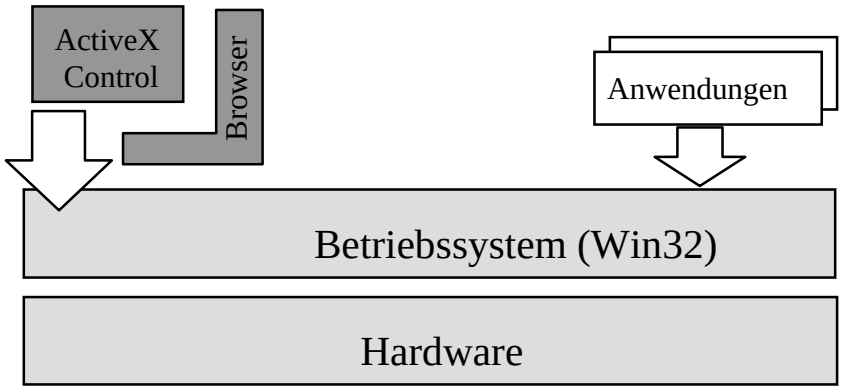

Abb. 2: ActiveX Model

Für Windows95/98 bedeutet dies vollen Zugriff auf alle Ressourcen. Unter WindowsNT hat das Control die gleichen Rechte wie der Browser selber bzw. jede andere Applikation die unter dem selben Account ausgeführt wird.

3.1 ActiveX Sicherheitsmechanismen

Da die ActiveX Technolgie nicht speziell für den Einsatz auf dem Internet entwickelt wurde, spielten Sicherheitsüberlegungen bei der Entwicklung keine Rolle. Aus diesem Grund verfügt ActiveX, anders als Java, über kein spezifiziertes Sicherheitsmodell. Für den Einsatz auf dem Internet wurde nachträglich die Möglichkeit eingeführt, ActiveX Controls digital zu signieren. Wenn ein solches signiertes Control vom Internet (oder Intranet) geladen wird, wird der Benutzer vor die Wahl gestellt, die Ausführung zuzulassen bzw. zu unterbinden Hierzu erscheint ein Fenster mit den Angaben aus dem zugehörigen Zertifikat. Der Benutzer hat außerdem die Möglichkeit die Ausführung von Controls die von bestimmten Quellen signiert wurden generell zu gestatten bzw. zu verbieten. Wird die Ausführung zugelassen hat das ActiveX Control automatisch vollen Zugriff wie oben beschrieben(„Alles oder Nichts„-Ansatz).

Das Signieren der Controls wird mit der von Microsoft entwickelten Authenticode Technologie durchgeführt. Diese Technologie erlaubt das Signieren von sogenannten CAB Files die verschiedene Objekte (u.a. ActiveX Controls) beinhalten können. Die gleiche Technologie wird im Internet Explorer auch für das Signieren von Java Applets verwendet.

4 Java

Sun hat mit dem Java-Sicherheitsmodell einen proaktiven Ansatz gewählt. Ziel des Modells war es, das Java-System von vorneherein so zu gestalten, daß Java-Applets auf dem Zielrechner ausgeführt werden können, ohne daß eine Gefahr für das System besteht.

4.1 Voraussetzungen

Der Ansatz des Sicherheitsmodells ist, in das Java-System Mechanismen einzubauen, mit deren Hilfe die Ausführung sicherheitskritischer Operationen auf einem Zielrechner kontrolliert werden kann. Sun entwickelte deshalb das Prinzip der „Sandbox„[4]. Die Idee ist, daß Applets innerhalb einer vom Benutzer kontrollierten, eingeschränkten Systemumgebung, einer „virtuellen Java-Maschine„ ausgeführt werden. Zugriffe auf Ressourcen außerhalb dieser Systemumgebung sollten nur erlaubt sein, wenn sie der Sicherheits-Policy der jeweiligen Umgebung entsprechen.

Für die Kontrolle sind im Java-Sicherheitsmodell drei Teile verantwortlich, deren fehlerfreie Funktion entscheidend für das sichere Ausführen von Java-Applets ist:

- der Security Manager,
- der Byte Code Verifier und
- der Class Loader.

[4] Siehe auch Fox, DuD 2/1998, S. 96.

In der Literatur werden der Class Loader, der Security Manager und der Byte Code Verifier manchmal als „the three lines of defence„ des Java-Sicherheitsmodells bezeichnet . Schon in [GRFE_96] wird darauf hingewiesen, daß dies einen falschen Eindruck hinterläßt. Die Teile stellen keine drei Verteidigungslinien dar, die alle von einem Angreifer überwunden werden müssen. Vielmehr setzt das Java-Sicherheitsmodell ein einwandfreies Funktionieren aller drei Teile voraus.

4.1.1 Java Virtual Maschine

Java-Applets werden nicht direkt vom Betriebssystem ausgeführt, sondern in einer plattformunabhängigen Form, dem sogenannten Byte Code, auf das System geladen und von der Java Virtual Machine (JVM) in systemspezifischen Maschinencode umgesetzt (interpretiert) (siehe Abb. 3). Die JVM ist der maschinenabhängige Teil des Java-Systems, der dafür sorgt, daß Java Applets auf verschiedenen Systemen ausgeführt werden können, und ist üblicherweise Teil eines Java-fähigen Browsers.

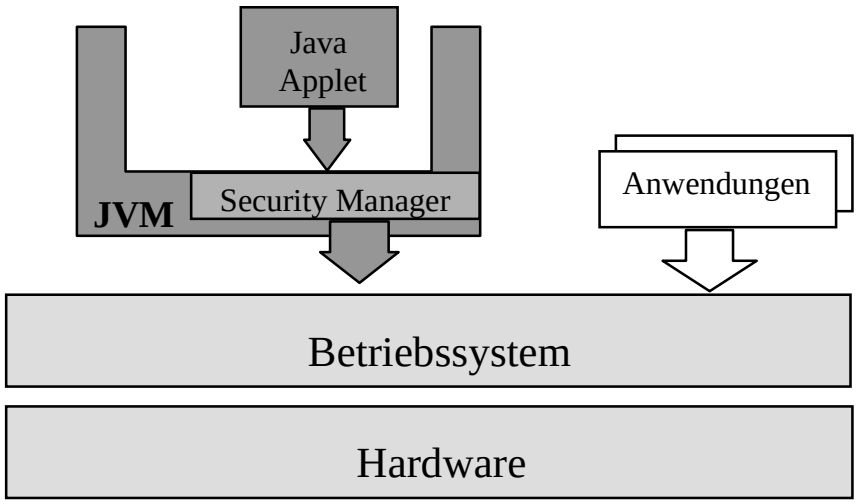

Abb. 3: Java Applet Model

Ein Java-Applet kann nur mit Hilfe der JVM auf Ressourcen des Betriebssystems zugreifen; dort werden die Zugriffe auf Systemressourcen kontrolliert. Der Teil, mit dem die Zugriffskontrolle realisiert wird, ist der Security Manager. Abhängig von jeweiligen Implementation wird von diesem eine Zugriffsentscheidung getroffen, wenn ein Applet auf eine kritische Ressource zugreifen will.

Welcher Zugriff dabei vom Security Manager auf Grundlage welcher Faktoren (z.B. Herkunft des Applets) gewährt wird, ist nicht im Java-Sicherheitsmodell festgelegt, sondern hängt von der jeweiligen Implementierung der Methoden des Security Managers ab.

Die Operationen, die als gefährlich eingeschätzt werden und deshalb mit Hilfe des Security Managers kontrolliert werden können, sind:

- Netzwerkzugriffe,
- das Laden von Applets,
- Zugriffe auf Java-Klassen,
- alle Operationen zum Manipulieren von und der Zugriff auf Threads,

- der Zugriff auf Systemressourcen wie z. B. die AWT Event Queue,
- das Erzeugen von Toplevel Windows,
- Zugriffe auf das Dateisystem, sowie
- das Aufrufen von lokalen Programmen und Betriebssystem-Kommandos.

4.1.2 Byte Code Verifier

Ein großer Teil der Sicherheitsmechanismen von Java ist eng mit der Sprachspezifikation von Java verknüpft. Auf den ersten Blick ähnelt Java C++, allerdings ist Java sehr viel strikter spezifiziert, um problematische Spracheigenschaften zu vermeiden. So gibt es in Java z. B. keine Zeiger und ein direkter Speicherzugriff ist nicht möglich. Außerdem ist die objektorientierte Sprache Java stark typgebunden, mit Zugriffsregeln für die Methoden und Parameter eines Objekts. Speziell diese Zugriffsregeln sind entscheidend für das Java-Sicherheitsmodell.

Die Aufgabe des Byte Code Verifiers ist es, zu prüfen, ob die geladenen Applets der Java-Sprach-Spezifikation entsprechen. Das Einhalten der Sprachspezifikation sollte bereits von Java-Compilern geprüft werden. Da bei Java-Applets von unbekannten Quellen nicht davon ausgegangen werden kann, daß der Byte Code mit einem vertrauenswürdigen Compiler (oder überhaupt einem Compiler) erzeugt wurde, überprüft der Byte Code Verifier vor Ausführen eines Applets, ob es der Java-Sprachspezifikation entspricht. Dabei versucht er auch festzustellen, ob ein Applet die Kontrollen des Security Managers umgeht.

Die Byte Code-Dateien (.class-Files), die als Teil eines Applets auf das Zielsystem geladen werden, enthalten neben den auszuführenden Operationen noch Zusatzinformationen, die den Prozeß des Byte Code Verifiers unterstützen. Der Byte Code Verifier versucht in mehreren Durchläufen zu erkennen, ob ein Applet irreguläre Zustände verursachen kann. Eine detailierte Beschreibung der Vorgehensweise des Byte Code Verifiers kann in [YELL_96] gefunden werden.

4.1.3 Class Loader

Die Aufgabe des Class Loaders ist es, dem auszuführenden Applet alle erforderlichen Klassenbibliotheken zur Verfügung zu stellen.[5] Von entscheidender Bedeutung ist dabei die Herkunft einer Klasse. Die Klassen, die lokal in einem speziellen Verzeichnis (spezifiziert in der Umgebungsvariable CLASSPATH) abgelegt sind, werden nicht denselben Kontrollen unterzogen wie Klassen, die über das Internet geladen werden. Wenn ein Applet eine entsprechende Klasse benötigt, durchsucht der Class Loader zuerst die lokalen Klassen und anschließend erst externe Quellen (z. B. das Herkunftsverzeichnis des Applets im Internet).

Der Class Loader sorgt ebenfalls dafür, daß verschiedene Applets, die gleichzeitig auf dem System ausgeführt werden, sich nicht gegenseitig beeinflußen können. Dies wird mit Hilfe eines getrennten Namensraums für jedes Applet realisiert.

[5] Java Applets werden üblicherweise nicht als ein komplettes Programm geladen, sondern die benötigten Teile (Java-Klassen) werden bei Bedarf nachgeladen.

4.2 Grenzen des Modells

Das Java-Sicherheitsmodell ist auf Grund verschiedener Eigenschaften kritisiert worden. Auf die wichtigsten soll hier kurz eingegangen werden:

- Komplexität der sicherheitskritischen Teile

  In der Literatur wird oft darauf hingewiesen, daß es das Ziel sein sollte, den sicherheitskritischen Teil eines Systems (Trusted Computing Base, TCB) so klein und einfach wie möglich zu gestalten. Denn je größer der sicherheitskritische Teil eines Systems ist, desto schwieriger ist es, diesen Teil fehlerfrei zu implementieren bzw. nachzuweisen, daß er als Ganzes effektiv ist. Ist dagegen der sicherheitskritische Teil kompakt, kann möglicherweise die Fehlerfreiheit formal verifiziert werden. Im Java-System ist der sicherheitskritische Teil relativ groß (ca. 23.000 Code-Zeilen), was eine intensive Überprüfung sehr aufwendig werden läßt. Die Schwierigkeit, das Java-Sicherheitsmodell korrekt zu implementieren, manifestiert sich auch in den bei Browser-Implementierungen immer wieder aufgetretenen Problemen [CERT1,CERT2].

- Keine eindeutige Relation zwischen Byte Code und Java Code

  Es gibt keine Möglichkeit nachzuweisen, daß es nicht möglich ist, Byte Code zu schreiben, der zwar nicht der Java-Sprachdefinition entspricht, aber trotzdem vom Byte Code Verifier als gültig anerkannt wird. Es ist bereits gelungen, gültigen Byte Code zu schreiben, der von einem Java-Compiler nicht generiert werden kann [LADU_97]. Die Java-Sprachspezifikation und deren Einhaltung sind jedoch von zentraler Bedeutung für das Java-Sicherheitsmodell. Daher ist dieses Problem besonders kritisch, denn es kann zu einer Untergrabung des Java-Sicherheitssystems führen.

- Verteiltes Modell

  Eine der Schwächen des Java-Modells ist, daß es sich um ein verteiltes Modell handelt. Alle Sicherheits-Checks werden auf jedem einzelnen ausführenden Rechner durchgeführt. Die TCB wird dabei beliebig groß. Um sicherzustellen, daß ein komplettes Netzwerk gegen Java-Angriffe geschützt ist, muß daher sichergestellt werden, daß jeder einzelne Rechner sicher ist. In der Firewall-Literatur [CHBE_94] wird immer wieder darauf hingewiesen, daß eine Verteidigungsstrategie, bei der jeder einzelne Netzwerk-Rechner geschützt werden soll, nahezu unmöglich zu kontrollieren und zu administrieren ist. Wie bereits erwähnt, kann bereits ein ungeschützter Rechner dazu führen, daß ein Angreifer Zugriff auf mehrere Komponenten eines Netzwerks gewinnt.

4.3 Weiterentwicklungen

Das Java-Sicherheitsmodell ist seit seiner Präsentation viel diskutiert worden. Einige der Weiterentwicklungen, die aus diesen Diskussionen hervorgegangen sind, werden in diesem Abschnitt vorgestellt.

4.3.1 Signed Applets

In den ersten Implementierungen des Java-Sicherheitsmodells in Netscape oder Microsoft Browsern hatte der Benutzer zwei Möglichkeiten:

- Das Ausführen von Applets konnte generell unterbunden werden

- Alle Applets wurden ausgeführt, allerdings mit sehr starken Restriktionen [GRFE_96]

Dabei wurden generell alle Applets, egal aus welcher Quelle oder mit welcher Funktion, gleich behandelt. Wünschenswert ist hingegen, daß es möglich ist, abhängig von der Herkunft des Applets entweder die Ausführung zu unterbinden oder einem Applet weiterreichende Rechte einzuräumen. Als Hauptproblem dabei erweist sich, daß es die im Internet eingesetzten Protokolle (TCP/IP, HTTP) nicht erlauben, die Herkunft eines Applets zweifelsfrei zu bestimmen. Mit dem Java Development Kit (JDK) 1.1 hat Sun deshalb sogenannte „Signed Applets„ eingeführt.[6] Hinter dem Begriff „Signed Applets„ verbergen sich digital signierte Applets (bzw. die einzelne Teile eines Applets) ähnlich wie der Authenticode Mechanismus von Microsoft[FRMU_96]. Durch eine Prüfung der digitalen Signatur kann auf dem Zielsystem festgestellt werden, von wem dieser Code signiert wurde und ob er seitdem verändert wurde.

Die erste Realisierung von signed Applets im JDK1.1 entsprach dem „Alles-oder-Nichts„ Ansatz von ActiveX, d.h. für Applets mit entsprechender Signatur wurden automatisch alle Zugriffsbeschränkungen aufgehoben.

Parallel zu den Bemühungen von Sun digitale Signaturen in den JDK einzubringen, haben auch Netscape und Microsoft begonnen eigene Mechanismen zum Signieren von Java Applets zu implementieren.

Microsoft benutzt dabei die gleiche Authenticode Technologie wie auch für ActiveX Controls. Beim Signieren von Java-Applets mit Hilfe von Authenticode hat der Signierer außerdem die Möglichkeit eine Sicherheitsstufe anzugeben in der das Applet ausgeführt werden soll (low, medium, high). Die Java Sicherheitseinstellungen in den verschiedenen Stufen sind im Browser (Internet Explorer 4.x) vorgegeben (z.B. low-security bedeutet vollen Zugriff für das Applet). Wenn der Benutzer der Ausführung des Applets zustimmt, wird dieses in der geforderten Sicherheitsstufe ausgeführt.

Netscape dagegen hat mit dem Netscape Object Signing einen anderen Ansatz verfolgt. Applets können in sogenannten JAR-Files (entwickelt von Sun) signiert werden. Mit Hilfe einer eigenen API können Java Applets so erweitert werden, daß sie sich das Recht (Privilege) auf eine spezielle Ressource zugreifen zu dürfen „erfragen„ müssen. Dies erfordert, daß das Applet mit Hilfe der Netscape Capabilities API erweitert wird. Bevor das Applet auf eine Systemresource zugreifen kann, muß es sich durch einen entsprechenden Funktionsaufruf dieses Recht holen. Im Browser hat eine solche Anfrage zur Folge, daß ein Fenster ähnlich dem im Microsoft Browser erscheint. Der Hauptunterschied ist hierbei, daß der Benutzer über einen spezifischen Zugriff entscheiden kann z.B. Lesezugriff auf eine bestimmte Datei. Der Benutzer hat die Möglichkeit das Privileg nur einmalig zu vergeben oder es generell Applets mit einer bestimmten Herkunft (d.h. digital signiert mit dem gleichen Schlüsselpaar) zu erlauben. Der Hauptunterschied zu den Ansätzen von Microsoft und Sun ist hier, daß im Applet Netscape-spezifische Änderungen durchgeführt werden müssen um die Technolgie anzuwenden.

Obwohl alle drei Techniken ähnliche Funktionalität implementieren und alle Zertifikate im X.509v3 Format unterstützen, sind doch alle drei Techniken inkompatibel. Es ist daher nicht möglich ein signiertes Applet für alle Plattformen zu generieren.

[6] Eine ähnliche Funktionalität ist auch von Microsoft und Netscape in die Browser eingebaut worden

4.3.2 Protection Domain-Architektur

Mit Java2 (lange Zeit bekannt als JDK1.2) führte JavaSoft einige weitreichende Erweiterungen des Sandbox-Modells (Security Manager, Byte Code Verifier, Class Loader) ein [GOMU_98]. Die Funktion dieser Teile bleibt prinzipiell erhalten, nur der Security Manager und der Class Loader wurden leicht modifiziert.

Das Hauptmerkmal der Erweiterungen ist, daß die Sicherheitsregeln nicht programmiert werden müssen, sondern textuell (oder mit dem in Java 2 enthalten Policy Tool) unabhängig von der Implementierung bestimmt werden können. Die Regeln können dabei basierend auf digitalen Signaturen und/oder basierend auf der Herkunft (URL) des Codes sehr detailliert (z. B. verschiedene Zugriffsarten auf einzelne Dateien oder Verzeichnisse) vergeben werden.

Der Byte Code wird ab Java 2 in sogenannten Protection Domains ausgeführt. Die Rechte des in einer Protection Domain ausgeführten Codes setzen sich dabei aus allen Rechten zusammen, die an die Herkunfts-URL des Codes vergeben worden sind, sowie den Rechten, die an die Identitäten vergeben wurden, die eine digitale Signatur für das Applet generiert haben (jedes Applet kann dabei von mehreren Identitäten unterschrieben werden). Es gibt in Java 2 auch keine Unterscheidung zwischen Byte Code, der lokal auf dem Rechner gespeichert ist, und solchem, der vom Internet geladen wird.

Jeder ausgeführte Byte Code hat nur genau die Zugriffsrechte der ausführenden Protection Domain. Einzige Ausnahme ist die sogenannte System Domain. Byte Code, der in der System Domain ausgeführt wird, darf direkt auf die Systemresourcen zugreifen bzw. im Auftrag der laufenden Applets Systemzugriffe ausführen.

Beim Zugriff auf sicherheitskritische Ressourcen wird auch weiterhin der Security Manager konsultiert. Allerdings sind die Zugriffsregeln nicht mehr im Security Manager implementiert, sondern werden vom Security Manager an den sogenannten Access Controller weitergeleitet. Der Access Controller sammelt alle Rechte, die dem Byte Code zugewiesen wurden und entscheidet dann darüber, ob der Zugriff erteilt werden soll oder nicht. Dabei werden nicht die Rechte des unmittelbar aufrufenden Byte Codes zugrunde gelegt, sondern die Rechte des Codes in der Kette der aufrufenden Funktionen, der die geringsten Rechte besitzt („least privilege„-Prinzip). Dadurch soll gewährleistet werden, daß kein Applet, das von Byte Code mit umfassenderen Zugriffsrechten aufgerufen wird, unzulässigerweise auf Systemressourcen zugreifen kann. Das explizite Untersagen von Aktionen für Applets bestimmter Herkunft oder für bestimmte Identitäten ist dagegen nicht möglich.

Derzeit unterstützen weder Microsoft noch Netscape die Erweiterungen in Java 2. Mit Hilfe des Java Plug-Ins von Sun kann die Java 2 Funktionalität auch jetzt schon in den Browsern zur Verfügung gestellt werden. Dies hat auch den Vorteil das die gleichen signierten Applets mit beiden Browsern verwendet werden können. Das Plug-In reagiert allerdings nicht auf die normalen Applet-tags sondern benötigt spezielle HTML-tags. Die Funktionalität der in die Browser eingebauten JVMs wird durch das Plug-In nicht beeinflußt.

5 Signierte Objekte

Obwohl digitale Signaturen das Problem der Herkunfstbestimmung technisch zu lösen scheinen, muß diese Technik doch mit Bedacht eingesetzt werden. Eine digitale Signatur gibt lediglich an, wer das Applet/Control signiert hat, sie macht aber keine Aussage darüber, wer

das Applet/Control entwickelt/programmiert hat bzw. ob das Applet unerwünschte Nebeneffekte hat oder nicht. Von einigen Herstellern wird den Benutzern suggeriert, daß eine digitale Signatur weitere Eigenschaften erfüllt, die in Wirklichkeit nicht gegeben sind (z.B. Authorship, Vertrauenswürdigkeit). Der Benutzer muß entscheiden, ob er den Signierer als vertrauenswürdig einstuft alle Teile des Applets/Controls entsprechend geprüft zu haben. Der Trend geht dazu Programme modular aus fertigen Teilen zusammenzubauen (z.B. mit Hilfe von JavaBeans) um so schneller und effektiver Software entwickeln zu können. Der Benutzer muß sich bewußt sein, daß er dem Unterzeichner vertrauen muß all diese Teile entsprechend geprüft zu haben.

Neben der technischen Realisierung müssen beim Einsatz von signed Objects noch eine Reihe von Randbedingungen geklärt werden:

- Eine Public Key-Infrastruktur (PKI) muß die erforderlichen Schlüsselzertifikate bereitstellen. Die Aussagekraft einer digitalen Signatur hängt entscheidend von der PKI ab von der das Zertifikat stammt. Indem der Benutzer die Ausführung eines Applets/Controls aufgrund des Unterzeichners zuläßt, muß er nicht nur der angeblichen Quelle (d.h. dem Unterzeichner) vertrauen. Entscheidend ist, daß die CA die entsprechend Sorgfalt bei der Püfung der Identität des Unterzeichners hat walten lassen, sowie alle weiteren nötigen Sicherheitsmaßnahmen befolgt.
- Heutzutage wird in der Regel auf Zertifikate von kommerziellen PKI Anbietern z.B. VeriSign zurückgegriffen. Die Anwendung solcher Zertifikate ist technisch relativ einfach da die meisten Browser standardmäßig mit den Root-Zertifikaten der großen kommerziellen PKI-Anbieter ausgestattet sind. Die meisten dieser kommerziellen Anbieter haben allerdings oft einige der wichtigsten Funktionalitäten wie z.B. Zertifikats Sperrlisten etc. nicht implementiert. Außerdem wird eine solche Funktionalität von den Browsern nicht unterstützt.
- Die Frage der Haftung bei digitalen Signatur muß geklärt werden. Derzeit ist die Frage der rechtlichen Anerkennung von digitalen Signaturen und der damit verbundenen Haftungsfragen noch ungeklärt. Sollte also ein signiertes Applet/Control Schaden auf dem System des Benutzers anrichten, heißt das noch lange nicht, daß irgendwelche Haftungsansprüche daraus abgeleitet werden können.
- Digitale Signaturen sind nicht wiederrufbar. Hat der Hersteller eines Applets einmal dieses Applet unterschrieben, so bleibt diese Unterschrift erhalten selbst wenn neue Versionen dieses Applets verteilt werden. Angenommen die ‘vertrauenswürdige’ Firma A signiert ein Applet und es stellt sich später heraus, daß dieses Applet eine Sicherheitslücke beinhaltet. Selbst wenn Firma A den Fehler behebt und die fehlerhafte Version ersetzt, kann ein Angreifer die alte fehlerhafte Version auch weiterhin einsetzen (z.B. auf dem eigenen Web-Server). Für den Benutzer bleibt das alte fehlerhafte Applet auch weiterhin ‘vertrauenswürdig’ da es von einer ‘vertrauenswürdigen’ Quelle unterzeichnet wurde. Der Benutzer hat keine Möglichkeit ohne weiteres eine fehlerhafte Kopie zu erkennen.
- Es muß festgelegt werden, wer Applets/Controls signieren darf (d.h. Firmen, Einzelpersonen oder unabhängige Instanzen).

Es wäre z.B. denkbar, daß im Bereich mit speziellen Sicherheitsanforderungen Applets von unabhänigen Instanzen (z. B. dem BSI oder anderen Zertifizierungsstellen) geprüft und

signiert werden, oder daß Unternehmen Applets nach einer Prüfung in einer Test-Umgebung signieren und damit zur Ausführung im internen Netzwerk freigeben.

Es muß außerdem geklärt werden, welche Rechte einem als „trusted„ erklärten Applet eingeräumt werden. Das Ziel sollte sein, einem Applet nur die Rechte zu gewähren, die es zur Ausführung seiner Aufgabe benötigt („need-to-know„-Policy). Diese Anforderung stellt zusätzliche Anforderungen an die Signed Applets bzw. die PKI, da nicht nur Informationen über die Herkunft des Applets benötigt werden, sondern auch anwendungsspezifische Informationen (z. B. welche Zugriffsrechte werden benötigt etc.).

6 Alternative Ansätze

Ähnlich wie in anderen Bereichen der IT-Sicherheit gibt es auch im Bereich der ausführbaren Inhalte von Web-Seiten mehrere Anbieter, die Sicherheitslösungen anbieten, die versprechen durch zusätzliche Maßnahmen die Sicherheit eines Systems zu verbessern. In diesem Abschnitt wird kurz auf einige der angebotenen und vorgeschlagenen Techniken eingegangen und kurz ihre Stärken und Schwächen zu diskutiert.

6.1 Filtern

Fast alle Firewall-Hersteller bieten Funktionalität an Java/ActiveX auf der Firewall zu filtern, so daß keine oder nur bestimmte Applets/Controls in das interne Netz gelangen. Dies ist relativ wirksam wenn es darum geht die entsprechenden HTML-tags aus WWW-Seiten zu entfernen. Eine lückenlose Filterung von mobilem Code ist allerdings nur schwer möglich, da ein solcher Code auch auf anderen Wegen (z.B. gepackt, verschlüsselt mit SSL, per-Email etc.) in das interne Netzwerk gelangen kann. Ähnlich wie bei der Virusproblematik ersetzen zentrale Kontrollen auf der Firewall nicht die lokalen Kontrollen, können aber durchaus als zusätzliche Maßnahme sinnvoll sein.

6.2 Scanner

Ein weiterer Ansatz ist der aus der Virusbekämpfung bekannten Ansatz Applets/Controls gegen bekannte Muster gefährlicher Applets/Controls zu vergleichen um so deren Ausführung auf dem System zu vermeiden. Im allgemeinen erfolgt ein solcher Vergleich anhand von Blacklists d.h. Listen mit bekannten gefährlichen Applets/Controls. Diese Technik hat mit Sicherheit ihre Grenzen, da es nahezu unmöglich ist alle gefährlichen Applets/Controls zu erfassen und eine ständig aktuelle Liste zu führen. Die Entwickler von Computerviren haben hier immer erstaunliche Kreativität bewiesen diese Art von Kontrollen zu umgehen.

Aus der Virenwelt ist aber bekannt, daß ein Großteil von Virenproblemen auf das Konto einer geringen Anzahl von bekannten Viren zurückzuführen ist. Aus diesem Grund können solche Scanner durchaus eine sinnvolle Ergänzung sein. Die meisten Antiviren-Produkte bieten einen Java/ActiveX Scanner standardmäßig an.

6.3 Inhaltskontrolle

Einige Hersteller vertreiben Produkte die anhand der Inspektion des Inhaltes eines Applets oder ActiveX Controls versuchen festzustellen, ob versucht wird möglicherweise schädliche Operationen auszufühen. Diese Kontrolle erfolgt zumeist statisch vor der Ausführung.

Aufgrund solcher Kontrollen eine eindeutige gefährlich-ungefährlich Entscheidung zu machen ist allerdings sehr schwierig und in der Praxis nur schwer anwendbar, da eine klare Abgrenzung zwischen gefährlichen und ungefährlichen Aktionen nur selten möglich ist.

6.4 SSL

Eine Alternative zum Signieren von Applets/Controls, ist die Verwendung von SSL zum Schutz der Übertragung von Daten zwischen Web-Server und Benutzer-PC. Der Vorteil dieser Technologie gegenüber dem Signieren von individuellen Applets/Controls ist, daß hier nicht nur die Herkunft und Unverfälschtheit der über die Verbindung geladenen Programm-Teile authentifiziert werden können, sondern daß hier alle weiteren Teile der WWW-Seite (z.B. HTML-Text, Bilder etc.) ebenso geschützt werden. Zusätzlich zur Authentisierung kann die Übertragung außerdem durch Verschlüsselung vor den Blicken Unberechtigter geschützt werden. Durch Verwendung der in SSL Version 3 vorhandenen Client-Authentication besteht zusätzlich für den Web-Anbieter die Möglichkeit einwandfrei zu authentifizieren, wer auf die Web-Seite zugreifen will. Der Anbieter könnte so den Zugriff auf Applets/Controls nur für berechtigte Benutzer zulassen (z.B. nach vorheriger Bezahlung). Für den Anbieter hat diese Methode außerdem den Vorteil, daß das Problem mit dem Zurückziehen von Unterschriften (siehe oben) unter Applets/Controls nicht mehr besteht. Der Angreifer müßte in der Lage sein eine ganze SSL-Session zu fälschen um so zum Erfolg zu kommen.

Bezüglich der notwendigen PKI-Unterstützung hat die Verwendung von SSL die gleichen Probleme wie signierte Applets/Controls (siehe oben). Moderne Browser bieten aber noch keine Möglichkeit die Authentifizierung von SSL mit den Zugriffsrechten für Applets/Controls zu verknüpfen.

7 Fazit

Wie in den meisten Bereichen der Computertechnik bringt der Einsatz von neuen Technologien nicht nur Vorteile sondern verursacht auch Probleme. Java und ActiveX sind da keine Ausnahme. Erinnern wir uns, daß vor nicht allzulanger Zeit die meisten Firmen davon absahen ihre Netzwerke an das Internet anzuschließen, da der Nutzen die Risiken eines Internetanschlußes nicht gerechtfertigt hat. Heutzutage hat das Internet so an Bedeutung gewonnen, daß es sich kaum noch eine Firma leisten kann nicht angeschlossen zu sein. Die Risiko-Nutzen Abschätzung hat sich zu Gunsten des Internets verschoben.

Im Bereich der ausführbaren Inhalte von Web-Seiten zeichnet sich eine ähnliche Entwicklung ab. Wie in diesem Beitrag besprochen, bergen diese Technologien eine ganze Reihe von Risiken. Jeder potentielle Benutzer von solchen Technologien sollte sich dieser Risiken bewußt sein. Eine generelle Entscheidung zum Einsatz von Java oder ActiveX kann nicht gegeben werden, die Entscheidung muß abhängig von der jeweiligen Situation gefällt werden. Noch vor einem Jahr war diese Entscheidung relativ einfach; Java und ActiveX wurden hauptsächlich zur Verschönerung von Web-Seiten eingesetzt. Durch das Abschalten von Java/ActiveX hatte der Benutzer nur selten ernsthaft Einschränkungen bzgl. der Funktionalität auf dem Internet in Kauf zu nehmen. Heute ist die Entscheidung schon um einiges schwieriger, da mehr und mehr Web-Seiten sehr stark auf Java/oder ActiveX aufbauen, und so der Nutzen des Internets durch das Abschalten von Java sehr stark beeinträchtigt sein kann. Vor allem Java wird auch mehr und mehr für die Entwicklung verteilter Anwendungen

eingesetzt. Natürlich steigt durch diesen verstärkten Einsatz gerade in lukrativen Bereichen wie E-Commerce auch der Anreiz für Angreifer Fehler und mögliche Angriffspunkte in solchen Systemen zu finden.

Wie oben beschrieben verfolgen Sun und Microsoft sehr unterschiedliche Strategien bzgl. der Sicherheit ihrer Systeme. Der „Alles oder Nichts„-Ansatz von ActiveX ist sicherlich nicht ausreichend die Problematik auf dem Internet zu lösen. ActiveX ist eine gefährliche Technik da sie automatisch jedem Control das auf dem System ausgeführt wird vollen Zugriff auf Systemressourcen einräumt. Es gibt eine Reihe von Beispielen welche die Gefährlichkeit dieser Technik verdeutlichen. Ohne große Probleme sind alle der am Anfang beschriebenen Angriffe mit ActiveX realisierbar. Allein das Signieren von ActiveX Controls bietet aus den oben genannte Gründen keine angemessenen Lösung.

Der Ansatz von Java ist hier deutlich vielversprechender, da die Sicherheit von Anfang an eine Rolle bei der Entwicklung gespielt hat. Was den Ansatz von Java besonders attraktiv macht, ist die Möglichkeit verschiedenen Applets verschiedene Ausführungsrechte einzuräumen. D.h. es ist möglich eine „Need-to-know„-Policy zu implementieren; jedem Applet werden nur genau die Rechte gestattet die es für seine Aufgabe benötigt. Vor allem das Protection Domain Modell ist ein wichtiger Schritt in die richtige Richtung. Dies ist ein Ansatz der auch für andere Bereiche (z.B. Betriebssysteme) sehr interessant sein kann. Die meisten Sicherheitsprobleme die durch Dinge wie Computerviren und trojanische Pferde verursacht werden sind darauf zurückzuführen, daß Anwendungen Rechte eingeräumt werden, die sie eigentlich zur Ausführung ihrer Aufgaben nicht benötigen (z.B. sich an andere Anwendungen anhängen oder die Festplatte zu formatieren.). Wie so oft ist in der Praxis leider nicht alles so rosig wie in der Theorie. Wie die Diskussion in diesem Beitrag gezeigt hat, war es auch im Falle von Java nicht einfach das theoretische Model in die Praxis umzusetzen. Wie auch für ActiveX gibt es auch für Java eine Reihe von bekannten Problemen (sogenannte Hostile Applets) die zur Gefährdung der Sicherheit eines Systems führen können. Die meisten dieser Hostile Applets verwenden jedoch Fehler in der Implementierung der JVM um ihre Attacken durchzuführen. Die strukturellen Schwächen des Java Sicherheitsmodells sind bisher nur indirekt für Attacken mißbraucht worden. Das Java Sicherheitsmodell wird außerdem ständig von mehreren Stellen (z.B. Secure Internet Programming Group, Princeton University) geprüft und nach Fehlern untersucht. Die meisten bekannten Probleme mit dem Sicherheitsmodell und den Implementierungen der JVM sind so auch von diesen Gruppen entdeckt worden, und konnten behoben werden bevor sie von Angreifern ausgenutzt werden konnten. Entgegen seines zum Teil schlechten Rufs hat es wenige bekannt gewordene, auf Java basierende, Angriffe gegeben.

Digitale Signaturen werden in der Zukunft mit Sicherheit eine große Rolle spielen. Ein entscheidendes Problem, das hier noch gelöst werden muß, ist das Problem, eine angemessene PKI zur Verfügung zu haben. Innerhalb einer Organisation zum Einsatz von Java auf dem Intranet können digitale Signaturen schon heute sinnvoll eingesetzt werden. Zum weltweiten Einsatz auf dem Internet muß noch eine geeignete Infrastruktur geschaffen werden.

Abschließend läßt sich nur sagen, daß, wie in allen Bereichen der IT-Sicherheit, vor der Entscheidung über den Umgang mit einer Technologie eine detailierte Analyse von Bedrohung und Nutzen des Einsatzes stehen muß. Die derzeitige Entwicklung läßt vermuten, daß das simple Abschalten von Applets/Controls nicht mehr lange akzeptabel ist.

Literatur

[LUC_98] Norbert Luckhardt, Nicht ganz dicht, c't 7/98.

[EIRO_88] Eichin, Rochlis, With Microsscope and Tweezers: An Analysis of the Internet Virus of 1988, 1988 IEEE.

[GRFE_96] McGraw, Felten, Java Security Hostile Applets, Holes and Antidotes, Wiley Computer Publishing, 1996.

[YELL_96] Yellin, Low Level Java Security, Sun Microsystems, 1996.

[CERT1] CERT(SM) Advisory CA-96-05, Java Implementations Can Allow Connection to an Arbitrary Host.

[CERT2] CERT(sm) Advisory CA 96.07, Weaknesses in Java Byte Code Verifier.

[LADU_97] LaDue, Security Threats from Deviant Java Byte Code, 1997.

[CHBE_94] Cheswick, Bellovin, Firewalls an Internet Security, Addison-Wesley, 1994.

[FRMU_96] Fritzinger, Mueller, Java Security, 1996, Sun Microsystems, Inc.

[GOMU_98] Gong, Mueller u.a., Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2, JavaSoft 1997.

[FELT_98] Princeton Classloader Attack, July 1998.
http://www.cs.princeton.edu/sip/History.html

[CHAP97] D.Chappell, D.S. Linthicum, ActiveX Demystified, Byte Sept. 1997.