Studienarbeit:

“Netzwerkprogrammierung mit Java2: Risiken und Lösungsansätze”

von:

André Luerssen

betreut durch:

Prof. Dr. Klaus Brunnstein

10. Februar 2001

Hiermit erkläre ich meine Studienarbeit selbständig durchgeführt zu haben.
Für diese Arbeit habe ich keine anderen, als die im Anhang angegebenen, Hilfsmittel und Quellen benutzt.

10. Februar 2001 André Luerssen

Inhaltsverzeichnis:

1 Einleitung

Diese Studienarbeit beschäftigt sich mit den Risiken der Netzwerkprogrammierung mit Java2 und soll, so weit wie möglich, Lösungsansätze aufzeigen.
Strukturell sollen die Probleme anhand einer bottom-up Sicht erläutert werden.

Angefangen mit Grundlagen der Netzwerkprogrammierung, allgemeinen Risiken und der Kryptographie wird dann die Sprache Java mit ihrer Entwicklung und ihren Konzepten vorgestellt. Insbesondere wird auf „hostile applets“ und Java-Viren eingegangen, da deren Technologie hinweise auf Sicherheitslücken gibt. Die Anwendung von Java in der Socketprogrammierung folgt. Diese Arbeit endet schließlich mit Empfehlungen zu einem möglichst sicheren Umgang mit dieser Sprache.
Es handelt sich durchgehend um eine low-level Sichtweise der Probleme. Die Darstellung höherer Konzepte wie RMI, Corba und Enterprise-Java-Beans waren erst geplant, wurden aber mit Rücksicht auf den Umfang dieser Arbeit wieder verworfen.
Entgegen dem Titel wird auch auf Java1 eingegangen, da man aus den Fehlern der ersten Version lernen kann und um auf konzeptuelle Probleme einzugehen. Auf Java Script wird ganz verzichtet, weil es außer dem Namen nichts mit Java zu tun hat.

Meine persönliche Motivation bestand darin, daß die Entwicklung verteilter Systeme an sich schon sehr komplex und schwer beherrschbar ist und ich nach Lösungen suchte potentielle Probleme bei meiner Entwicklung zu minimieren.
Java wird nachgesagt, es sei eine „sichere“ Programmiersprache und das sie durch das Konzept der Sandbox das lokale System schützt, so daß man Java z.B. im WWW zur Generierung von aktiven Inhalten (Applets) nutzt.
Ebenfalls besitzt Java eine sehr umfangreiche und übersichtliche Klassenbibliothek, so das es mir als guter Kandidat für meine durchzuführende Programmieraufgabe erschien.
Ich wollte jedoch erst mit meiner Spezifikation des Systems beginnen, wenn ich die Vor- und Nachteile von Java genau untersucht habe.
Wie ich bei meinen Untersuchungen feststellte, ist neben den klassischen softwaretechnischen Problemen, die Java z.T. gut löst, die Komplexität und ihre Beherrschbarkeit das größte Problem.
So werde ich mich im Rahmen meiner Diplomarbeit mit höheren Konzepten wie RMI, Corba und natürlich dem Java2-Enterprise-Modell beschäftigen.

2 Internetprotokolle

In diesem Kapitel sollen nur kurz die notwendigen Kenntnisse über Interprotokolle vermittelt werden. Dabei werden die Protokolle nur kurz beschrieben. Eine ausführliche Behandlung findet der Leser in den jeweiligen RFC´s.

2.1 OSI Architektur

Das Architekturmodell der Internationalen Standard Organisation (ISO) für offene Netze (Open System Interconnection, OSI) bezweckt die komplizierte Interaktion zwischen offenen Systemen übersichtlicher zu machen, um hierdurch den Entwurf und die Implementierung offener Systeme zu erleichtern [Kerner 92].
Die Protokolle des Internets (aus dem ARPA - Net entstanden) lassen sich in dieses System grob einordnen.

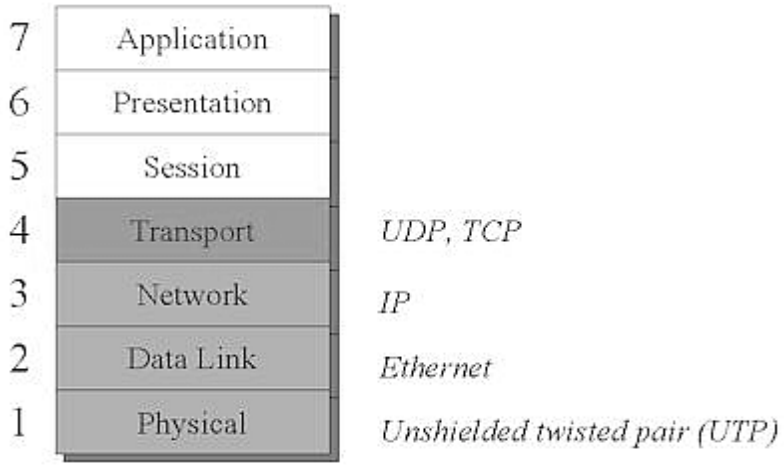


Abbildung 1 OSI - Architekturmodell [Baccala 97]

2.2 Client – Server Modell

Das Client-Server Modell ist ein wichtiges Modell für verteilte Systeme. Es beinhaltet eine Reihe von Server Prozessen, wobei jeder Prozeß als ein Ressource Manager, für die Ressourcen eines bestimmten Typs, handelt. Das Modell beinhaltet eine Reihe von Client Prozessen, die dem jeder Prozeß einen Task durchführt, der den Zugriff auf eine geteilte Hardware- oder Softwareressource benötigt. Ressource Manager können auch Zugriff auf eine fremde Ressource benötigen, so das auch sie zu Client Prozessen werden, d.h. Server Prozesse können auch Client Prozesse beinhalten.
Diese Prozesse (Programme) kommunizieren über ein entsprechendes Protokoll miteinander.

2.3 Objektorientierte Sichtweise

Ein verteiltes - objektorientiertes System ist eine Sammlung von Objekten, welches Dienstnutzer (Clients) und Diensterbringer (Server) durch eine definierte gehaspelte Schnittstelle trennt, d.h. die Clients sind von der Implementierung der Dienste, Datenrepräsentation und ausführbaren Code getrennt.
In dem objektbasierten System senden die Clients Nachrichten an ein Objekt, welches die Nachricht interpretiert und dann entscheidet welcher Dienst zu erbringen ist.
Diese Dienst- oder Methodenauswahl kann entweder in dem Objekt selbst oder in einem Broker (siehe RMI oder Corba) geschehen.
Die „Clientobjekte“ rufen also Methoden von „Serverobjekten“, welche dann einen entsprechenden Wert zurückliefern.

2.4 Network – Layer: IP

Das Internet-Protokoll IP [RFC 791] dient zur Übertragung von Datenpaketen. Es ist in etwa in Schicht 3 des OSI – Modells angeordnet. Die Aufgaben liegen in der Adressierung und der Fragmentierungen von Datagrammen. Es handelt sich um ist einen verbindungslosen Dienst.

2.5 Transport – Layer: TCP und UDP

Das Transmission Control Protocol [RFC 793] liegt über dem IP - Protokoll und erweitert es. TCP arbeitet verbindungsorientiert und bietet zusätzlich noch eine Flußkontrolle.
Das User Datagram Protocol [RFC768] ist das verbindungslose Gegenstück zu TCP. Bei UDP kommt es daher zu Vertauschungen oder auch zu Verlusten von Paketen.

Die Daten werden in einzelne Pakete gesplittet, die u.a. die Empfängeradresse, die sogenannte IP-Zieladresse, im Datagramm mitführen. Die IP-Adresse setzt sich aus der Netzwerknummer des angeschlossenen Netzwerkes und der Hostnummer für den entsprechenden Rechner innerhalb des Netzes zusammen. Dann werden die Datenpäckchen im Internet von Rechner zu Rechner weiter vermittelt und beim Empfänger wieder zusammengesetzt.

Dieses Verfahren nennt man Transmission Control Protocol (TCP). Das Transmission Control Protocol stellt einen zuverlässigen Datenübertragungsservice zwischen zwei Netzwerkrechnern dar. Im TCP-Protokoll werden alle zu übertragenden Datensegmente mit einer Sequence Number versehen, um Pakete, die aufgrund unterschiedlicher Übertragungswege vertauscht wurden, wieder in die richtige Reihenfolge zu bringen und um verlorengegangene Pakete erneut anfordern zu können.

Das TCP-Protokoll ist ein verbindungsorientiertes, zustandsstabiles Datenübertragungsprotokoll, d.h. zwei Netzwerkanwendungsprogramme bauen eine Verbindung auf und übertragen darüber ihre Daten. Erst wenn die beiden Programme keinen Kontakt mehr wünschen, wird die Verbindung wieder aufgegeben. Bevor mit der eigentlichen Datenübertragung begonnen werden kann, wird ein sogenanntes 3-Weg Handshake ausgeführt. Im ersten Datensegment schickt der Rechner, der die Verbindung aufbauen will, die erste Sequence Number und setzt das Synchronize Sequence Number (SYN) Bit.

Der die Verbindung annehmende Rechner antwortet mit einem Datensegment, das seine erste Sequence Number beinhaltet und in dem zusätzlich zum SYN - Bit das Acknowledgment (ACK) Bit gesetzt ist. Der initiierende Rechner sendet nun ein Datensegment, in dem nun das ACK-Bit gesetzt ist, um den Erhalt des zweiten Datensegmentes zu bestätigen.

Eine Aufgabe des TCP - Protokolls besteht in der Überprüfung fehlerhaft übertragener Daten durch die Berechnung der Prüfsumme und der daraus folgenden Anforderung einer erneuten Übermittlung eines fehlerhaften Datenpakets. Durch dieses TCP/IP-Protokoll entfallen komplizierte Verbindungsauf- und abbauprozeduren, aber die Datenpakete werden durch die langen Datagramme auch deutlich größer. Folgende Header- und Metainformationen, sind zu jedem Paket im Datagramm gespeichert:

- IP – Ziel und Quelladresse
- IP – Optionen
- Protokollart (z.B. TCP, UDP (User Datagramm Protocol) oder ICMP (Internet Control Message Protocol))
- ICMP – Nachrichtentyp
- Bei TCP - Paketen Angaben über den Verbindungsaufbau (ACK - Bit)

Metainformationen sind solche Daten, die zwar nicht in den Datenpaketen stehen aber dem Router trotzdem bekannt sind. Dies wäre z.B. die Schnittstelle, über die ein Paket eintrifft oder weitergeleitet wird. Obwohl Pakete die Quelladresse im Datagramm haben, enthält sie keine Informationen darüber, von wem die Daten kommen. Dies erschwert die Suche nach eventuellen Angreifern.

2.6 Internetadressen, Namenvergabe und Routing

Im Internet werden die Host über Namen adressiert, die von sogenannten Domain Name Systems DNS [RFC1034], [RFC1035] zu 32-bit IP - Adressen (bei Ipv4, bei dem neuen Ipv6 [RFC 1883], [RFC 1884] sind es 128-bit Adressen) aufgelöst werden. Anhand dieser Adressen und einem Routing-Algorithmus werden die Pakete, in der IP-Schicht, von so genannten Routern „geroutet“, d.h. an das Ziel weitergeleitet.

3 Kryptographie

3.1 Verschlüsselung

Um Informationen vor dem Zugriff durch unbefugte Benutzer zu schützen, gibt es zwei Methoden [Janzen 99]: access-path control und data encryption. Dabei beruht die moderne Kryptographie (data encryption) auf der Existenz algorithmischer Probleme (NP), für die es nur sehr langsame Algorithmen gibt, die man praktisch nicht benutzen kann.
Eine Nachricht in Klartext (plaintext) wird mit Hilfe einer Funktion (Schlüssel, Key(s)) verschlüsselt (encrypts) und man erhält die verschlüsselte Nachricht (cryptotext, o.a. ciphertext). Der Empfänger der Nachricht entschlüsselt (decrypts) die Nachricht ebenfalls mit einem Schlüssel, dabei unterscheidet man zwei Arten von Verschlüsselung:

Die symmetrische Secret-Key-Verschlüsselung (z.B. AES),
bei der ein geheimer Schlüssel zur Verschlüsseln und Entschlüsseln der Nachricht verwendet wird. Diese Variante setzen häufig Programme ein, die Daten verschlüsseln, die beim Anwender bleiben. Soll die codierte Datei jedoch weitergegeben werden, muß dem Empfänger das Paßwort auf einem sicheren Übertragungsweg mitgeteilt werden. Da es neben dem Gespräch unter vier Augen keine wirklich sichere Methode gibt, wird dies zum Problem, welches die asymmetrische Public-Key-Methode umgeht.

Die asymmetrische Public-Key-Verschlüsselung (z.B. RSA),
bei der zwei Schlüssel, und zwar ein öffentlicher und ein privater (geheimer) Schlüssel, verwendet werden. (Das können dieselben Schlüssel sein, die bei der digitalen Signatur zum Einsatz kommen.) Der öffentliche Schlüssel ist jedem zugänglich, der geheime nur dem Teilnehmer. Außerdem können mit dem öffentlichen Schlüssel die Nachrichten verschlüsselt und nur mit dem geheimen entschlüsselt werden. Der Sender codiert seine Nachricht mit dem öffentlichen Schlüssel des Empfängers. Eine so verschlüsselte Nachricht läßt sich dann nur mit dem privaten Schlüssel des Empfängers wieder entschlüsseln.

3.2 Einweg – Hashfunktionen

Hashfunktionen kommen in einer Vielzahl von Anwendungen zum Einsatz. Ziel einer Hashfunktion ist es allgemein, den i.d.R. unendlichen Definitionsbereich möglichst "gleichmäßig" auf den Wertebereich abzubilden. Primitive Hashfunktionen wären zum Beispiel die "Modulo"-Funktion oder die Quersumme einer Zahl; für Textdaten wird gerne und oft die fast ebenso simple XOR-Funktion benutzt, die auf alle Zeichen des Textes hintereinander angewendet, genau ein Zeichen als Resultat liefert, unabhängig von der Länge des Textes.

Hashfunktionen können auch als Prüfsummen eingesetzt werden. Eine Prüfsumme hat zwei wichtige Kriterien zu erfüllen:

- Es soll sehr unwahrscheinlich sein, daß zwei Datenmengen "zufällig" die gleiche Prüfsumme ergeben.

- Es soll sehr schwierig sein, zu einer vorgegebenen Prüfsumme einen passenden Text zu finden, selbst dann, wenn der Algorithmus bekannt ist.

Die erste Forderung wird um so besser erfüllt, je größer der Umfang der Prüfsumme ist und je "gleichmäßiger" die benutzte Hashfunktion ist. Eine Funktion, die beispielsweise ein komplettes Buch auf eine 512-Bit-Prüfsumme abbildet, wird diese Forderung i.d.R. sehr gut erfüllen, denn bei 2^{512} (2 hoch 512) Möglichkeiten ist die Wahrscheinlichkeit einer gleichen Abbildung gering.
Aus der zweiten Forderung ergibt sich auch die Notwendigkeit, daß eine kleine Änderung im Urbild eine große (d.h. nicht direkt nachvollziehbare) Änderung im Funktionswert nach sich ziehen sollte. Wäre das nicht der Fall und könnte man beispielsweise einzelne Bits des Funktionswertes direkt durch Änderungen am Urbild beeinflussen (wie das bei der XOR-Funktion der Fall ist), so wäre es ein leichtes, zu einem gegebenen Funktionswert ein geeignetes Urbild zu erstellen.
Weitere Beispiele für Hashfunktionen sind MD5, MD4 und SHA.

3.3 Signaturen

Unterschriften waren seit jeher ein wichtiger Bestandteil für die Gültigkeit von Dokumenten. Das Prinzip dabei ist, daß jeder Mensch eine Unterschrift hat, die sich von allen anderen unterscheidet und deren Fälschung schwer ist. Wenn ich, zum Beispiel, ein Dokument unterschreibe, bin ich gesetzlich für das unterschriebene Dokument belangbar.

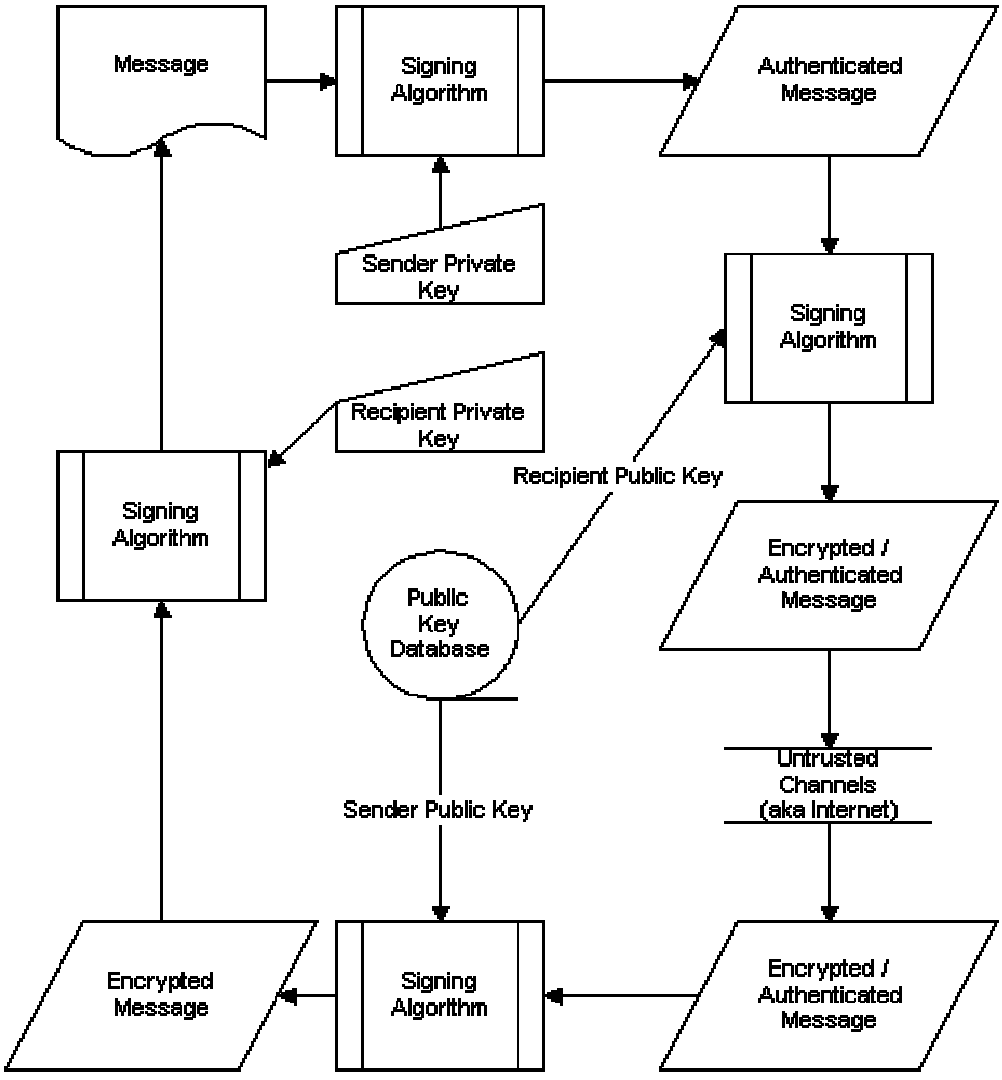
In der Kryptographie gibt es eine analoge Vorgehensweise zur Unterschrift, genannt Signatur. Sie basiert auf den schon erwähnten public key Verfahren. Nachdem bei den public key Verfahren ein private key existiert, der eindeutig einer Person oder Institution zugeordnet werden kann, können Operationen mit dem private key ebenfalls eindeutig dieser Person oder Institution zugeordnet werden. Zu jedem private key existiert wiederum ein public key, der allen zugänglich ist. Somit sind Operationen mit dem public key allen möglich, die diesen besitzen.

Ein Signaturalgorithmus sieht zwei Operationen vor, die auf ein Dokument M gemacht werden können. Derjenige, der M signiert, führt mit seinem private key p Operationen mit M aus, deren Ergebnis die Signatur S von M ist: S = Sign(M, p)

Die zweite Operation betrifft die Überprüfung einer Signatur. Wer im Besitz des public keys q des Signierenden ist, kann verifizieren, ob er das Dokument M signierte. Er führt dazu eine Operation Ver(S, M, q) aus, welche die Signatur verifiziert.

3.4 Zertifikate

Zertifikate sind vergleichbar mit Bürgschaften, in denen jemand für die Vertrauenswürdigkeit eines anderen bürgt. Wenn derjenige, der bürgt, vertrauenswürdig ist, wird dem, für den er gebürgt hat, ebenfalls vertraut.
Auf die Computerwelt umgesetzt stellt der Bürge ein Zertifikat aus. Ein Zertifikat ist eine Datenstruktur, in der der public key q des Zertifikatbesitzers (für den gebürgt wird) enthalten ist. Des weiteren enthält es die Signatur des public keys, welche vom Zertifikataussteller (Bürgen) erzeugt wird. Der Zertifikatbesitzer kann das Zertifikat dazu verwenden, daß seinem öffentlichen Schlüssel vertraut wird, er kann sich mit diesem Zertifikat authentifizieren.
Wenn der public key des Zertifikatausstellers bekannt ist, kann verifiziert werden, ob das Zertifikat des Zertifikatbesitzers eine gültige Signatur des Zertifikatausstellers enthält. Dabei wird die Signatur von q mit dem public key des Zertifikatausstellers überprüft.
Es genügt die Vertrauenswürdigkeit eines public key zu garantieren, da der öffentliche Schlüssel in public key Verfahren dazu verwendet wird, mit dem Besitzer des private keys zu kommunizieren bzw. eine Signatur desselben zu akzeptieren. Vertraut man also seinem public key, so traut man automatisch auch ihm.
Ein System, bei dem sich eine große Anzahl von Benutzern mit Hilfe von Zertifikaten authentifizieren kann, wurde durch das X509 Protokoll verwirklicht.



4 Firewall

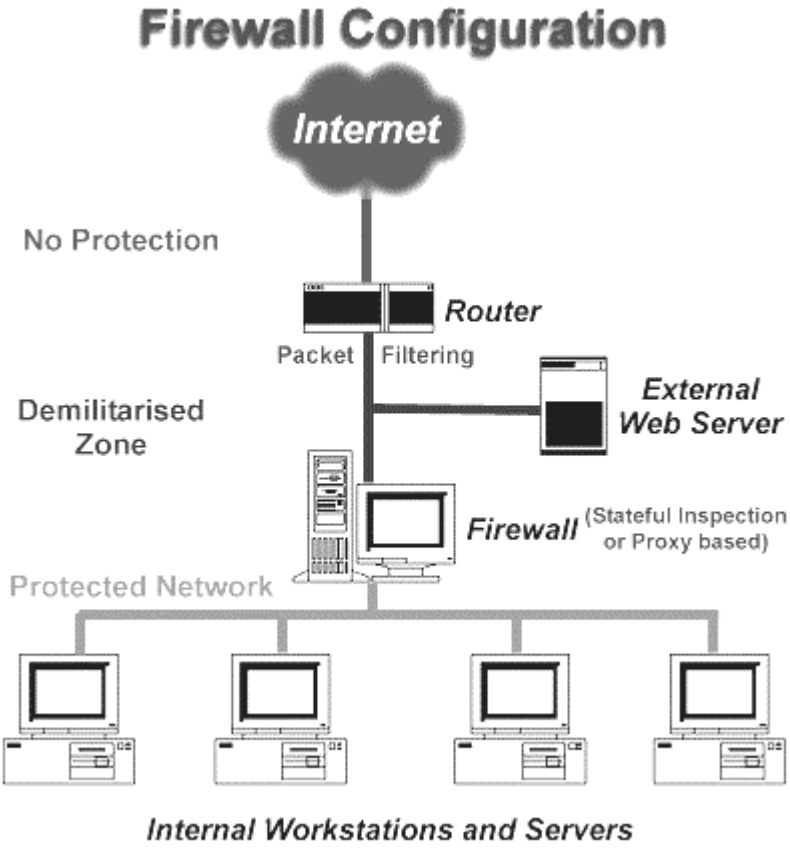


Abbildung 2 Firewall [firewall1]

Wenn ein lokales Netz (LAN) an das Internet angeschlossen wird, setzt man es der Gefahr aus, das Zweite von außen auf das LAN Zugriff erhalten, der nicht erwünscht ist. Um diesen möglichen Zugriff abzuwehren, installiert man eine Firewall. Dies ist eine Architektur aus Hardware- und Softwarekomponenten, die zwischen das Internet und das Intranet gelagert wird. Die Firewall sichert das LAN, indem es den Zugang kontrolliert. Auf diese Weise können Angriffe, also unerlaubte Zugriffe, abgewehrt werden.

Eine Firewall ist eine Gruppe zusammen wirkender Programme, die sich auf einem Gateway befinden, welche den Übergang von einem privaten Netz (Intranet) zu einem externen Netz (Extranet) schützen soll. Technisch betrachtet ist eine Firewall aus einem Router, der jedes Paket mit seiner Herkunfts- und Zieladresse überprüft, und einem Proxy, welches das die Anfragen anstelle der Intranethosts stellt, bzw. beantwortet, zusammengesetzt.

Abhängig von der gewählten Architektur besteht eine Firewall aus mehreren Komponenten. Während bei der Screening Router Architektur das LAN durch Vermittlungsrechner (Screening Router oder kürzer Router) gesichert wird, kombiniert die Screened Subnet Architektur vor- und nachgeschaltete Router mit der Screened Host Architektur, bei der ein Rechner mit spezieller Sicherungssoftware installiert wird. Durch das Hintereinanderschalten von unabhängigen Komponenten, die unterschiedliche Filterstrategien realisieren, wird ein erheblich höheres Sicherheitsniveau geschaffen. Ein Nachteil bei dieser Architektur besteht in den fehlenden Protokollierungsmöglichkeiten der Router. Dadurch fehlen wichtige Protokollinformationen, die beim Erkennen und Zurückverfolgen von Angriffen helfen würden.

Unter einem Screening Router versteht man einen Rechner, der zwei Netzwerke mittels Paketfilterung "sicher" verbindet. Er stellt die billigste aber auch die unsicherste Firewallarchitektur dar. Billig, weil ein Router bei jeder Netzwerkanbindung schon vorhanden ist und weil auf jedem Router Berechtigungs- und Verweigerungslisten zur Paketfilterung installiert werden können. Unsicher, weil Paketfilterung allein keine nennenswerte Hürde für einen Angreifer darstellt. IP-Spoofing ist z.B. eine Angriffsmethode, gegen die ein Screening Router keinen Schutz bietet. Auch bietet ein Router keine Möglichkeit der Protokollierung des IP- Verkehrs. Das macht es sehr schwierig einen Angriff zu erkennen oder gar zurückzuverfolgen.

Paketfilterung ist ein Netzwerk-Sicherheitsmechanismus, der überprüft, welche Pakete an ein Netz und aus einem Netz weitergereicht werden dürfen und wohin sie weitergeleitet werden. Dazu werden die Quell- und Zieladressen in den Datenpaketen verwendet. Man kann auch mittels Paketfilterung Pakete bestimmter Protokolle zulassen oder blockieren. Allerdings kann man nicht innerhalb eines Protokolls Verbindungen von bestimmten Benutzern zulassen oder blockieren, da Pakete keine Informationen darüber enthalten, von welchem Benutzer sie kommen. Der wesentliche Vorteil von Paketfilterung ist, daß man von einer einzigen Stelle aus detaillierte Schutzmaßnahmen ergreifen kann, die im ganzen Netzwerk wirksam sind. Gewisse Schutzmechanismen können ausschließlich durch Router mit Paketfilterung bereitgestellt werden und dies auch nur, wenn sie an bestimmten Stellen im Netzwerk eingesetzt werden. Ein Router an der Grenze zwischen dem LAN und dem Internet kann zum Beispiel alle Pakete aus dem Internet zurückweisen, wenn als Quelladresse im Paket eine interne Adresse angegeben ist. Dies deutet auf einen Adressbetrug hin. Ein wesentlicher Vorteil von Paketfilterung ist, daß sie kein Anwenderwissen und keine Mitarbeit von Benutzern erfordert, da keine Anpassung der Software erforderlich ist.
Paketfilterung hat allerdings auch einige Nachteile. Einer davon ist, daß Paketfilterungsregeln oft schwer formulierbar und einmal ausformuliert, auch schwer zu testen sind. Da Pakete zwar angeben, zu welchem Port sie gehören, nicht aber zu welcher Anwendung, muß man, wenn man Einschränkungen in bezug auf höhere Protokolle als IP festlegt, dazu Portnummer verwenden.

Die einfachste und zugleich unsicherste Form der Sicherung eines LAN ist der generic direct service. Die Datenpakete werden mittels Paketfilterung im äußeren und inneren Router auf deren Zugangsberechtigung überprüft. Aufgrund dieser Filterung erhalten nur die Pakete Zugang zum Intranet, die in den Routern als "sicher" erkannt worden sind. Der Client wird demnach direkt mit dem Anwendungsserver verbunden und ist nur mittels Paketfilterung geschützt. Wenn man das LAN direkt mit dem Internet verbindet, ist es vollkommen ausreichend, die Filterungsregeln auf nur einem Router zu installieren, da sie in den meisten Fällen sowieso identisch sind.

Der Bastionhost fungiert als die zentrale Vermittlungsstelle zwischen dem Internet und dem LAN. Auf ihm wird spezielle Software, sogenannte Proxy Server, installiert, die die Verbindungen aufnehmen und an die jeweiligen Original Server weiterleiten. Wenn man proxying unterstützt, d.h. wenn man also Proxy Server für die unterschiedlichen Dienste bereitstellt, so erhält man auf dem Bastionhost einen Engpaß, an dem man Verbindungen besser kontrollieren kann.
Da ein Bastionhost am ehesten einem Angriff aus dem Internet ausgesetzt ist, muß er so einfach wie möglich aufgebaut sein. Je einfacher der Aufbau strukturiert ist, desto einfacher ist er auch zu schützen. Nur die absolut notwendigen Dienste sollten auf dem Bastionhost laufen, da durch Software- oder Konfigurationsprobleme dieser Dienste Sicherheitslücken entstehen könnten. Auf dem Bastionhost sind also nur solche Dienste einzusetzen, die man entweder im Internet anbieten möchte und/oder die für den Zugang zum Internet notwendig sind. Von diesen Diensten können diejenigen, die als sicher eingestuft werden können, über Paketfilterung realisiert werden. Da jeder auf dem Bastionhost angebotene Dienst zu Sicherheitsproblemen führen kann, sollten nicht benötigte Dienste unbedingt deaktiviert werden. Eine weitere Sicherheitsmaßnahme ist das Sperren der Benutzerports des Bastionhosts, also Ports mit einer Nummer größer als 1023. Somit kann man von vornherein einen erfolgreichen Angriff auf die Benutzerports ausschließen.

Ein Proxy Server ist ein Programm, das stellvertretend für interne Clients mit externen Servern kommuniziert (proxy (engl.) - Stellvertreter, Vollmacht). Proxy Clients unterhalten sich mit Proxy Servern, die bestätigte Client-Anfragen an die eigentlichen Server weiterleiten und die Antworten zurück an die Clients übermitteln.

Beim generic proxy service wird zwischen dem LAN und dem Internet ein Unternetz aufgebaut, das sogenannte Perimeter Netz. An dieses Subnet wird ein weiterer Rechner angeschlossen, der Bastionhost. Auf dem Bastionhost werden Proxy-Server installiert, so daß dieser als zentraler Kontaktpunkt zwischen LAN und Internet fungieren kann. Der Client verbindet sich mit dem Proxy-Server, welcher dann Kontakt mit den eigentlichen Servern im Internet aufnimmt. Infolgedessen kommt es zu keiner direkten Verbindung zwischen LAN und Internet (das Bastionhost enthält zumeist auch zwei Netzwerkkarten). Dies hat den Vorteil, das wenn ein Angriff stattfindet, wird "nur" der Bastionhost getroffen.

5 Risiken zeitgenössischer IT – Systeme

5.1 Angriffsarten

5.1.1 Denial of Service (DoS)

DoS Attacken bezwecken das „Ausschalten“ von Diensten. Neuerdings beobachtet man sogenannte DDoS-Angriffe. DDoS ist eine Abkürzung für "Distributed Denial of Service" ("denial": Ablehnung, Leugnung). Im Februar 2000 wurden verschiedene, große Internet-Dienste (wie z.B. Yahoo, CNN, Amazon, eBay, ETrade) durch DDoS-Attacken lahm gelegt. Hierbei hatten sich die Angreifer Zugang zu hunderten von Rechnern im Internet verschafft (darum das "distributed"), um die Wirksamkeit ihrer Attacken durch die Vielzahl der gleichzeitig angreifenden Rechner stark zu erhöhen.

Die beobachteten Angriffe basierten auf zwei wesentlichen Schwachstellen:
Zum einen konnten die Absenderadressen der "angreifenden" Datenpakete gefälscht werden (IP-Spoofing), zum anderen konnten vor den eigentlichen Angriff auf einer großen Anzahl Dritter - nur unzureichend geschützter - Internet-Rechner unberechtigter Weise Programme installiert werden, die dann ferngesteuert durch massenhaft versendete Datenpakete den eigentlichen Angriff ausführten.

Das besondere an diesen DDoS-Angriffen ist, daß diese daher auch diejenigen treffen können, die sich ansonsten optimal vor Eindringlingen aus dem Internet geschützt haben. Insofern sind Rechner, auf denen noch nicht einmal sogenannte Grundschutzmaßnahmen umgesetzt sind, nicht nur für den jeweiligen Betreiber eine Gefahr, sondern auch für alle anderen Rechner im Internet.

Wirksame Maßnahmen gegen verteilte Denial-of-Service-Angriffe müssen in einer konzertierten Aktion an vielen Stellen in der vorhandenen komplexen Internetstruktur getroffen werden. Serverbetreiber im Internet, die Ziel der genannten Angriffe waren, können eine Reihe von sinnvollen Maßnahmen ergreifen, aber das DoS-Problem nicht vollständig lösen. Vielmehr müssen verschiedene Zielgruppen (Inhalte-Anbieter, Serverbetreiber, Netzvermittler und Endanwender) - jeder in seinem Bereich - tätig werden. Nur gemeinsam kann das Internet im Hinblick auf die Gefährdung durch DoS-Angriffe sicherer gemacht, die Durchführung von Denial-of-Service-Angriffen erschwert sowie eine spätere Verfolgung der Urheber dieser Angriffe erleichtert werden.

5.1.2 Man in the middle

Unter dem „Mann in der Mitte“ versteht man alle Arten von Angriffen, bei denen sich ein Dritter eine Verbindung zwischen zwei Kommunikationspartnern schaltet. Er kann die Kommunikation der beiden belauschen oder auch manipulieren. Der Angriff geschieht folgendermaßen: Ein Client nimmt zu einem Server Kontakt auf. Dies beobachtet ein Dritter, den Client und Server nicht bemerken, und wartet auf die Antwort vom Server. Die Antwort des Servers wird vom Angreifer abgefangen und gegebenenfalls durch seine eigene (verfälschte) Anwort ersetzt.

5.1.3 Spoofing

Beim Spoofing täuscht man eine andere Identität vor. Man unterscheidet z.B. IP-Spoofing (fälschen der IP-Adresse) und DNS-Spoofing (vortäuschen eines fremden DNS-Servers und Verfälschen des Name Service Caches eines Hosts mit der Folge: Authentizitätsverlust!)

5.1.4 Tigerteam

Ein Tigerteam (U.S. military jargon) oder auch Exploitteam genannt, ist ein Team von Hackern, das von einer Organisation angeheuert wird, um dessen Sicherheitsmaßnahmen zu testen. Ein erfolgreicher Test ist jedoch keine Garantie für Sicherheit, Sicherheit ist eine Systemeigenschaft und wird bei der Spezifikation hergestellt.

5.2 Klassische Netzwerkangriffe

5.2.1 ICMP

Das Internet Control Message Protokoll [RFC 792] definiert IP Pakte, die dem Austausch von Nachrichten, über Netzzustand, dienen. Diese Pakete können dazu benutzt werden um eine fremde Verbindung abzubrechen, dabei besteht der Header jedes ICMP-Paket aus drei Feldern: Type, Code und Checksum.

Paket-Typ: „Destination Unreachable"
Normalerweise wird diese Art von Paket an den Sender gesendet, wenn die IP-Schicht auf dem Zielrechner oder ein Gateway ein Nachricht nicht zustellen kann.
In [Bellovin 89] wird gezeigt, wie ein Angreifer eine fremde TCP - Verbindung (unter Kenntnis der IP-Adressen und Portnummern, der Opfer) abbrechen kann.

Paket-Typ: "Time Exceeded"
und Code = 0 : der TTL-Zähler (TTL = TimeToLive) eines Datagramms ist gleich 0
oder Code=1: nicht alle Fragmente eines fragmentierten Paketes sind rechtzeitig am Zielrechner eingetroffen.
Auch dieser Pakettyp kann dazu benutzt werden eine fremde Verbindung zu beenden.

Paket-Typ: „Redirect"
Mit Hilfe dieses Pakettyp ist es möglich Pakete auf einen anderen Gateway umzuleiten. Da nur der erste Gateway in einer Verbindung dieses Signal senden darf, ist diese Attacke nicht ganz einfach. Sie wird auch in [Bellovin 89] beschrieben.

Paket-Typ: „Echo"
Erhält ein Host eine Echo-Aufforderung, so muß er diese an den Absender zurückschicken. Beim „ Ping of Death" ist es möglich (spez. bei Windows 95) ein Echo-Paket beliebiger Länge abzuschicken. Dies brachte früher viele Unix-Rechner zum abstürzen, da ihr TCP/IP Stack falsch implementiert war. Dieser Fehler ist bei den meisten Systemen inzwischen behoben worden.
Fälscht ein Angreifer die Absenderadresse seiner Echo-Pakete, in die eines anzugreifenden Host im Subnetz, und sendet diese an eine Broadcast-Adresse dieses Subnetzes, so bricht der Host höchst wahrscheinlich unter der Last der Antworten zusammen. Diesen Trick nennt man „Smurfing" [Huegen 98].

5.2.2 TCP/IP

5.2.2.1 SYN flooding

TCP hält für jeden Port eine Warteschlange für noch nicht bearbeitete Anfragen bereit. Ist diese Warteschlage voll, so werden keine Verbindungen mehr angenommen.
Der Angreifer sendet eine Folge von Paketen mit gesetztem SYN-Flag und gefälschter Quelladresse. Der angegriffene Rechner bekommt daher keine Antwort auf seine SYN/ACK-Pakete. So wird die Warteschlange gefüllt und es werden keine weiteren Verbindungen angenommen. Bis die Warteposition, nach einem Timeout, wieder freigegeben wird.

5.2.2.2 IP Spoofing

Beim IP-Spoofing gibt der Angreifer vor von einem Rechner zu senden, dem der angegriffene Host vertraut. Das insbesondere von den Gebrauch der rutil-Programme (rlogin, rsh, etc.) interessant ist.

5.2.2.3 Folgenummern

Jedes IP-Paket enthält eine Folgenummer. Beobachtet ein Angreifer eine Kommunikation in dem er z.B. am gleichen Ethernetstrang sitzt oder einen Router kontrolliert, über den die Verbindung läuft, kann er Pakte mit höheren Folgenummern einschleusen. Dies führt zur Desynchronisation der ursprünglichen Kommuntikationspartner. Und die Verbindung ist nun übernommen worden. Ein Beispiel für diesen Angriff wäre „Telnet-Hijacking“. Zwischen den ursprünglichen Kommunikationspartnern entsteht dann ein sogenannter ACK-Storm, der durch Resynchronisationsversuche entsteht.

5.2.3 DNS

5.2.3.1 DNS-Spoofing

Ein Angreifer versucht hier die Einträge in einem DNS-Server zu manipulieren, so das mögliche Opfer nicht auf dem gewünschten Server landen, sondern auf dem des Angreifers. Der Angreifer setzt also einen eigenen DNS-Server (in der Angriffsdomain) auf und versieht diesen mit falschen Einträgen. Nun stellt er eine Anfrage bei dem zu manipulierenden DNS-Servers nach einem Rechner in seiner Angriffsdomain. Dieser fragt dann bei dem eigenen DNS-Server nach und bekommt die falschen Einträge mit übermittelt. Diese Einträge befinden sich erstmal nur im Cache des angegriffenen DNS-Servers.
Ein ahnungsloser Bankkunde könnte so, anstatt bei seiner Hausbank zu landen, auf den Rechner des Angreifers umgeleitet werden.

5.3 Kryptoschwächen

5.3.1 HTTP

Die Http-Authentisierung wird nur mit dem Base64 verfahren verschlüsselt. Was eine nur sehr schwache Verschlüsselung darstellt!

5.3.2 SSL

Ein man-in-the-middle Angriff ist bei einer SSL-Verbindung, ohne Client- und Serverauthenifizierung möglich! Auf eine nähere Darstellung wird in dieser Arbeit verzichtet.

5.3.3 DES

Die Electronic Frontier Foundation (www.eff.org) begann 1997 mit ihren Untersuchungen über das Cracken von DES , um herauszufinden wie billig und einfach ein hardwarebasierter DES-Cracker konstruiert werden kann. Weniger als ein Jahr später gewannen sie den RSA DES Challenge II-2 mit einer Maschine, deren Preis unter 250.000 USD lag. Der DES-Code wurde in drei Tagen gebrochen, was bewies, das DES nicht besonders sicher ist.

5.4 Java Malware

Um die Verletzlichkeit des Java Sicherheitsmodell aufzuzeigen, seien hier einige Sicherheitslöcher (engl.: Vulnerabilities), und die Wege wie diese ausgenutzt (engl.: Exploits) werden können, erklärt. Exploits und aus diesen resultierenden maliziösen Code seinen kurz erklärt. Schließlich soll diese Arbeit keine Anleitung zu hacken/cracken geben. Maliziösen Code finden wir in Form von „hostile Applets“, Java Viren, und „Backdoors“ d.h. Trojanern.

5.4.1 Vulnerabilities

Verletzlichkeiten des Systems (Vulanerabilities) ergeben sich aus Fehlern im Entwurf, sogenannte Designfehler, die schwer zu beheben sind, und Implementationsfehler, die i.d.R. leicht zu beheben sind. Die Vulnerabilities; die im Jahr 2000 bekannt wurden, werden im Folgenden ausführlich erklärt. Dabei konzentriere ich mich nur auf Meldungen, die folgende Produkte betreffen: Sun JDK/SDK/JRE, Netscape Navigator und Microsoft Internet Explorer. Dabei differenziere ich nach Vulnerabilities mit/ohne Lösung (Solution), mit/ohne Exploits und nach Art des Fehlers (Design-/ Implementationsfehler /etc.). Hier werden Java1 und Java2 Fehler aufgezeigt.

Java Vulnerability Meldungen 2000:

31.01.2000
Microsoft Java Virtual Machine getSystemResource Vulnerability
Exploit: ja Solution: ja Typ: Access Validation Error
betrifft: Microsoft Internet Explorer 4.0/5.0 mit VM 2000,3100,3200
Quelle: [SecurityFocus bid 957]
Info:
Mit Hilfe der Funktionen: getSystemResourceAsStream() und getSystemResource() kann ein Applet oder eine RemoteAppikation auf die lokale Platte eines Users zugegriffen werden.

16.04.2000
Microsoft Internet Explorer for Macintosh java.net.URLConnection Vulnerability
Exploit: nein Solution: nein Typ: Design Error
betrifft: Microsoft Internet Explorer 4.5/5.0 Mac MRJ 2.1.4/MRJ 2.2/ VM
betrifft nicht : Microsoft Internet Explorer 4.5 Mac / MRJ 2.2
Quelle: [SecurityFocus bid 1209]
Info:
Mit Hilfe der Klasse URLConnection kann ein Applet eine beliebige Netzwerkverbindung aufbauen. Dies wird auch von getImage() gerufen, was das Java Sicherheitsmodell für Applets verletzt!

16.04.2000
Microsoft Internet Explorer for Macintosh getImage and classloader Vulnerabilities
Exploit: ja Solution: nein Typ: Boundary Condition Error
betrifft: Microsoft Internet Explorer 4.01/4.5/5.0 Mac
Quelle: [SecurityFocus bid 1339]
Info:
Mit Hilfe eines HTTP Redirekt und eines Applets können Bilder und Java Dateien ausgelesen werden.

19.04.2000
MS IE 5.01 JSObject Cross-Frame Vulnerability,
Exploit: ja Solution: nein Typ: Access Validation Error
betrifft: Microsoft Internet Explorer 5.01 und Internet Explorer for Unix 5.0
Quelle: [SecurityFocus bid 1121]
Info:
Das Cross-Frame Sicherheits Modell vom MS IE 5.01 kann mit einem Java Applet durchbrochen werden. Bekommt das Applet einen Parameter mit Javascript Code in der Form „javascript: *URL“* übergeben, so kann die setMember Methode der JSObject Klasse dazu benutzt werden den 'href' auf das DOM (Document Object Model) eines anderen Frames oder Windows auf den der URL setzen.

10.06.2000
Multiple Vendors java.net.URLConnection Applet Direct Connection Vulnerability
Exploit: ja Solution: nein Typ: Boundary Condition Error
betrifft: Microsoft Internet Explorer 4.01/4.5/5.0 Mac (und iCab)
Quelle: [SecurityFocus bid1336]
Info:
Das Sicherheitsmodell des Apple Mac OS Runtime Java (MRJ) wurde in der Funktion java.net.URLConnection ignoriert. So ist ein Zugriff auf beliebige Hosts möglich.

10.06.2000
Multiple Vendors HTTP Redirect Java Applet Vulnerability
Exploit: ja Solution: nein Typ: Boundary Condition Error
betrifft: Microsoft Internet Explorer 3.0/4.0/4.01 Mac (und iCab)
Quelle: [SecurityFocus bid1337]
Info:
Der Betreiber eine maliziösen Website kann eine Applet auf seine Seite setzen, daß HTTP-Redirect-Requests zu einem entfernten Host bearbeitet und von dort beliebige Dateien zurückleitet. Das Applet-Sicherheitsmodell wurde bei der Implementierung des Apple Mac OS Runtime Java (MRJ) ignoriert.

03.08.2000
Multiple Vendor Java Virtual Machine Listening Socket Vulnerability
Exploit: ja Solution: ja Typ: Origin Validation Error
betrifft: Microsoft Internet Explorer 4.0/5.0/5.01/5.5 mit VM 2000,3100,3200,3300 und Netscape Communicator 4.73/4.74/4.72/4.7/4.61/4.6/4.51/4.5/4.0/4.08/4.07/4.06/4.05/4.04
betrifft nicht: Netscape Communicator 6.0 für Win und Netscape Communicator 4.75
Quelle: [SecurityFocus bid1545]
Info:
Mehrere Fehler bei der Implementierungen verschiedener Java-Hersteller erlauben es einem maliziösen Applet „Listening“-Sockets zu erstellen, um Netzwerkverbindungen anzunehmen, die laut der Policy nicht erlaubt sind.
Der Trojaner Brown Orifice (s.u.)nutzt diese Sicherheitslücken!
Vor der Erstellung einer java.net.ServerSocket Klasse muß diese mit Hilfe der SecurityManager.checkListen() Methode prüfen, ob dies zulässig ist. Durch Implementierungsfehler wird eine SecurityException nicht ausgelöst.

Ist der Socket erstellt muß seine ServerSocket.accept() Methode gerufen werden, damit dieser Verbindungen entgegen nehmen kann. Oder durch Bildung einer Unterklasse und unter Benutzung der ServerSocket.implAccept() Methode eine eigene Implementierung der accept Methode. Die ServerSocket.accept() Methode ruft normalerweise SecurityManager.checkAccept(), um zu überprüfen ob der Server Verbindungen entgegen nehmen kann.
Die ServerSocket.accept() und ServerSocket.implAccept() Methoden nehmen beide erst einmal eine Verbindung entgegen um die entfernte IP und den Port zu bestimmen und dann gegebenenfalls die Verbindung, mit Hilfe der Socket.close() Methode und Auslösen einer SecurityException, die Verbindung zu beenden.
Da die ServerSocket.implAccept() Methode einen Socket als Argument und zur Verbindung eine maliziösen Klassen nimmt, die eine SecurityException ignoriert, kann dessen Methode close() überladen werden, so das die Verbindung nicht getrennt wird. Die maliziöse Klasse kann nun mit Hilfe des manipulierten Sockets, Verbindungen entgegen nehmen.
Sun Implementation der ServerSocket.implAccept() Methode hat die eine Sicherheitslücke durch rufen der Methode Socket.impl.close(), anstatt Socket.close(), geschlossen.
Weiteres zu diesem Thema ist in dem Kapitel über Brown Orifice zu finden.

03.08.2000
Netscape Communicator URL Read Vulnerability
Exploit: ja Solution: ja Typ: Access Validation Error
betrifft: Netscape Communicator
4.73/4.74/4.72/4.61/4.6/4.51/4.5/4.0/4.08/4.07/4.06/4.05/4.04
betrifft nicht: Netscape Communicator 6.0 für Win
Quelle: [SecurityFocus bid1546]
Info:
Ein Fehler in Netscape Communicator's Implementation erlaubt es einem maliziösen Applet jede Ressource, die per URL auf lokalen Maschine erreichbar ist, mit Hilfe der Klassen netscape.net.URLConnection und netscape.net.URLInputSteam, auszulesen. Diese Klassen prüfen vor dem Zugriff nicht mittels SecurityManager.checkRead() bzw. SecurityManager.checkConnect() ob der Zugriff gestattet ist. Durch den Aufbau von Verbindungen zu anderen Host ist es so möglich eine Firewall auszutricksen.
(auch diesen Bug benutzt Brown Orifice)

05.10.2000
Microsoft Virtual Machine com.ms.activeX.ActiveXComponent Arbitrary Program Execution Vulnerability
Exploit: ja Solution: ja Typ: Access Validation Error
betrifft: Microsoft Internet Explorer 4.0/5.0/5.01/5.5 mit VM 2000,3100,3200,3300
Quelle: [SecurityFocus bid1754]
Info:
Durch Einfügen eines com.ms.activeX.ActiveXComponent Java Objekts in ein <Applet> Tag wird das Kreieren und/oder das Scripting beliebiger ActiveX Objekte möglich, auch wenn diese Sicherheitsprobleme darstellen. Selbst wenn Active Scripting ausgeschaltet ist, so ist dessen Gebrauch mit Hilfe von Java noch möglich.

18.10.2000
Microsoft Virtual Machine Arbitrary Java Codebase Execution Vulnerability
Exploit: ja Solution: ja Typ: Access Validation Error
betrifft: Microsoft Internet Explorer 4.0/5.0/5.01/5.5 mit VM 2000,3000
Quelle: [SecurityFocus bid1812]
Info:
Ein Angreifer kann in eine HTML Seite mit einem <OBJECT> Tag sein Applet (als JAR) laden lassen und mit CODEBASE ein beliebiges Ziel angeben und so jedes namentlich bekannte File auslesen.

29.11.2000
JRE Disallowed Class Loading Vulnerability
Exploit: nein Solution: nein Typ: Access Validation Error
betrifft: viele Sun JDK ´s für Linux, Solaris und Windows
betrifft nicht: einige Sun JDK´s für Linux und Solaris
Quelle: [SecurityFocus bid 2051]
Info:
Sun gibt leider keine genauen Informationen zu diesem Vulnerability heraus. Unter gewissen Umständen soll es möglich sein, daß nicht vertrauenswürdige Klassen nicht erlaubte Klassen rufen.

5.4.2 Hostile Applets

Neben den im vorherigen Abschnitt erwähnten Exploits, waren es i.d.R. Applets die die Vulnerablities ausnutzten. Es gibt noch eine Reihe von weiteren hostile Applets drei davon seinen hier kurz vorgestellt [mladue 00].

Pickpocket ,
Applikationen, die das login und das Paßwort für einen bequemen User speichern waren schon immer ein beliebtes Angriffsziel. Unglücklicherweise hat Sun´s Java Wallet dieses Feature und es ist leicht für ein nicht vertrauenswürdiges Applet an dies Informationen zu gelangen. Es ist im home Verzeichnis des Users zu finden, welches man über die Property user.home herausfinden kann. Nun liest das Applet das Java Wallet Properties File (jecf.properties) aus. Als Resultat erhält man die login Daten des Users und sein verschlüsseltes Paßwort, was man leicht decrypten kann.

CrashComm405,
ist ein „Einzeiler“ der Netscape´s Communicator 4.05 zu abstürzen bringt. Es ist wohl das kleinste bekannte hostile Applet. Es benutzt die Klasse Softwareupdate aus dem Paket netscape.softwareupdate.

```
import netscape.softwareupdate.*;
...
//einen extra Thread eröffnen…
//und diesen starten
...
public void run (){
        xyz = new SoftwareUpdate (null, null);
        }
//und schon stürzt Netscape ab!
```

DiskHog,
ist ein Applet, das die Festplatte des Users füllt. Im Netscape Communicator 4.05 gibt es ein neues Paket, netscape.secfile, das einem Applet erlaubt, unter einer „secure directory“, virtuell uneingeschränkten Zugriff auf die Festplatte zu nehmen. Dieses secfile ist i.d.R. das user.home Verzeichnis. Das Applet läuft im Hintergrund ab und füllt die Festplatte bis es zu spät ist.

5.4.3 Der Trojaner (Backdoor): Brown Orifice

Am 3. August 2000 veröffentlicht der Autor Dan Brumleve (so nennt er sich jedenfalls) ein Java Applet, das sich zwei Sicherheitslücken in der Virtual Machine von Netscape zu nutze macht [McQueen 00].
Sein Attack (Hostile) Applet macht aus jedem Netscape Browser bis Version 4.74, der das Applet lädt, ein Webserver, aus den jeder Zugriff hat. Dieses maliziöse Applet erlaubt es auf dem Opferrechner beliebige Dateien zu betrachten, verändern und zu löschen.
Im Unterschied zu anderen Attacken mußten erst 1.000 Rechner infiziert werden, bevor Netscape mit einer Warnung reagierte.
Es wurde entdeckt, daß die Sun Java Virtual Machine für Java 1 an einer Sicherheitslücke leidet, die es einem bösartigen User erlaubt mit anderen Rechner, als dem das Applet geladen wurde, eine Netzwerkverbindung aufzubauen.
Die Java 2 Virtual Machine von Sun Microsystems ist nicht verwundbar, da der Fehler behoben wurde.
Es steht weiterhin fest, daß die Hauptklassen, die Netscape benutzt, an einem weiteren Fehler leiden. Dieser Fehler erlaubt es, daß Dateien auf dem Client beliebig gelesen, geschrieben, und verändert werden können.
Nicht alle Hersteller von Java VM´s und Hauptklassen haben diese Verwundbarkeit. Die Microsoft Virtual Machine war nicht mit diesem Fehlern belastet. Es stellt sich die Frage, warum Microsoft nicht von Brown Orifice betroffen ist und welche Implementierungsfehler bei Netcape und Sun gemacht worden sind.
Die Fehler wurden schon im Kapitel über Vulnerabilities erläutert. Es handelt sich zum einen um:
Multiple Vendor Java Virtual Machine Listening Socket Vulnerability [SecurityFocus bid1545]
Netscape Communicator URL Read Vulnerability [SecurityFocus bid1546]

Vor dem ersten Fehler ist Microsoft’s Virtual Machine gut geschützt, da die Hauptklassen nicht den Methoden ServerSocket.open() und Socket.open() vertrauen.
Der Security Manager lost folgende Exception aus:

```
com.ms.security.SecurityExceptionEx[BOServerSocket.]: cannot access 8080
        at com/ms/security/permissions/NetIOPermission.check
        at com/ms/security/PolicyEngine.deepCheck
        at com/ms/security/PolicyEngine.checkPermission
        at com/ms/security/StandardSecurityManager.chk
        at com/ms/security/StandardSecurityManager.checkListen
        at java/net/ServerSocket.
        at java/net/ServerSocket.
        at BOServerSocket.
        at BOHTTPD.init
        at com/ms/applet/AppletPanel.securedCall0
        at com/ms/applet/AppletPanel.securedCall
        at com/ms/applet/AppletPanel.processSentEvent
```

at com/ms/applet/AppletPanel.processSentEvent
at com/ms/applet/AppletPanel.run
at java/lang/Thread.run

Vor dem zweiten Fehler ist Microsoft's Virtual Machine gut geschützt, da Microsoft´s Standard Security Manager streng jede URL Verbindung zur lokalen Maschine verbietet.

5.4.4 Java Viren

5.4.4.1 Strange Brew

Erster plattformunabhängiger Virus. Im August 1998 entdeckt [Nachenberg 98]. Er ist parasitär, d.h. er befällt .class Dateien. Es sind Applets und Applikationen betroffen. Durch Applets besteht keine Gefahr, da die Sicherheitsrestriktionen der Browser (Sandbox) dies verhindern. Die VM sorgt dafür, daß lokale Dateien vor dem Zugriff durch das Applet geschützt sind.
Applikationen hingegen haben Zugriff auf lokale Dateien. So kann der Virus sich nur über Applikationen verbreiten.
Bei Java2 kann dies durch eine entsprechende Policy vermeiden werden. Allerdings stellt gerade diese Policy bei Java2 eine potentielle Gefahr dar, wenn diese unzureichend oder falsch konfiguriert ist.

Neben dem parasitären Verhalten ist der Virus ein „direct action Virus", d.h. sobald er die Kontrolle von der infizierten Applikation bekommt versucht er sofort weitere Dateien zu infizieren, terminiert dann und gibt die Kontrolle zurück an die Applikation. Er verbleibt nicht im Speicher und führt auch keine weiteren Handlungen aus.
Strange Brew erhält allerdings nicht immer die Kontrolle der Applikation. Dies hängt stark von dem Gebrauch und der Programmlogik der Applikation ab.

Erhält er die Kontrolle laufen zwei Phasen ab. Zuerst untersucht er das aktuelle Verzeichnis nach weiteren infizierten „class" Dateien. Wird so eine Datei gefunden, werden Teile der viralen Programmlogik und Daten in den Speicher geladen, die nötig sind um weitere Dateien zu infizieren. Danach beginnt die zweite Phase. Findet der Virus keine infizierte Datei terminiert er und gibt die Kontrolle zurück zur Applikation.

Hat der Virus nun ein infiziertes File gefunden und dessen Teile in den Speicher geladen sucht er nach weiteren Dateien um diese zu infizieren. Wenn die Größe einer Datei durch 101 teilbar ist, wird angenommen, daß die Datei schon infiziert ist. Die liegt daran, daß Strange Brew alle infizierten Datei so verändert, das ihre Größe durch 101 teilbar ist. Dies führt dazu das der Virus einige nicht infizierte Dateien, deren ursprüngliche Größe ebenfalls durch 101 teilbar ist, übersieht. Wird nun eine nicht infizierte class Datei entdeckt überprüft der Virus anhand von einigen internen Regeln ob die Datei infiziert werden kann. Ist die Datei nicht passend wird ihre Größe, durch einfügen einiger Bytes, so verändert, das sie durch 101 teilbar ist. Dies erleichtert später das finden von infizierbaren Dateien.

Ist nun eine passende Datei gefunden fügt er sich zu der Hostdatei (Hostdatei wird die zu infizierende Datei genannt) dazu. Der Virus erstellt in der Hostdatei eine neue Methode (Strange_Brew_Virus(), daher der Name) mit seiner Programmlogik vor allen anderen Methoden der Hostdatei. Danach wird die original Programmlogik so verändert, daß die virale Methode die Kontrolle bekommt. Während des patchens wird die Fehlerbehandlung der Hostdatei verändert, was teilweise zu fehlerhaften Programmabläufen führt. Die meisten Applikationen arbeiten aber weiterhin korrekt. Zum Schluß werden noch einige Tabellen und Felder in der Hostdatei verändert.
Der Virus infiziert alle passenden .class Dateien im aktuellen Verzeichnis bevor er die Kontrolle wieder an die Applikation gibt. Die Hostdateien werden dabei im grob 3.890 Bytes vergrößert. Der Timestamp von Verzeichnis und Datei wird abschließend manipuliert.

Der Einfügeprozeß ist nicht sehr gut designed und fehlerhaft. Dies führt dazu, daß Dateien falsch infiziert werden oder das der Virus abstürzt. Wenn der Virus während der Infizierung abstürzt, wird die Applikation terminiert und weitere Infizierungversuche scheitern.

Der Strange Brew Virus hat ursprünglich keine Payload und führt neben der Infizierung und möglichen Schaden durch fehlerhafte Infizierung, zu keinen weiteren Schäden.
Der Virus ist nicht „in the wild“, d.h. er ist nicht verbreitet und Opfer sind nicht bekannt. Es wurde nicht beabsichtigt den Enduser oder Unternehmen zu bedrohen. Doch Java und WWW Entwickler haben (durch häufigen Datenaustausch) ein gewisses Risiko.
User, die von dem Virus betroffen sind, werden merken, daß ihre Applikationen länger zum laden brauchen oder nicht mehr ordnungsgemäß funktionieren.

Wird ein infiziertes Java Applet über das Netz geladen und in einem WWW Browser zur Ausführung gebracht, so erscheinen folgende Meldungen:

Netscape 4.05:
Applet <Applet name> can't start: class got a security violation: method verification error

IE4.0:
error: com.ms.lang. VerifyErrorEx: WVLayout.Strange_Brew_Virus: invalid constant value

5.4.4.2 Bean Hive

Im August 1999 wurde dieser zweite Java Virus entdeckt [Nachenberg 99]. Dieser Virus wurde entwickelt, um Applets und Applikationen zu infizieren. Ziel war es nicht den Enduser oder Unternehmungen anzugreifen. Daher enthält er keine destruktive Payload. Er besitzt eine Reihe neuer interessanter Technologien, um sich zu verbreiten.

Lädt ein User ein Applet aus dem Netz, läuft dieses in der Sandbox ab und hat grundsätzlich keinen Zugriff auf lokale Dateien. Die Sicherheitsrestriktionen des Browsers (wenn richtig konfiguriert) verhindern dies. Allerdings kann das Applet die VM nach erweiterten Rechten fragen, welche diese Anfrage dann über den Browser an den User weiter reicht. Dadurch werden eine Menge neuer Anwendungen für Applets möglich. Die so gelockerten Restriktionen führen, genauso wie bei Applikationen, dazu daß sich der Virus verbreiten kann. Der erste Java Virus Strange Brew versuchte Zugriff auf Dateien zu erlangen ohne vorher ordnungsgemäß die VM nach Zugriff zu fragen und wurde daher sofort terminiert.

Der Bean Hive Virus ist auch daher beachtenswert, da er den wenigen Viren gehört, bei denen die Programmlogik über mehrere Dateien verteilt ist. Die meisten andren Viren bestehen nur aus einer Datei. Bei der Replikation kopieren sie ihre Replikationslogik von dem „alten“ in das „neue“ Hostfile. Der Autor plazierte sieben von neun Dateien in einem Jar-Archiv (für Netscape Navigator) bzw. Cab-Archiv (für MS Internet Explorer). Dieses heißt dann BenHive.jar bzw. BeanHive.cab.

Der Virus besteh aus folgenden Komponenten:

Im Archiv sind (Worker)

BeanHiveFrame.class	: Hauptklasse (Dropper)
BeanHiveApplet.class	: Hauptklasse (Dropper)
e89a763c.class	: Dateiformatsparsing
a98b34f2.class	: Dateizugriffsfunktionen
be93a29f.class	: bereitet die Datei für Infektion vor (teil1)
c8f67b45.class	: bereitet die Datei für Infektion vor (teil2)
dc98e742.class	: fügt den Virusstarter in die Hostdatei ein

aus von dem Attackerhost geladen (Queen)

BeanHive.class	: sucht nach Dateien im Verzeichnisbaum

und dem Hostfile (Opfer).

Der User kann mit dem Virus infiziert werden, wenn er auf eine Website des eines Virenautors surft. Dann wird automatisch das gesamte Archiv, welches die sieben Virus Module enthält, auf den Rechner des Users geladen. Die Hauptklasse des Virus (BeanHiveFrame.class) wird dann sofort in der Virtual Machine ausgeführt. Zuerst fragt diese Klasse nach Zugriff auf das lokale System (wie oben beschrieben). Wenn der User sich entscheidet dies nicht zu erlauben, kann der Virus nicht Arbeiten und terminiert sofort. Entscheidet sich der User dem Virus die Rechte zu erteilen, fährt dieser fort. Es ist wichtig zu bemerken, daß der User damit jedem Code des Autors diese Rechte erteilt. Dies liegt an der Art wie Java signierten Code verwaltet. Die Rechte werden nicht an eine Klasse sondern an den Autoren (bzw. dessen Signatur) vergeben. (Bei Java2 spielt der Herkunftsort, die Codebase, noch eine Rolle.)

Nachdem der Virus die nötigen Rechte erhalten hat, fragt er in einem Standarddialog welches File er infizieren soll. Dies macht deutlich, daß der Virus nur zu Demonstrationszwecken entwickelt wurde. Ein bösartiger Virus würde dies natürlich nicht tun. Diese Hauptklasse ist ein „Dropper“. Sie dient nur dazu den Virus auf den Host Rechner zu laden und wird später nicht mehr benötigt.

Der Dropper benutzt nun die vier anderen Dateien aus dem Archiv, um die gewünschte Datei zu infizieren. Zu erst wird die Klasse e89a763c.class, welche die Infektionsroutine beinhaltet, gerufen, um zu prüfen, ob das Hostfile infizierbar ist. Ist dies der Fall ruft die Klasse die drei anderen Klassen (c8f67b45.class, dc98e742.class und be93a29f.class), um das Zielprogramm zu infizieren. a98b34f2.class stellt eine Reihe von Dateizugriffs-funktionen zur Verfügung. Alle drei Dateien arbeiten zusammen, um die achte Komponente des Virus in die Zieldatei einzubringen. Nun ist das Zielprogramm mit einem kleinen Teil der viralen Logik erweitert worden. Abschließend wird das Zielprogramm noch so modifiziert, daß der virale Teil vor dem ursprünglichen Code ausgeführt wird.

Startet der User das infizierte Programm erhält der Virus sofort die Kontrolle. Seine Logik ist sehr einfach:
Sie versucht nur die neunte und letzte Komponente (BeanHive.class) zu lokalisieren und zu starten. Der Autor des Virus nennt diese neunte Komponente „Queen“. Diese Datei ist nicht in dem Archiv (BeanHive.jar bzw. BeanHive.cab) enthalten und ist auch nicht auf dem Rechner des Users, vor der ersten Ausführung des Virus, zu finden. Kann diese Datei nicht gefunden werden , wird diese über das the World Wide Web geladen.

Wenn das infizierte Java Programm die Datei im Netz findet, wird diese sofort herunter geladen. Das BeanHive.class Modul enthält die Logik, um das aktuelle Verzeichnis des Users nach weiteren infizierbaren Java Dateien zu durchsuchen. Es wird versuchen drei weitere Dateien zu infizieren, bevor es die Kontrolle wieder an das infizierte Hauptprogramm gibt. Das BeanHive.class Modul benutzt bei der Infizierung weiterer Dateien die schon bei der ersten Infizierung benutzten Helperklassen (c8f67b45.class, dc98e742.class, be93a29f.class und a98b34f2.class), um das Hostfile zu infizieren. Diese installieren wieder die achte Komponente im Zielprogramm.

Sollen diese weiteren infizierten Programme sich weiter ausbreiten, benötigen sie die BeanHive.class (Queen) und alle Dateien aus dem Archiv (Worker) auf dem Zielrechner. Sendet sich eine Infizierte Datei zu einem anderen Rechner oder wird nur eine der acht anderen Komponenten gelöscht kann sich der Virus nicht weiter selbst verbreiten.

6 Java

6.1 Java Grundlagen

Java wurde von James Gosling entwickelt [Sun Java History] und bekam zunächst den Namen “Oak”, benannt nach einem Baum vor seinem Fenster. Es handelt sich um eine objektorientierte Sprache, die aus einer Untermenge von C++ und Smalltalk entstanden ist. Sie enthält Elemente aus Smalltalk, wie Bytecode und Garbage Collection. Andere Elemente, wie z.B. Pointerarithmetik aus C++ wurden weggelassen um ein möglichst stabiles System zu schaffen (40% - 60% aller Fehler bei der Programmierung entsteht durch falsches Speichermanagement). Weitere Features wurden dazu genommen.
James Gosling gehörte zum sogenannten “Green Team” welches die “Star Seven” (*7) Plattform (benannt nach einer Funktion der Telefonanlage im Entwicklungscenter) im Sommer 1992 als erstes Demo präsentierte.
Das geheime Greenteam, initiiert von u.a. Patrick Naughton, Mike Sheridan und James Gosling, bestand aus 13 Personen welche von Sun angeheuert wurden.
Es begann 1991 die Technologie zu entwickeln. Ziel war es für hybride Systeme im Consumer-Electronic-Bereich eine plattformunabhängige moderne Programmiersprache zu entwickeln, mit der sich elektrische Geräte (z.B. Waschmaschinen) ebenso programmieren lassen wie Computer. Nachdem die Sprache konzipiert war und die Projekte in diesem Bereich nicht den gewünschten Erfolg brachten, kam man auf die Idee, Java für Internet-Anwendungen einzusetzen. Im 1993 folgenden Boom des WWW erkannte man, daß Java, durch die Portabilität, die eine Kernforderung bei der Konzeption der Sprache war, ideal für Internet-Anwendungen geeignet ist. Applets , kleine Java-Programme, sollten durch ein eigens dafür definiertes Tag in HTML-Seiten eingebunden werden und dadurch die bisher statischen HTML-Seiten dynamischer gestalten. Sun entwickelte dafür einen Web-Browser namens HotJava. Es war erste Browser, der Applets unterstützte.

6.2 Neuerungen in Java

Vom JDK 1.0 bis zum JDK 1.2 gab es zum Thema Sicherheit einige Änderungen. Das Sicherheitsmodell vom JDK 1.0 war noch sehr einfach gehalten, d.h. eine Java-Anwendung hatte entweder alle Rechte oder keine. Zu welcher Kategorie eine Anwendung gehörte, hing hauptsächlich davon ab, ob sie als Applet oder Applikation lief. Bei Applets mußte man zusätzlich zwischen Applets auf der lokalen Platte und Applets im Netzwerk unterscheiden. Es war z. B. nicht möglich, Applets im Netzwerk erweiterte Rechte zuzuteilen, was z.B. in einem Intranet Sinn machen würde. Deshalb erweiterte man das JDK 1.1 um signierte Applets, welche dann dieselben Rechte wie eine Standalone-Anwendung haben.

Das Modell des JDK 1.1 erlaubt es zwar, signierten Applets weitere Rechte zu erteilen, die Zuteilung erfolgt aber dennoch binär. Entweder hat das Applet alle Rechte oder überhaupt keine.

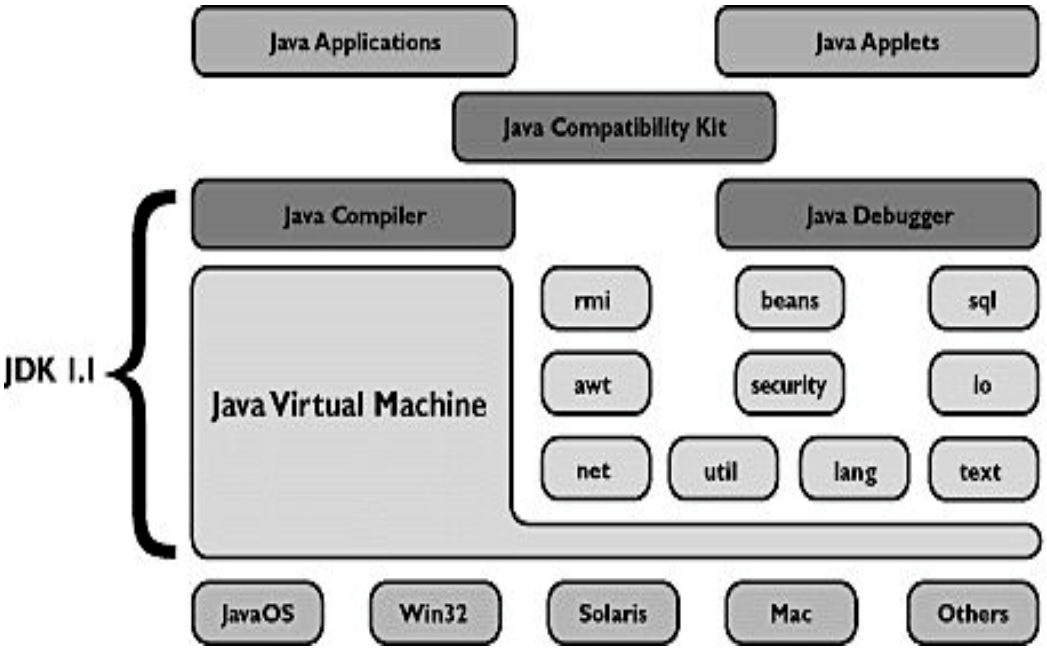
Dieses Problem behebt erst das Modell des JDK 1.2. Dort ist es möglich, Applets entweder nach Herkunftsort (Codebase) oder nach Urheber (Signierer) differenzierter Rechte zuzuteilen.

In der Praxis spielt das Sicherheitsmodell des JDK 1.2 bei Applets z.Zt. nur eine Rolle, wenn das Java-Plug-In eingesetzt wird. Da das Java-Plug-In die Laufzeitumgebung des Original-JDK verwendet, wird dort auch das aktuelle Sicherheitsmodell benutzt.
Im Moment besitzen die gängigen Browser Internet Explorer 5.0 und Netscape Communicator (bis 4.5) eine eingebaute Java-Laufzeitumgebung und ein eigenes Sicherheitsmodell.
Beim Internet Explorer beschränkt sich die Inkompatibilität auf unterschiedliche Granularität in der Vergabe und der Art, wie Rechte vergeben werden.
Beim Netscape Communicator müssen sogar Änderungen am Quellcode vorgenommen werden, um dessen Sicherheitsmodell zu benutzen.

6.2.1 Neuerungen in Java 1.1

6.2.1.1 Allgemeines

Die Komponenten von Java 1.1 sind in folgernder Grafik zu sehen:



Verwerfen von Methoden und Klassen
In Java 1.1 und 1.2 wurden einige Methoden und Elemente wieder verworfen und durch neue ersetzt. Dies geschah u.a. aus mangelnder Portabilität und Anpassungsfähigkeit hinsichtlich der Internationalisierung oder einfach nur wegen ungünstigen Bezeichnern. Die alten Methoden werden aber weiterhin unterstützt, um Abwärtskompatibilität zu wahren Eine als verworfene Methode wird dem javadoc-Tag @deprecated gekennzeichnet.

Geschachtelte Klassen
Es möglich Unterklassen in Klassen und Methoden zu bilden.

Neue Verwendungen von *final*
final kann jetzt auch für Methodenparameter, lokale Variablen in Methoden und den Parameter von catch-Anweisungen benutzt werden. Es gibt jetzt auch leere Finals, d. h. Konstanten, denen erst nach der Deklaration höchstens einmal ein Wert zugewiesen werden kann.

Anonyme Arrays
Array-Literale können nun verwendet werden ohne eine Variable dafür zu vereinbaren, genauso wie bei den Literalen der einfachen Datentypen.

Klassenliterale
Das Class-Objekt einer Klasse kann jetzt auch deklarativ ermittelt werden:
Class c = java.lang.Integer.class;

Steuerung des Finalisierungszeitpunkts
Klasse System enthält neue Methoden, um die Ausführung von Finalisierern zu starten.

Neue Wrapper-Klassen
Es jetzt zu allen einfachen Datentypen eine entsprechende Wrapper-Klasse, da die Klassen Short, Byte und Void ergänzt wurden.

Object Serialization
Die Object Serialization bietet die Möglichkeit, Java-Objekte persistent zu machen. Die persistenten Objekte können so in Dateien gespeichert oder über Netzwerkverbindungen übertragen werden. Letzteres bildet die Grundlage für RMI.

Neue AWT-Features
Das neue Event-Modell ermöglicht ein vereinfachtes Event-Handling für komplexe Oberflächen, nun muß sich ein Objekt bei einer Komponente für die Benachrichtigung über das Auftreten eines bestimmten Ereignisses registrieren lassen. Des weiteren gibt es nun Klassen, die das Drucken aus Java heraus erlauben, Tastatur-Shortcuts für Menüfunktionen, Unterstützung für Zwischenablagen und die Windows-Implementierung des AWT wurde komplett neu geschrieben, um die Einheitlichkeit zwischen den Plattformen zu erhöhen.

Neue AWT-Komponenten
PopupMenu und ScrollPane wurden eingeführt.
Internationalization
a. Neue Reader- und Writer-Klassen im I/O-Paket stellen Streams zur vollen Unicode-Unterstützung zur Verfügung.
b. Verbesserte Unterstützung verschiedener Zeit- und Datumsformate
c. Unterstützung für mehrsprachige Menüs, Meldungen und dergleichen

JAR-Archive und signierte Applets
Mit JAR-Archiven können Bytecode-Dateien und Ressourcen (bei Applets) in einer Archivdatei zusammengefaßt werden. Sie bilden die Basis für die elektronische Signatur von Java-Applets.

Neue Netzwerk-Features
a. Unterstützung von IP-Multicasting
b. Unterstützung einiger der Socket-Optionen von BSD-UNIX, nun kann ein Timeout für Socket-Operationen spezifiziert werden.
c. Die Java-Sockets unterstützen SOCKS-Proxies.

Ressourcen
Auf Ressourcen kann jetzt in einheitlicher Weise zugegriffen werden, die nun mit Zeichenketten identifiziert werden, die wie Property-Namen strukturiert sind.

6.2.1.2 Neue API‘s und Pakete

RMI (Remote Method Invocation)
Netzwerkkommunikation wird durch RMI Methodenaufrufe abstrahiert. Weiteres zu RMI in Kapitel 7.

JDBC (Java Database Connectivity)
JDBC definiert eine Schnittstelle zum Datenbankzugriff direkt aus Java heraus.

Reflection API
Das Reflection API dient der Analyse von Klassen und Objekte zur Laufzeit

Java Beans API
Das Java Beans API definiert einige Erweiterungen zum neuen Event-Modell, diese grafischen Komponenten (Java Beans) kommunizieren über Events.

Security API
Das Security API definiert eine Schnittstelle für verschiedene kryptographische Verfahren wie digitale Signaturen oder Prüfsummenberechnung.

Paket java.text
Dieses Paket ist Bestandteil der Internationalization.

Paket java.util.zip
Die Klassen des Pakets util.zip können ZIP-Archive auslesen und erzeugen.

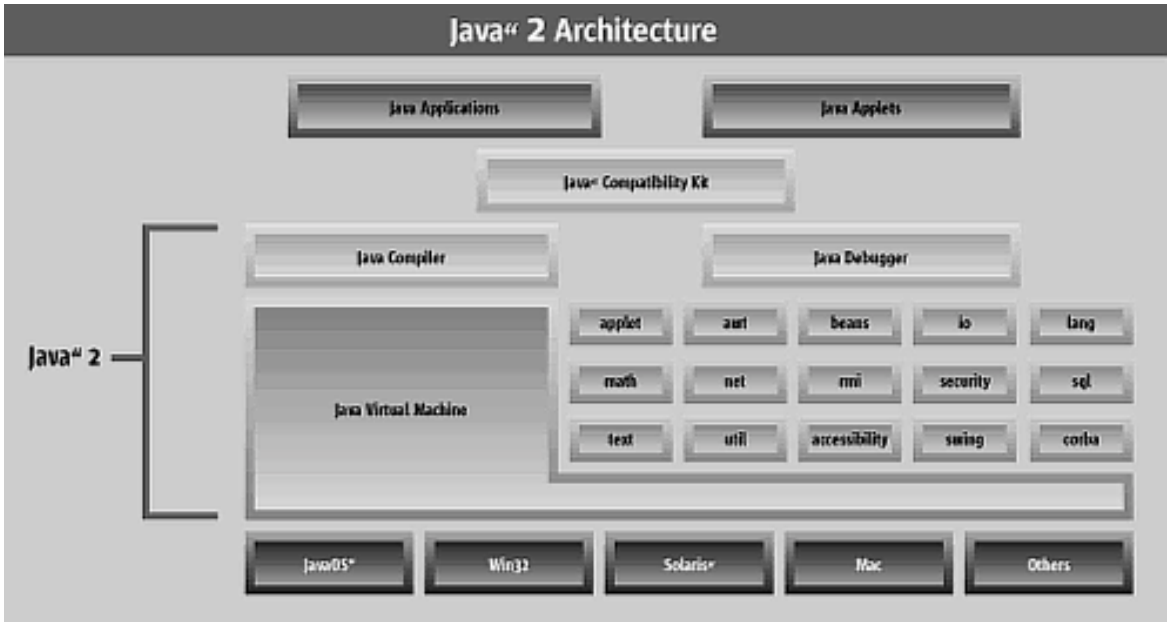
Paket java.lang.math
Das Paket lang.math enthält Klassen zur Darstellung und zum Rechnen mit großen und beliebig genauen Dezimal- und Ganzzahlen.

JNI (Java Native Interface)
JNI ist eine plattformunabhängige Schnittstelle für das Einbindung von nativem Code.

6.2.2 Neuerungen in Java 1.2

6.2.2.1 Allgemeines

Die Komponenten von Java 1.1 sind in folgernder Grafik zu sehen:



Zum Zeitpunkt der Freigabe des Final Release des JDK 1.2 etablierte Sun den Begriff der "Java-2-Plattform", also ein neuer Name für die Virtual Machine und die stark erweiterten Klassenbibliotheken auf dem Stand der Version 1.2.

Java Foundation Classes (JFC)
Die JFC-Klassen stellen einige komplexe Oberflächenkomponenten zur Verfügung, deren Look-and-Feel austauschbar ist.

Flexibilisierung des Security-Mechanismus
Eine individuelle Zuteilung der Zugriffsrechte auf Ressourcen an Programme gestatteten verschiedene Permission-Klassen, dadurch wird feinere Granularität erreicht.

Schwache Referenzen
Schwache Referenzen auf Objekte können mit den Klassen im neuen Paket java.lang.ref gehalten werden, d.h. daß das referenzierte Objekt trotzdem vom Garbage Collector entfernt werden kann.

Erweiterte Gleitpunkt-Darstellungen
Plattformen, die die erweiterte Gleitpunkt-Darstellungen unterstützen, können nun mit höherer Genauigkeit und Geschwindigkeit Gleitpunkt-Operationen durchführen.

Multithreading
Die Methoden stop(), suspend() und resume() der Klasse Thread wurden verworfen, da sie Inkonsistenzen hervorrufen bzw. sich als Deadlock-trächtig erwiesen haben.
Gegenüber anderen Threads völlig gekapselte, threadlokale Variablen können mit den neuen Klassen ThreadLocal und InheritableThreadLocal können vereinbart werden. Dies

erhöht die Threadsicherheit. Threads werden jetzt unter Solaris durch die nativen Threads des Betriebssystems realisiert.

Vergleichen und Sortieren von Objekten
Es ist nun möglich eine Ordnung auf einer Klasse, durch die Interfaces java.lang.Comparable und java.util.Comparator, zu definieren. Die neue Klasse java.util.Arrays kann dann die Objekte dieser Klasse sortieren.

Paketversionierung, Reflection: Package
Das verteilen von Software und Updates wird durch die neue Klasse java.lang.Package erheblich vereinfacht. Diese ermöglicht es Pakete zu versionieren und zur Laufzeit zu bestimmen, ob ein Paket kompatibel zu einer benötigten Version ist.

Sound
Eine höhere Wiedergabequalität und Unterstützung der Formate wav und aiff bietet der neue Sound-Engine, der bisher nur au-Dateien verarbeiten konnte.

Performance-Steigerungen
Die Virtual Machine konnte verbessert werden, was die Ablaufgeschwindigkeit merklich erhöht. Das JDK ist in den Windows- und Solaris-Versionen mit einem Just-in-Time-Compiler ausgestattet, der den Bytecode zur Laufzeit in nativen Code übersetzt.

Verbesserungen bei RMI
Die Implementierung eigener Socket-Factories wurden wesentlich vereinfacht. Weiter sind nun persistente Referenzen auf ein entferntes Objekt möglich, das vom Server, durch neue Aktivierungsklassen, erst bei einer Anfrage aktiviert wird.

Verbesserungen bei der Object Serialization
Die persistenten Daten eines Objekts von seinen Datenelementen unabhängig zu machen ermöglichen neue Interfaces im I/O-Paket. Abwärtskompatibilität zum bisherigen Format der Object Serialization bleibt gewahrt.

6.2.2.2 Neue API‘s und Pakete

Java 2D-API
Das Java 2D-API enthält Klassen für komplexe, an Postscript angelehnte Zeichenoperationen sowie umfangreiche Funktionalität zur Bildverarbeitung und Filterung. Die Klassen des 2D-API ergänzen die Pakete java.awt und java.awt.image. Letzteres enthält jetzt auch Klassen für affine Abbildungen.

Drag and Drop
Das neue Paket awt.dnd bietet Unterstützung für Drag-and-Drop-Funktionalität.

Accessibility API
Dieses API bietet die Möglichkeit, AWT-Komponenten mit Spezial-Hardware wie Braille-Terminals anzusteuern.

Collections API
Das Collections API stellt einige Container-Klassen sowie Algorithmen zur Verfügung, die auf die Inhalte von Containern angewendet werden können.

Java IDL
Die Java IDL stellt Klassen für die Verwendung von CORBA in Java-Anwendungen zur Verfügung. Ferner gehört das Tool idltojava zu Java IDL, mit der aus einer IDL-Spezifikation die zugehörigen Java-Dateien generiert werden können.

Paket java.util.jar
In Analogie zum Paket util.zip können jetzt auch JAR-Archive von Programmen ausgelesen und erzeugt werden.

7 Java 2 Security

7.1 Language Level Security

7.1.1 Unberechtigter Speicherzugriff

Die Sprache Java entstand aus einer Untermenge der Sprache C++ mit einem leicht verändertem Syntax. Um unberechtigten Speicherzugriff zu verhindern wurde die Pointerarithmetik entfernt. Pointer gibt es weiterhin in Form von Referenzen, so daß Datenstrukturen wie z.B. verkettete Listen und Bäume generiert werden können.

Die Spezifikation von Java definiert genau den Umgang mit nicht initialisierten Variablen. Jeder heap-basierter Speicher wird automatisch initialisiert, stack-basierter Speicher nicht. Lokale Variablen müssen initialisiert werden. Ansonsten gibt es schon zur Compilezeit einen Fehler. So können alle Klassen- und Instanzvariablen niemals nicht definierte Werte annehmen.

7.1.2 Garbage Collection

Ein weiteres Feature von Java das zur Sicherheit beiträgt ist Garbage-Collection. Zur Laufzeit wird nicht mehr referenzierter Speicher freigegeben. So muß der Entwickler nicht schon zur Entwicklungszeit entscheiden, wann es sicher ist, Speicher freizugeben und die durch falsche Freigabe entstehenden Fehler werden vermieden.

7.1.3 Weitere Language-Level Sicherheitskonzepte

Java ist stark Typisiert, d.h. schon zur Compilezeit werden die Typen geprüft und illegale Casts führen zur Fehlermeldung. Auch hierdurch wird der Zugriff auf Speicherbereiche verhindert die nicht zum original Objekt gehören.
Zugriffsmodifier (public, protected, *friendly*, and private) regeln den Zugriff bzw. die Sichtbarkeit von Variablen, Methoden und inneren Klassen. [*friendly* ist kein Schlüsselwort. Wird kein Schlüsselwort angegeben wird *friendly* angenommen.]
Der Modifier *final* verhindert bei Klassendefinition ein weiteres Vererben. Auf Methoden angewendet wird ein Überschreiben verhindert. Bei Variablen werden diese zu Konstanten. Es existiert auch eine Variante von final, das “blank final”. Hier kann nur einmal, im Konstruktor, ein Wert zugewiesen werden.

7.1.4 Sicherheit durch Offenheit

Im Gegensatz zu vielen anderen Sprachen wird das Sicherheitsmodell, Konzepte und Implementierungsdetails, wie der Source, von SUN offengelegt. Es liegt also kein Black-Box verhalten vor und von “Security through obscurity” kann nicht die Rede sein.

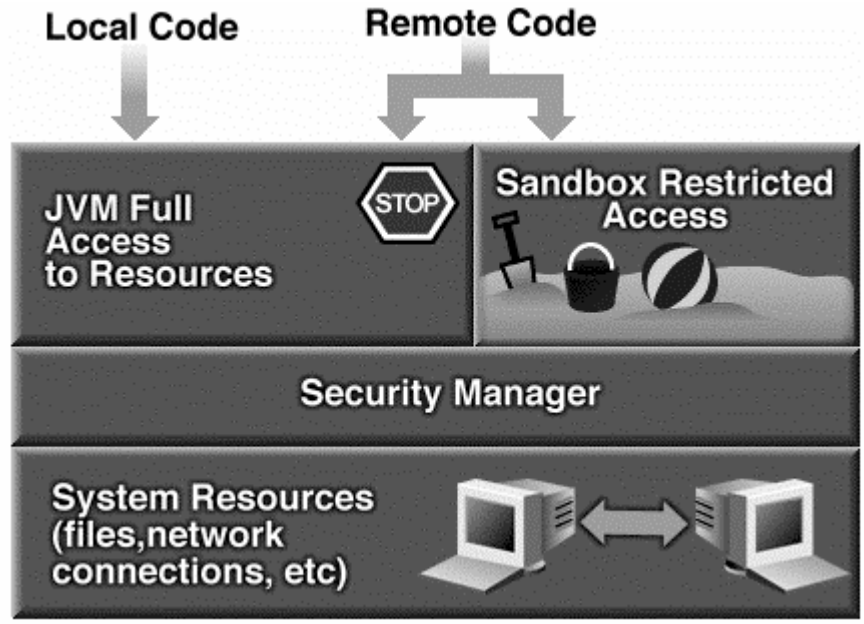
7.1.5 Auditing

Ein wichtiges Merkmal von sichern Systemen ist das Auditing. Leider ist es noch nicht in Java implementiert.

7.2 Java Sandbox Security

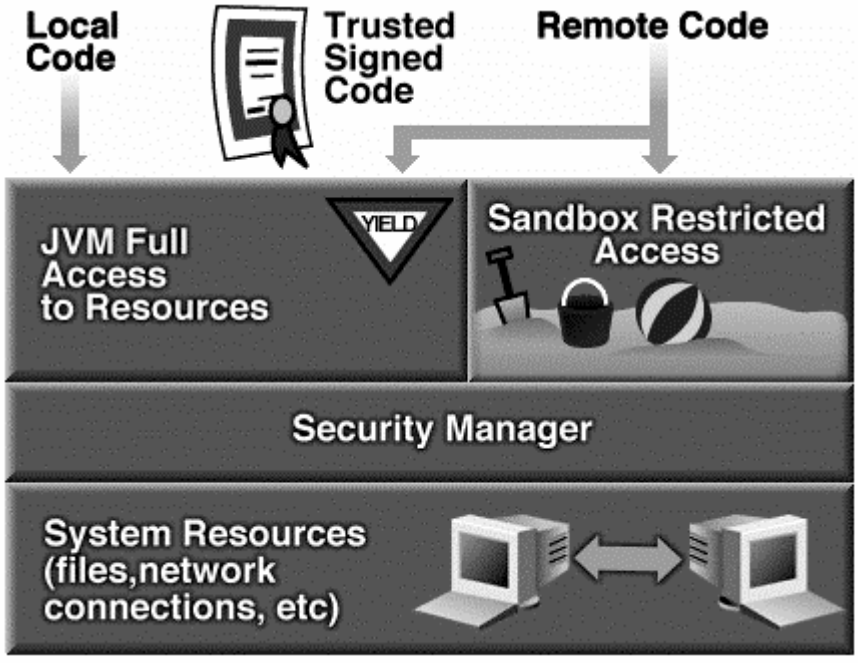
Das original Security Model der Java Plattform war das “Sandbox” Modell, es sollte nicht vertrauenswürdigen Code (der z.B. aus dem Internet heruntergeladen) eine stark restriktiven Umgebung bieten. Das Sandboxmodel (siehe Abbildung) gibt lokalen, vertrauenswürdigem Code vollen Zugriff auf die Systemressourcen und heruntergeladenen Code (z.B. Applet) stellt es nur die limitierte Sandbox - Umgebung zur Verfügung.
Der Security Manager regelt diesen Zugriff.

JDK 1.0 Security Model:



JDK 1.1führte das Prinzip der signierten Applets ein (siehe Abbildung).Ein digital signiertes Applet wird wie lokaler Code behandelt, mit vollem Zugriff auf die Ressourcen, wenn der Public-Key zum Verifizieren der Signatur vertrauenswürdig ist. Applets ohne Signatur laufen weiter in der Sandbox ab. Signierte Applets werden mit ihren Signaturen in signierten JAR-Archiven ausgeliefert.

JDK 1.1 Security Model:



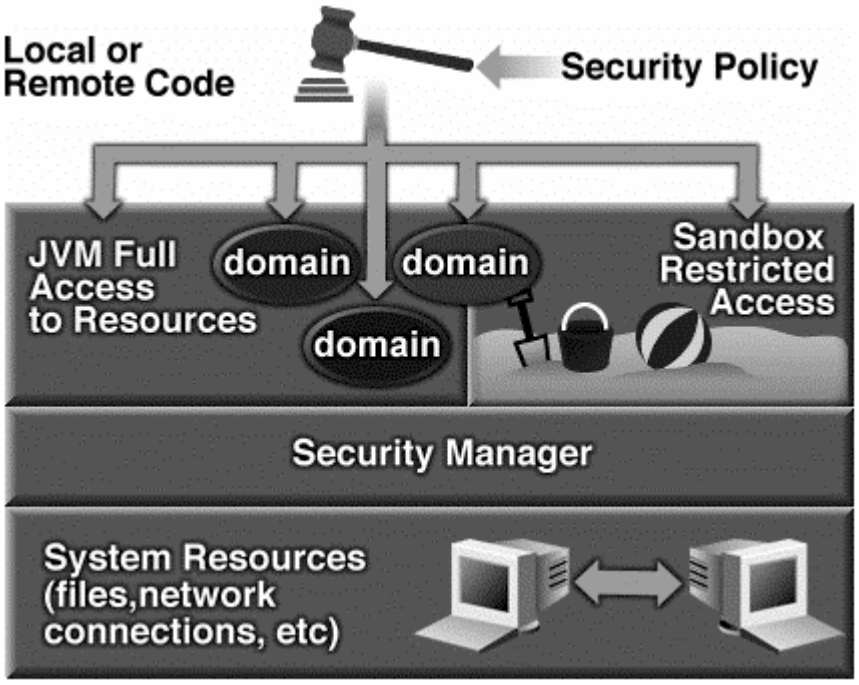
JDK 1.2 bietet gegenüber JDK 1.1 eine Menge neuer Möglichkeiten. Jeder Code, egal ob lokal oder remote geladen, kann Gegenstand der Security Policy sein

Die Security Policy definiert eine Reihe von Rechten (Permissions) für verfügbaren Code von unterschiedlichen Unterzeichnern und/oder Herkunftsorten. Sie kann vom User oder Administrator konfiguriert werden. Jede Permission spezifiziert eine bestimmten Zugriff auf eine bestimmte Ressource, wie Lese- oder Schreibrecht auf ein File oder Directory oder Netzwerkrechte für Hosts, wie Ports etc.

Das Laufzeitsystem organisiert den Code in individuelle Domänen, welche eine Sammlung von Klassen, deren Instanzen die gleichen Rechte besitzen, darstellen. Eine Domäne kann wie die Sandbox konfiguriert werden, so daß Applet weiter in einer restriktiven Umgebung ablaufen, wenn der User oder Administrator dies wünscht. Applikationen laufen wie bisher ohne Restriktionen. Sie können aber auch Subjekt einer Security Policy werden.

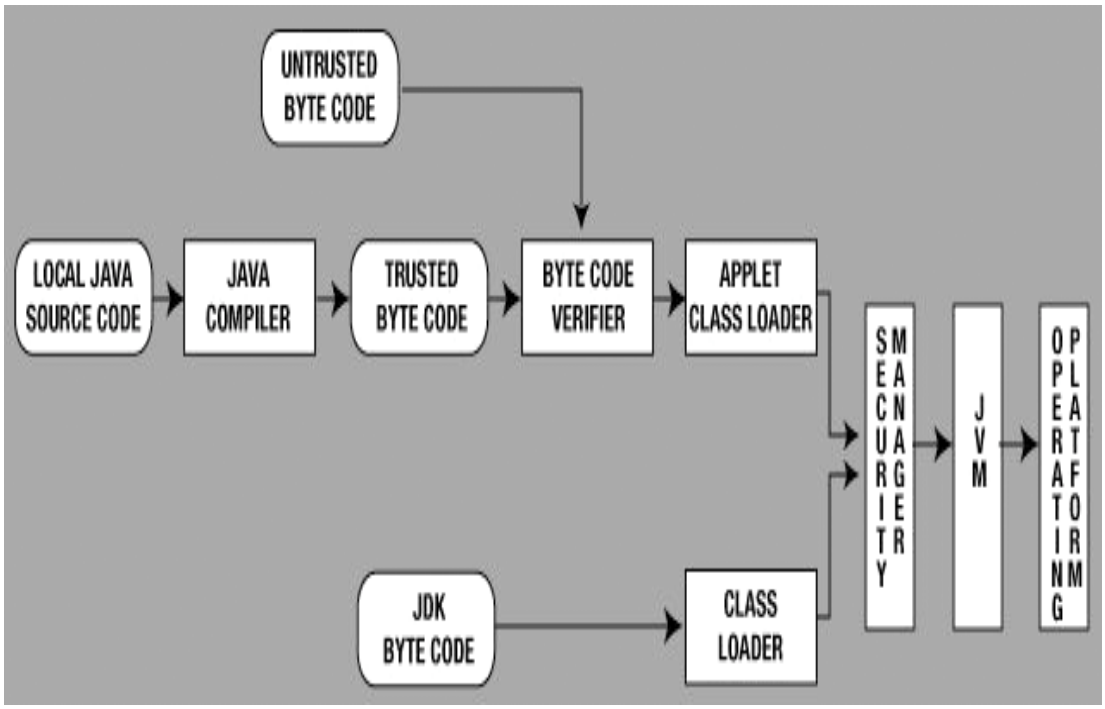
Die neue Sicherheitsarchitektur von JDK 1.2 zeigt folgende Abbildung.
Der Pfeil auf der linken Seite (s.u.) zeigt auf die Domäne mit vollem Zugriff auf die Ressourcen. Der Pfeil ganz rechts zeigt auf den entgegengesetzte Extrem, einer Domäne mit den Restriktionen einer Sandbox. Die Domänen dazwischen haben entsprechende mehr oder weniger eingeschränkte Rechte.

JDK 1.2 Security Model:



7.3 Java Virtual Machine Security

Die folgende Grafik soll das Zusammenwirken der verschiedenen Sicherheitskomponenten verdeutlichen:



7.3.1 Bytecode Verifier

Die Maschinensprache der JVM (Java Virtual Machine) ist der Bytecode, wie er in der JVM Spezifikation spezifiziert wurde. Die Spezifikation definiert wie Klassenteile, wie Variablen, Methoden und Modifier in Java Class Files gespeichert werden. Ein Java Bytecode generierender Compiler konvertiert aus einem Eingabefile einzelne Teile in Bytecode Instruktionen und speichert diese in einer Datei mit der Endung ".class" ab. Die Eingabe muß nicht unbedingt Java sein. Es existieren einige Crosscompiler die aus einem Sourcefile wie Ada, Cobol oder Delphi Java Bytecode generiert.
Man könnte, wenn es nur Language-Level Security in Java geben würde einen Compiler schreiben der die Restriktionen umgeht, also "Malicios Code" erzeugt, oder man schreibt gleich in Bytecode (wie etwa bei der Assemblerprogrammierung)

Wenn man im JRE eine Klasse zur startet, ist der normale Wege, daß das Class File geladen wird, die main Methode ausgeführt wird und dann ergänzende Klassen geladen werden.
Wird eine Klasse geladen, überprüft zuerst die JVM den Bytecode.

Diese Verifizierung besteht aus vier Schritten:

Die Magicnumber im Class File wird überprüft. Das sind die ersten vier Bytes im File: 0xCAFEBABE.

Es wird alles überprüft, was ohne in den Code zu schauen getestet werden kann, z.B. wenn das Objekt nicht vom Typ *Object* ist, wird die Superklasse gesucht.

Es wird der Bytecode verifiziert um sicher zu stellen, daß alle Opcodes gültig sind. Weiter werden die Typen und Opcodes der Argumente der Methoden überprüft.

Die letzte Prüfung findet erst beim Aufruf der Methode statt. Es kann also passieren, daß, eine Methode nie aufgerufen und damit auch nie überprüft wird. Geprüft wird hier die Existenz der Member und deren Zugriffsrechte (public, protected, private, friendly).

JRE 1.1 überprüft allerdings nur Klassen, die über das Netz geladen wurden. Alles was lokal installiert wurde (im CLASSPATH) wird nicht überprüft. Dies ist die Standardeinstellung.
Man kann allerdings die Prüfung mit Optionen der *java* und *jre* Kommandos einstellen:

-verifyremote :	(default) nur Klassen, die übers Netz geladen wurden, werden geprüft
-verify	alles wird geprüft
-noverify	nichts wird geprüft

Im JDK 1.2 sind die default Einstellungen gleich. Es werden nur übers Netz geladene Klassen verifiziert. Die Systemklassen (lokale Klassen) sind etwas anders spezifiziert als die CLASSPATH Umgebungsvariable und die Kommandozeilenoptionen sind etwas anders.

Die Systemklassen werden entweder mit der System Eigenschaft *sun.boot.class.path* oder der Kommandozeilenoption:

–Xbootclasspath:[Verzeichnis oder jarFile]

Der Wert für *sun.boot.class.path* wird von JRE automatisch gesetzt.

Die Optionen für das *java* Kommando sind folgende:

-Xverify:remote	(default) nur Klassen, die übers Netz geladen wurden, werden geprüft
-Xverify:all	alles wird geprüft, ich empfehle dringend diese Option zu setzen!
-Xverify:none	nichts wird geprüft

Tritt ein Verifizierungsfehler auf wird von JRE , anstatt eines Reports oder verification errors, ein Fehler wie: “Can´t find class MyClass” ausgegeben.

Sind alle Überprüfungen erfolgreich beendet kann der Verifier die Opcodes durch sogenannte “quick instructions” ersetzen um die Ausführungsgeschwindigkeit zu erhöhen und um sicher zu stellen, daß es keine erneute Überprüfung gibt. Da jede weitere Überprüfung wieder mit Performanceverlust verbunden ist.

7.3.2 Class Loader

Die Verantwortlichkeit des Ladens und Verifizierung der Klassen liegt beim Class Loader. Dieser findet und lädt den Byte Code für die Klassendefinition. Nach dem Laden wird der Code erst verifiziert und dann erst die aktuelle Klasse erzeugt.
Neben dem Finden, Laden und Verifizieren übt der Class Loader andere sicherheits-bezogene Pflichten aus.
Erstens, der Class Loader unterbindet das Laden anderen java.* Packages über das Netzwerk. Dieses stellt sicher, das die JVM nicht ausgetrickst wird, indem falsche Repräsentationen von Kern Class Libraries geladen werden, die das Sicherheitsmodell durchbrechen könnten.
Zweitens, der Class Loader bietet für jede Klasse, die von andren Orten geladen wurde, einen extra Namensraum. So kommt es bei Klassen mit gleichen Namen, die von verschiedenen Rechnern geladen wurden, zu keinen Konflikten. Diese Klassen können in der JVM auch nicht miteinander kommunizieren. Die Informationen bleiben also gekapselt, so daß untrusted Code niemals Informationen von trusted Code erhalten kann.
Der Class Loader ist eine abstrakte Klasse. Wird diese kreiert muß nur eine abstrakte Methode, in Unterklassen, überschrieben werden.

```
protected abstract Class loadClass(String name, boolean resolve) throws
ClassNotFoundException
```

In loadClass müssen im Wesentlichen fünf Operationen durch geführt werden.
Die fünf Operationen seinen kurz erläutert:

1. Finde heraus ob die Klasse bereits geladen ist. Die ClassLoader Superklasse besitzt einen private Hashtabelle von geladenen Klassen. Um herauszufinden ob etwas geladen wurde, werden die Methoden findLoadedClass() und findSystemClass() benutzt. Es ist notwendig beide zu überprüfen. Nicht wegen der Klasse selbst, sondern wegen den implizit geladenen Superklassen der Klasse (wie Object).

```
// in loadClass()
Class c = findLoadedClass (name);
if (c == null) {
try {
        c = findSystemClass (name);
        }
        catch (Exception e) {
                //ignoriere  exceptions.
        }
```

2. Wenn nichts geladen wurde, dann lade die Daten der Klasse. Das kann über JDBC aus einer Datenbank, über eine Netzwerkverbindung per URL oder durch einen andern eigenen Mechanismus erfolgen.

```
if (c == null) {
// meine Definition von loadClassData
byte data[] = loadClassData(name);
```

3. Rufe defineClass() um die Bytes in eine Klasse zu konvertieren. Dieses erzeugt einen ClassFormatError, wenn der Bytearray ungültig ist. Üblicherweise erzeugt man eine ClassNotFoundException, wenn defineClass() den Wert null zurückliefert.

```
c = defineClass (name, data, 0, data.length);
if (c == null)
throw new ClassNotFoundException (name);
```

4. Verankere (resolve), insofern veranlaßt, die Klasse. Bis die Klasse nicht verankert ist können keine Instanzen erzeugt oder Methoden aufgerufen werden. Das Flag kann z.B. falsch sein um nur die Existents einer Klasse zu prüfen.

```
if (resolve)
resolveClass (c);
```

5. Übergebe die neue erzeugte Klasse.

```
return c;
```

Es ist möglich, daß *name* in loadClass eine Ortsangabe mit einschließt, dann muß der Name von der Ortsangabe ausgeparst werden und es werden einige Änderungen im Code nötig.

Es gibt noch viele weitere mögliche Operationen die während des Ladens von Klasse eingesetzt werden können, was die Flexibilität der JVM zeigt.

Für einen üblichen Class Loader könnte man folgenden Code benutzen:

```
ClassLoader loader = new CustomClassLoader(params);
Class c = loader.loadClass ("TheClass", true);
TheClass tc = (TheClass)c.newInstance();
```

JDK 1.2 führte den URLClassLoader ein um die Notwendigkeit übliche Class Loader zu erzeugen zu verringern. URLClassLoader ist eine spezielle Unterklasse des neuen SecureClassLoader, welcher es erlaubt Rechte je nach Herkunftsort der geladenen Klasse zu vergeben. (darüber später mehr.)
Mit dem URLClassLoader kann jede Klasse oder Ort komplett in Form einer URL angegeben werden (z.B. file:, http:, jar: URL) und durch diesen geladen werden.
Möchte man Operationen wie Verschlüsselung anwenden oder sollen die Klassen Bytes direkt aus einer Datenbank gelesen werden, so ist es nur nötig eine Unterklasse von URLClassLoader zu erzeugen und dort die gewünschte Funktionalität zu implementieren.

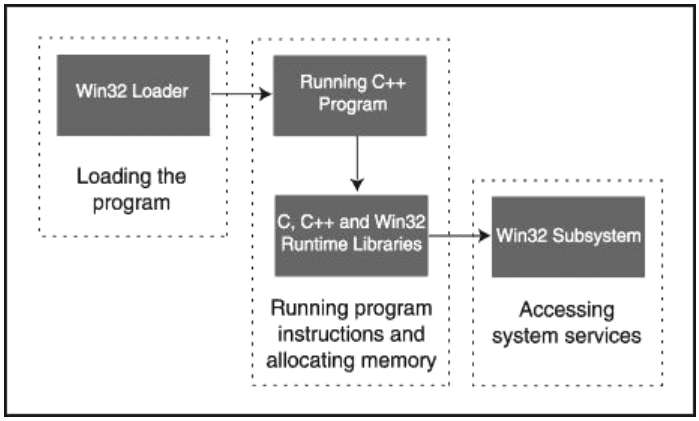
Der einfache Gebrauch von URLClassLoader erzwingt also kein Ableiten in Unterklassen. Es muß dem URLClassLoader nur mitgeteilt werden wo er die Klassen zu finden hat. Jede URL, die mit einem / endet wird als Verzeichnis interpretiert alles andere wird als JAR Datei erkannt.

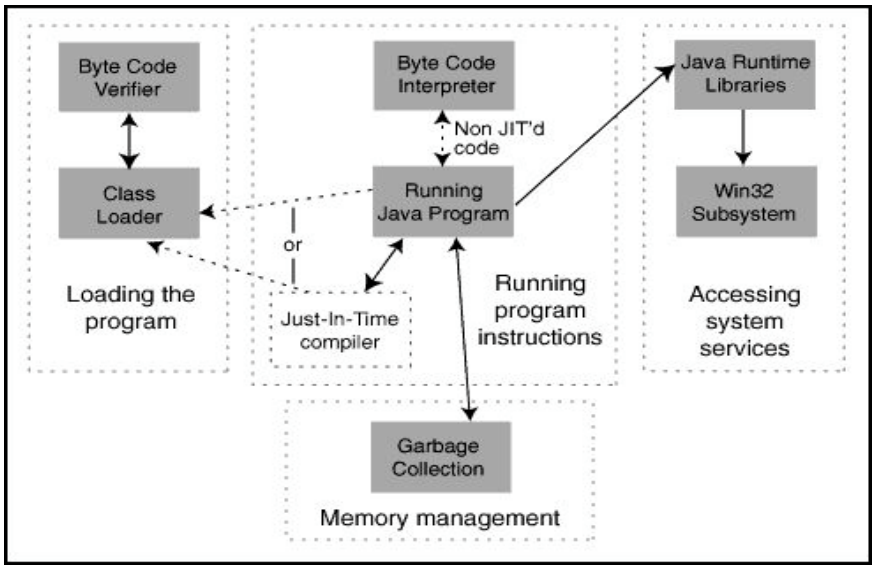
```
try {
        URL urlList[] = {
        new URL ("http://java.sun.com/share/classes/"),
        new URL ("http://www.jars.com/jediSearch.zip"),
        new URL ("http://java.miningco.com/library/weekly/"),
        new File ("myJar.jar").toURL()};
        ClassLoader loader = new URLClassLoader (urlList);
        Class c = loader.loadClass ("TheClass");
        TheClass tc = (TheClass)c.newInstance();
        }
catch (MalformedURLException e) {
        // load classes another way or display error message
        }
```

Weitere Operationen, wie z.B. Entschlüsselung, können hier auch durchgeführt werden. Wenn Entschlüsselung durchgeführt werden soll, muß der Algorithmus in den Byte Code des Class Loaders eingebettet sein. Das hat den Nachteil, daß wenn jemand den Byte Code decompiliert er Zugriff auf den Algorithmus bekommt und damit Zugriff auf den verschlüsselten Byte Code.

7.3.3 Runtime Checking

Nachdem Klassen geladen wurden ist der nächste durchzuführende Level der JVM Security das Runtime Checking. Wegen späten Bindung, die die JVM unterstützt, wird die Typprüfung bei Zuweisungen und die Überprüfung der Arraygrenzen erst sehr spät, nämlich zur Laufzeit (runtime) vollzogen. Während einige der Typprüfungen schon zur Compilezeit möglich sind, gibt es andere spezielle Typen von Objekten, deren zukünftiger Status vorher nicht bestimmbar ist und daher zur Laufzeit getestet werden muß.
In diesen Fällen stellt die JVM sicher, daß nur ordnungsgemäße Zuordnungen durchgeführt werden. Sie trägt auch dafür Sorge das jedes Element Teil der Klassenhierarchie ist. Während die Entfernung von Pointer Arithmetik eine reine Compiler-Typ Operation ist, wird der Test auf gültige Arraygrenzen zur Laufzeit vollzogen. Wenn ein Array erzeugt wird, dann wird auch seine Größe festgelegt. Der Zugriff auf einzelne Elemente läuft dann später über die Angabe der Position. In Sprachen wie C oder C++ kann man jeden Wert als Position angeben, was zur Folge hat, das man auf andere Speicherbereiche, als die des Arrays, zugreifen kann. In Java kennen Arrays ihre Größe. Wird eine ungültige Position im Array angegeben, so wird eine ArrayIndexOutOfBoundException ausgelöst, was den Zugriff auf Speicher, der nicht zum Array gehört verbietet.
Die folgenden Grafiken sollten die Unterschiede bei der Ausführung von C/C++ und Java deutlich machen:





7.3.4 Security Manager

In diesem Abschnitt wird beschrieben, wie man Sicherheitseinstellungen bis JDK 1.1 bei Applikationen mit einem eigenen SecurityManager vornehmen konnte. Die Zuteilung der Rechte an unterschiedliche Anwendungen erfolgt bis dahin durch Ableiten einer eigenen Klasse von SecurityManager. Dort werden die vergebenen Rechte explizit in Form von Quellcode festgelegt.

Das JDK 1.2 braucht einen solchen Mechanismus nicht mehr, da die Rechte alle im Klartext in der Policy-Datei festgehalten werden. Die Rechte sind also unabhängig vom Quellcode. Um einen Eindruck von der Entwicklung des JDK zu bekommen wird an dieser Stelle dennoch die Vorgehensweise bei der Definition eines eigenen SecurityManagers vorgeführt.

Applikationen besitzen alle Rechte, die andere Standalone-Anwendungen auch haben. Somit sind sie per Voreinstellung keinen Restriktionen eines SecurityManagers unterworfen.

Mit Hilfe der Klasse ClassLoader können aber auch in eine Applikation zur Laufzeit neue Klassen über eine Verbindung zu einem anderem Rechner geladen werden. Es kann nun z.B. sein, daß man nur die Funktion einer Klasse, die geladen wird, kennt. Hat man nur Zugriff auf den Bytecode, kann man nicht beurteilen, ob diese Klasse nicht doch unbemerkt auf Dateien auf dem lokalen Rechner zugreift und/oder Informationen über eine Netzverbindung verschickt. Da bei Applikationen kein SecurityManager festgelegt ist, werden auch keine Operationen überprüft.

Sollen in einer Applikation trotzdem bestimmte Operationen nicht möglich sein, wie z. B. das Löschen von Dateien, so ist es notwendig, einen SecurityManager in der Applikation zu installieren. Hierzu verwendet man die Klasse SecurityManager, die im Paket System definiert ist.

Diese Klasse ist abstrakt. Um sie verwenden zu können, muß eine neue Klasse abgeleitet werden. In ihr sind Methoden definiert, die bei Ausführung bestimmter Operationen aufgerufen werden.

checkAccess(Thread)
Bei Zugriff auf einen Thread

checkConnect(String, int)
Bei Verbindungen zu einem Host

checkDelete(String)
Beim Löschen von Dateien

checkRead(String)
Beim lesenden Zugriff auf Dateien

checkWrite(String)
Beim schreibenden Zugriff auf Dateien

Dies sind jedoch längst nicht alle Überprüfungsmethoden, die ein SecurityManager besitzt. Die Klasse SecurityManager bietet noch eine Vielzahl weiterer Methoden, die zur Überwachung bestimmter Operationen dienen.

Wenn ein SecurityManager nun bestimmte Operationen unterbinden soll, müssen die dafür relevanten Methoden überschrieben werden.

Alle Methoden des SecurityManagers, die nicht implementiert werden, lösen in jedem Fall eine SecurityException aus.

Ist ein SecurityManager für die Überwachung bestimmter Operationen geschrieben, dann muß dieser installiert werden.

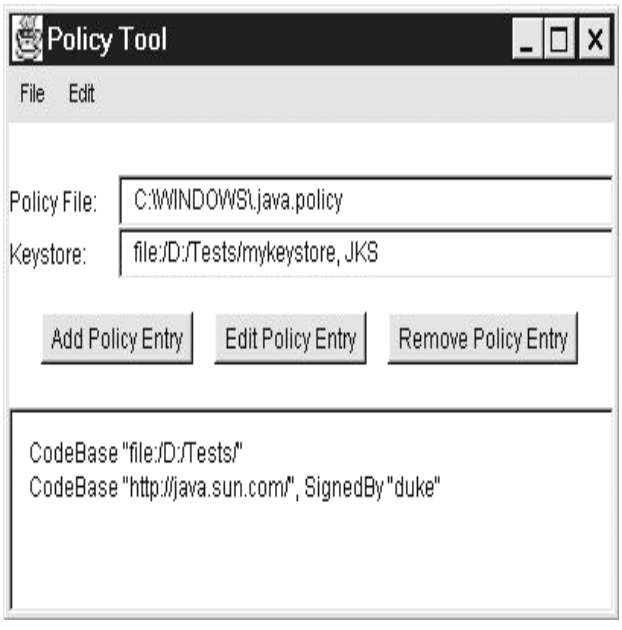
System.setSecurityManager(new MySecurityManager());

System.setSecurityManager() bekommt als Parameter ein Exemplar des zu installierenden SecurityManagers übergeben. Diese Methode kann nur einmal in einer Applikation aufgerufen werden. Ist schon ein SecurityManager installiert, wird bei einem zweiten Aufruf von setSecurityManager() eine SecurityException erzeugt.

7.3.5 Access Controller und Permissions

Da nur eine SecurityManager zugelassen ist ergeben sich einige Limitierungen. Es ergeben sich z.B. Probleme, wenn mehrere Clients verschiedene SecurityManager benötigen, um ordnungsgemäß zu Funktionieren. Dies kann nicht funktionieren, da nur ein SecurityManager installiert sein darf. Eine Lösung besteht darin, daß man den Source Code beider SecurityManager zu einem vereinigt. Die Realisierung ist sehr fehlerträchtig und auch nicht weiter skalierbar. Erst recht, wenn man sich vorstellt, man müsse 10 oder 20 SecurityManager vereinigen.
Der JDK 1.2 SecurityManager erweitert das Konzept eines internen Access Controlers von JDK 1.1, um die Möglichkeit sehr speziell Rechte den einzelnen Operationen zuordnen zu können.

Dies geschieht mit dem policytool Programm. Mit dem sehr gezielt individuelle Rechte gesetzt werden können (siehe Abbildung)



Die initialisierungs Policy, mit dem Namen java.policy, ist im Verzeichnis lib/security unter der installierten Laufzeitumgebung zu finden.
Eine weitere Policy ist im Benutzervereichnis (Home Directory), ebenfalls mit dem Namen java.policy zu finden.
Um die Rechte zu erweitern muß nun kein weiterer SecurityManager mehr installiert werden. Es wird mit dem policytool einfach eine weitere Policy generiert, in der die entsprechenden Rechte gesetzt sind. Oder es wird einfach eine bestehende Policy editiert.

Beim Programmstart kann man nun über die Systemeigenschaft java.security.policy die zu benutzende Policy spezifizieren.
Oder, wenn man den Security Access beobachten möchte, kann man die Systemeigenschaft java.security.debug setzen. Um alle Debugoptionen zu sehen benutzt man die Help-Einstellung:

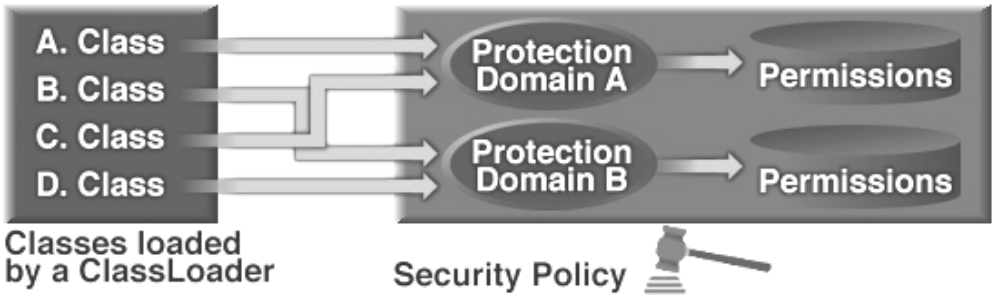
java -Djava.security.debug=help myFile

all	schaltet das komplette Debugging an
access	gibt alle checkPermission Ergebnisse aus.
jar	jar Verifizierung
policy	Laden und Rechtevergabe
scl	Rechtevergabe beim SecureClassLoader

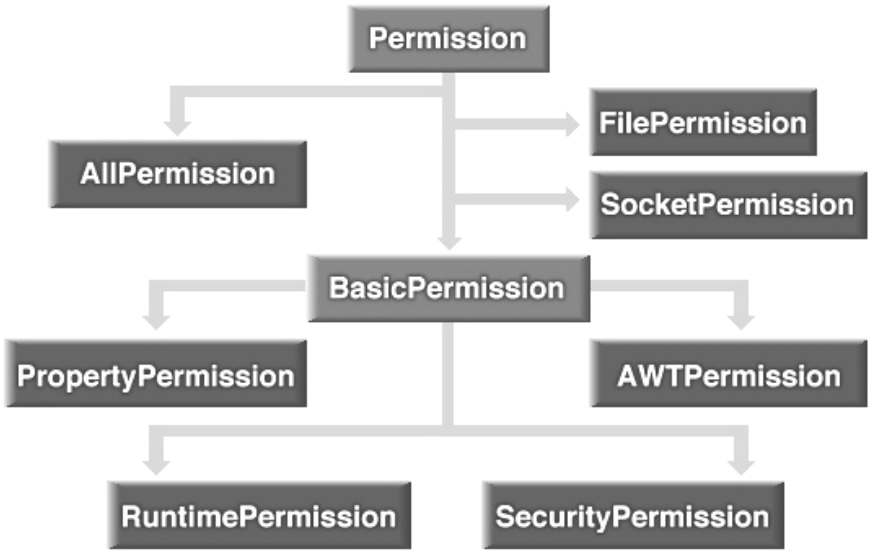
Die folgenden Optionen können mit access genutzt werden:

stack	inclusive Stack Trace
domain	gibt alle Domänen in ihrem Kontext aus
failure	Vor dem Auftreten Exception wird der Stack ausgegeben und die Domäne die dieses Recht nicht besitzt.

Bei der Zuweisung von Permissions kann man diese nach dem Signierer des Codes und/oder dem Ursprung (Ort) des Codes (der codebase) basieren lassen oder man vergibt das Recht an jeden. (siehe Abbildung)



Die Klasse java.security.Permission dient als Basisklasse für alle 1.2 Klassen, die sich auf Permissions beziehen.



Ein Eintrag in das Policyfile könnte z.B. so aussehen:

```
grant signedBy "Andre", codeBase "http://www.sectrix.net" {
                permission java.io.FilePermission "c:\\temp\-";
                        }
```

Hiermit würde Code, der von dem User Andre signiert wurde und von der Domäne http://www.sectrix.net geladen wurde, Schreibrechte auf das lokale temporäre Verzeichnis und dessen Unterverzeichnisse bekommen. Würde man anstelle des Minus (–) ein Stern (*) verwenden, würden die Rechte nur für das spezielle Verzeichnis gelten.

Neben der Vergabe solcher Rechte mit dem Policytool kann man mit dem AccessController Rechte prüfen. Sind diese Rechte nicht vorhanden wird eine AccessControlException ausgelöst, dies ist eine Unterklasse, der mehr generischen SecurityException. Beide sind Laufzeit Exceptions.

Das folgende Beispiel zeigt wie man überprüft, ob ein User Leserechte auf die Datei test.out in dem plattformspezifischen temporären Verzeichnis hat. (Dazu benötigt der User keine Leserechte auf die java.io.tmp.dir Eigenschaft, welche per Grundeinstellung nicht verfügbar ist.)

```
String tempPath = System.getProperty ("java.io.tmpdir");
File f = new File (tempPath, "test.out");
FilePermission perm =  new FilePermission(f.getAbsolutePath(), "read");
AccessController.checkPermission (perm);
```

Für die vom System vergebenen Rechte ist keine explizite, manuelle Prüfung notwendig. Dies erledigt das System.
Bei selbst definierten Rechten muß man dieses selbst testen. Dies soll anhand der Eigenschaft eines Beans dargestellt werden. Um die Eigenschaft zu lesen muß man die Leserechte besitzen. Zum verändern dieser Eigenschaft benötigt man also Schreibrechte.

```
private String state;
...
public String getState() {
MyPermission p = new MyPermission ("state", "read");
AccessController.checkPermission (p);
// throws exception if no permission
return state;
}

public void setState (String newValue) {
MyPermission p = new MyPermission ("state", "write");
AccessController.checkPermission (p);
// throws exception if no permission
state = newValue;
}
```

Möchte man Applikationen mit den gleichen Restriktionen wie Applets laufen lassen, dann kann man die Laufzeitumgebung mit der Systemeigenschaft java.security.manager starten lassen.

```
java -Djava.security.manager MyClass
```

7.4 Java API – Level Security

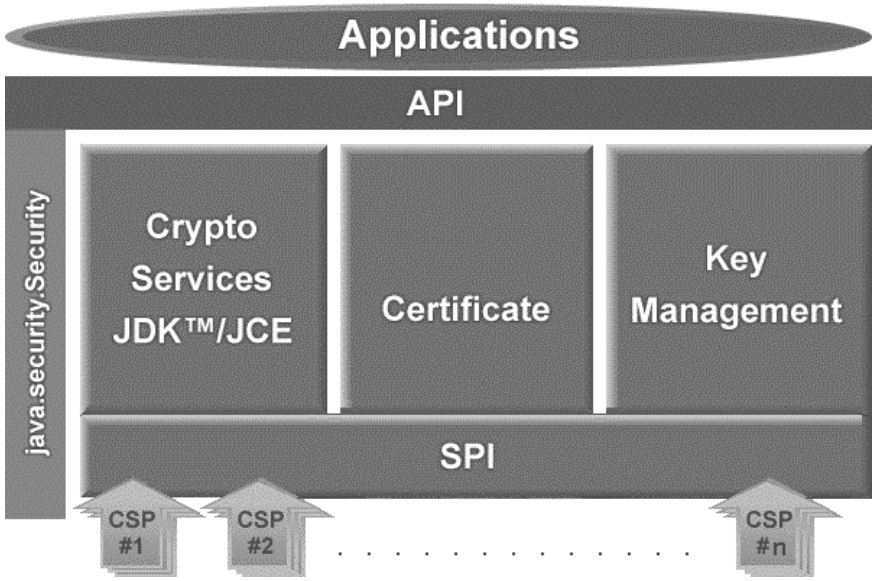
Neben denen schon diskutieren Sicherheitskonstrukten in der Sprache und der Virtual Machine bietet die Java Technologie mehrere Pakete, die sich als nützlich erweisen, wenn man sichere Applikationen bauen möchte.
In JDK 1.1 gibt es davon drei Pakte: java.security, java.security.acl und java.security.interfaces
Diese bilden die Java Cryptography Architecture (JCA). Die JCA ist der Framework, der Java Programmen kryptographische Möglichkeiten bietet.

In JDK 1.2 wurde der JCA von 1.1 modifiziert und es kamen die Pakte java.security.spec und java.security.cert hinzu. Dazu kommt noch die Java Cryptography Extension (JCE) (javax.crypto.*), die allerdings den US-Export Restriktionen unterliegt.
Die JCE erweitert das JDK 1.2 um weitere Funktionalitäten, wie z.B. sichere Streams, Schlüsselgenerierung und Verschlüsselungshilfen, Schlüsselaustausch, "message authentication code" (MAC) und mit der Möglichkeit (einfach) weitere Verschlüsselungalgorithmen hinzuzufügen.

Das JCA beinhaltet eine Provider Architektur, die mehrere interoperable Kryptographie Implementationen erlaubt. Der Ausdruck cryptographic service provider (CSP) oder einfach Provider bezieht sich auf ein oder mehrere Pakete, die konkrete Implementationen von Kryptographie.

Eine Engine Klasse definiert einen kryptographischen Dienst in abstrakter Weise (also ohne konkrete Implementation). Sie definiert API Methoden, die es Applikationen erlauben auf einen speziellen Typ von kryptographischen Service, wie einen Signatur Algorithmus, zu zugreifen.

Die Applikationsschnittstelle die von der Engine Klasse unterstützt wird service provider interface (SPI) genannt.



java.security

Das java.security Paket beinhaltet größtenteils abstrakte Klassen und Interfaces, Sicherheitskonzepte kapseln, wie z.B. Zertifikate, Schlüssel und Signaturen. Konkrete Implementationen werden, wie oben schon angedeutet, hier nicht gefunden. Diese können selbst programmiert oder von einem CSP bezogen werden

Es gibt also eine User und eine Provider Seite in diesem Paket. Der User, also der Programmierer, baut Applikationen mit Kryptoalgorithmen eines Providers. Unternehmen implementieren Algorithmen auf der Provider Seite. Um einen Algorithmus zu aktiveren, den ein Provider installiert hat gibt es zwei Wege:

Man aktualisiert die /lib/security/java.security Datei unter dem JRE Installation Verzeichnis um einen neuen Provider zu spezifizieren. Es muß mindestens ein Provider installiert sein. Als Voreinstellung ist bei dem JDK von SUN sun.security.provider.Sun eingetragen.

Oder man fügt den neuen Provider mit Security.addProvider (Provider). Während dies scheinbar, der zu bevorzugende Weg ist, ist er es jedoch nicht. Da, wenn ein besserer Provider (mit z.B. stärkerer Kryptographie) gefunden wird, das Programm erneut kompiliert und verteilt werden muß.

Im JCA 1.1, ist es einem Provider möglich für folgende Klassen Implementationen zu liefern: KeyPairGenerator, MessageDigest und Signature.
Man kann nach der spezifischen Implementation mit Hilfe der Methode getInstance() und dem Namen des gewünschten Algorithmus fragen:

```
MessageDigest md = MessageDigest.getInstance ("MD5");
KeyPairGenerator kpg =    KeyPairGenerator.getInstance ("DSA");
```

Das System findet den Provider der den gewünschten Algorithmus unterstützt, indem er seine sortierte Liste durchsucht und die spezielle Unterklasse zurück liefert. Oder man fragt direkt nach einem speziellen Provider:

```
kpg = KeyPairGenerator.getInstance ("DSA", "MageLang");
```

In JCA 1.2 gibt es zusätzlich die Klassen: AlgorithmParameterGenerator, AlgorithmParameters, CertificateFactory, KeyFactory, KeyStore und SecureRandom für die Provider Implementationen liefern kann.

java.security.acl

Das java.security.acl Paket definiert die Unterstützung von Access Control Lists.Diese können dazu benutzt werden um den Zugriff auf Ressourcen auf alle möglichen Arten zu beschränken. Das Paket besteht aus Interfaces und Exceptions.

java.security.interfaces

Das letzte JCA 1.1 Sicherheitspaket java.security.interfaces beinhaltet Interfaces zur nutzung des Digital Signature Algorithm (DSA). Man möchte z.B. den DSAKeyPairGenerator, nach dem erzeugen einer Instanz, initialisieren:

```
DSAKeyPairGenerator dkpg = (DSAKeyPairGenerator)
KeyPairGenerator.getInstance ("DSA");
DSAParams params = new DSAParams (aP, aQ, aG);
dkpg.initialize (params, new SecureRandom());
```

Im JCA 1.2 beinhaltet das java.security.interfaces Paket zusätzlich noch Interfaces für RSA (benannt nach den Erfindern: Ron Rivest, Adi Shamir und Leonard Adleman). Zusätzlich zur RSA public key Unterstützung gibt es für RSA private keys Unterstützung für Chinese Remainder Theorem (CRT) und ohne.

java.security.cert

Das JCA 1.2 Sicherheitspaket java.security.cert bietet eine Unterstützung zum Gebrach und der Generierung von Zertifikaten. Und Zugriffsmechanismen auf JAR Dateien über JarURLConnection und JarEntry. Es werden auch X.509 Zertifikate unterstützt.

java.security.spec

Das java.security.spec Paket beinhaltet Interfaces, die das Schlüssel Spezifikationsformat beschreiben. Diese Klassen erlauben es einfach Java Sicherheitsschlüssel, basieren auf Parameter von Tools außerhalb von JDK, zu generieren.

7.5 Sicherheit in Applets

Bei Applets spielt Sicherheit eine große Rolle, da sie in HTML-Seiten eingebunden werden können. Wenn man eine HTML-Seite in den Browser lädt, kann man vorher nicht wissen, was sich auf dieser Seite befindet. Ist dort ein Applet eingebunden, wird es sofort vom Browser geladen und gestartet. Der Benutzer, der eine Seite zum ersten Mal lädt, kennt normalerweise die Funktion evtl. eingebundener Applets vorher nicht. Der Programmierer des Applets könnte bei uneingeschränktem Zugriff auf den Client-Host Informationen bekommen, ohne daß der Betrachter der HTML-Seite etwas davon merkt. Der Internet Explorer und der Netscape Communicator bieten beide eine Option, mit der man einstellen kann, ob Java-Applets im Browser ausgeführt werden können oder nicht, doch dies setzt den Applets keine Sicherheitsrestriktionen.

Die Überwachung eines Applets unterliegt dem SecurityManager. Der SecurityManager überprüft bei einer sicherheitsrelevanten Operation, ob sie ausgeführt werden darf oder nicht.

Bevor der Bytecode im Browser ausgeführt wird, unterliegt er der Prüfung durch eine andere Instanz: dem Bytecode-Verifier. Der Bytecode-Verifier prüft den Code auf Korrektheit. Dadurch kommt der Bytecode nur dann zur Ausführung im Browser, wenn er korrekt ist und keine unerlaubten Befehlskombinationen enthält. Code, der gegen die Regeln des Bytecode-Verifiers verstößt, wird erst gar nicht ausgeführt.

Die Überwachung des Applets zur Laufzeit durch den SecurityManager ist der nächste Schritt.

Automatisch beim Start des Browsers bzw. des Appletviewers wird auch der SecurityManager gestartet. Dieser bleibt aktiv, bis der Browser bzw. Appletviewer beendet wird. Die Implementierung des SecurityManagers ist bei Applets vom Browser bzw. Appletviewer vorgegeben und kann nicht innerhalb eines Applets ersetzt werden.

Wird innerhalb eines Applets eine unerlaubte Operation ausgeführt, löst dies bis einschließlich JDK 1.1 eine SecurityException, beim JDK 1.2 eine AccessControlException aus.

Die für die Sicherheit eines Systems wichtigsten Operationen beinhalten unter anderem das Lesen bzw. Schreiben von Dateien und den Aufbau von Netzwerkverbindungen. Der SecurityManager schränkt die Benutzung dieser Operationen massiv ein. Hierbei muß man jedoch wiederum unterscheiden zwischen Applets:

- die im Browser in einer integrierten Virtual Machine gestartet werden
- die im Appletviewer oder mit dem Java-Plug-In gestartet werden
- die über Netz geladen werden
- die lokal geladen werden

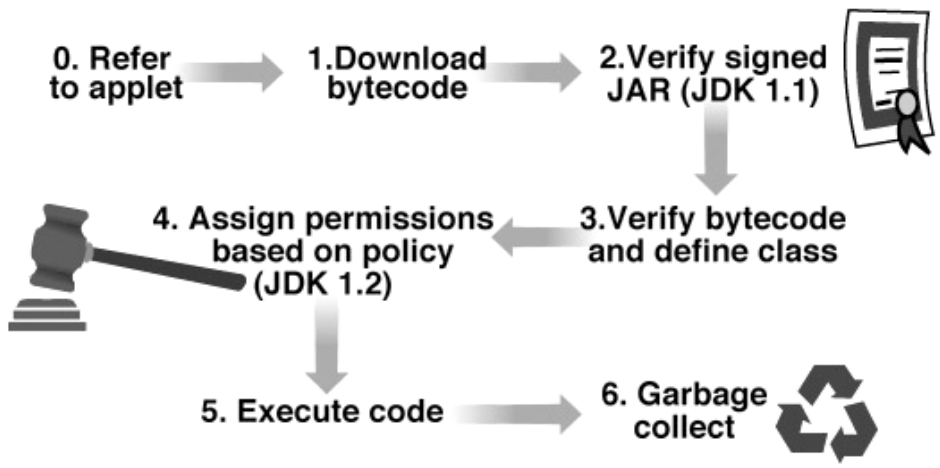
In Java ist mit lokal allerdings nicht nur gemeint, daß sich das Applet auf der lokalen Platte befindet. Zusätzlich muß das Verzeichnis, im CLASSPATH liegen, um als lokal zu gelten. In Verzeichnissen, die in der Umgebungsvariable CLASSPATH stehen, sollten nur Klassen abgelegt werden, die vom Benutzer verifiziert worden sind. Steht eine Klasse in einem Verzeichnis des CLASSPATH, so besitzt sie gegenüber anderen Klassen erweiterte Zugriffsmöglichkeiten.

Diese Unterteilung beruht auf dem »name space«- Konzept in Java. In einem »name space« befinden sich Klassen mit gleichem Herkunftsort. Jede Klasse, die geladen wurde, wird in einen »name space« eingefügt. Auf diese Weise können mehrere Klassen mit gleichem Namen, aber unterschiedlichem Herkunftsort geladen werden. Die Identifizierung einer Klasse erfolgt über Name und Herkunftsort. Nur Klassen, die sich auf der lokalen Platte und gleichzeitig im CLASSPATH befinden, werden dem »name space« für lokale Klassen zugeordnet und besitzen erweiterte Zugriffsrechte. Alle anderen Klassen werden nur anhand ihres Herkunftsortes identifiziert. Applets, die lokal geladen werden, können ein Frame oder ein Window anzeigen, ohne daß im Fenster eine Warnung erscheint: Dem Applet wird mehr Vertrauen entgegengebracht, wenn es lokal ist.

Im folgenden werden die Sicherheitskonzepte für Applets im Internet Explorer, im Netscape Communicator, im Appletviewer und im Java-Plug-In erläutert. Werden keine besonderen Einstellungen vorgenommen, gelten für oben genannte Appletviewer normalerweise folgende Regeln:

- Applets dürfen nicht auf lokale Dateien zugreifen.
- Applets dürfen nur zu dem Rechner Netzwerkverbindungen aufbauen, von dem sie geladen wurden.

Der Lebenszyklus eines Applets und dessen Sicherheitsüberprüfungen wird in folgender Grafik deutlich:



8 Socketprogrammierung mit Java 2

8.1 Grundlagen

Eine Socket-Verbindung zwischen zwei Rechnern kann man durch vier Parameter charakterisieren:

- Rechnername des Quellrechners
- Portnummer des Quellrechners
- Rechnername des Zielrechners
- Portnummer des Zielrechners

Durch diese vier Werte ist die Verbindung zwischen zwei Rechnern eindeutig beschrieben.

Unter dem Rechnernamen und der Portnummer des Zielrechners muß ein Server-Dienst zur Verfügung stehen, um eine Kommunikationverbindung aufzubauen. Der Server wartet unter dieser Adresse auf eingehende Anfragen von Clients.

Will ein Client mit einem Server kommunizieren, versucht er über den Rechnernamen und die Portnummer (beide müßten bekannt sein) eine Verbindung herzustellen. Beim Aufbau der Verbindung teilt der Client dem Server den Namen und die Portnummer seines Rechners mit. Danach haben beide Kommunikationspartner die Informationen, die sie benötigen, um Daten miteinander auszutauschen.

Die Kommunikation zwischen zwei Rechnern kann sowohl verbindungsorientiert über Streamsockets (TCP) als auch verbindungslos über Datagramsockets (UDP) durchgeführt werden.

Die Übertragung der Daten erfolgt bei beiden Kommunikationsformen mit Hilfe sogenannter Pakete (IP). Jedes Paket kann eine bestimmte Menge an Daten aufnehmen. Ist die Menge der zu verschickenden Daten größer als das Fassungsvermögen eines Pakets, was in den allermeisten Fällen zutrifft, werden die Daten auf mehrere Pakete aufgeteilt, die nacheinander zum Zielrechner geschickt werden.

Das Socket-Konzept ist in Java durch drei unterschiedliche Klassen geprägt:

Socket, DatagramSocket
Mit diesen Klassen kommt der Programmierer am häufigsten in Kontakt. Über ihre Schnittstellen wird die Kommunikation zwischen zwei Programmen abgewickelt.

SocketImpl, DatagramSocketImpl
Diese Klassen enthalten die eigentliche Implementierung eines Sockets. SocketImpl und DatagramSocketImpl sind abstrakte Klassen, die meist mit native-Methoden implementiert werden. Das ist notwendig, um die Umsetzung der Socket-Kommunikation auf eine Plattform vornehmen zu können.

SocketImplFactory
Diese Klasse dient der Erzeugung einer bestimmten Socket-Implementierung. Jeder vom Programmierer verwendete Socket wird von einer SocketImplFactory erzeugt. Auch diese

Klasse ist abstrakt. Sie enthält nur die Methode createSocketImpl(), die als Ergebnis ein Exemplar der Klasse SocketImpl zurück liefert.

Durch die SocketImplFactory ist also das Verhalten der Sockets festgelegt, die in einer Anwendung benutzt werden. Die Implementierung ist in den entsprechenden SocketImpl-Klassen verborgen. Durch die Kapselung der Implementierung von der einheitlichen Schnittstelle, die in der Klasse Socket bzw. DatagramSocket definiert wird, ist es einfach, durch wenige Änderungen am Programm eine andere Socket-Implementierung für die Kommunikation zu verwenden. So könnte man sich z. B. vorstellen, Sockets zu implementieren, die die zu übertragenden Daten verschlüsseln.

Bei der Ersetzung müssen folgende Schritte vorgenommen werden:
Ableiten einer eigenen Klasse von SocketImpl und Implementierung der Funktionen.
Ableiten einer Klasse von SocketImplFactory und Implementierung der Methode createSocketImpl(). Die einfachste Implementierung von createSocketImpl() ist das Erzeugen eines Exemplars von SocketImpl und Rückgabe des erzeugten Objekts.
Setzen der SocketImplFactory im Programm. In jeder Anwendung kann die SocketImplFactory vom Programmierer gesetzt werden.

Es existiert eine SocketImplFactory sowohl für Client- als auch für Server-Seite. Deshalb enthalten die Klassen Socket und ServerSocket Methoden, mit denen die SocketImplFactory für eine Applikation gesetzt werden kann. Nachdem eine SocketImplFactory gesetzt ist, werden alle benötigten Sockets von ihr erzeugt. Sowohl für Client-Sockets als auch für Server-Sockets kann die SocketImplFactory allerdings nur einmal gesetzt werden. Beim Versuch, zum zweiten Mal die SocketImplFactory zu setzen, wird eine SocketException ausgelöst. In Applets ist das Setzen einer SocketImplFactory nicht möglich.

8.2 Streamsocket / TCP-Socket

Mit Streamsockets wird eine feste TCP-Verbindung zu einem anderen Rechner aufgebaut, die für die Dauer einer Übertragung bestehen bleibt.

Allgemein beinhaltet die Socket-Programmierung folgende Schritte:

- Aufbau der Verbindung
- Initialisierung der Streams
- Abwicklung der Ein- und Ausgabe
- Abbau der Verbindung

Eine Verbindung zu diesem Server kann mit

server = new Socket(*Servername, port*);

aufgebaut werden.
server ist hierbei eine Objektvariable des Typs Socket.

Dem Konstruktor von Socket wird der Host-Name und die Port-Nummer des Rechners übergeben, zu dem der Kontakt hergestellt werden soll.

Wenn kein Socket erzeugt werden kann, können folgende Exceptions ausgelöst werden:

UnknownHostException (wenn der Host-Name falsch angegeben wurde)
IOException

Um über den angelegten Socket mit dem anderen Rechner zu kommunizieren, benötigt man Methoden, über die die Ein- und Ausgabe abgewickelt werden können:

Die Klasse Socket besitzt die Methoden getInputStream() und getOutputStream(), die als Ergebnis die entsprechenden Ein- und Ausgabe-Streams liefern.

Soll die Verbindung wieder abgebaut werden, muß man die Methode close() des Sockets aufrufen:

server.close();

Auch bei dieser Methode kann eine IOException ausgelöst werden.

8.3 Datagramm-Sockets / UDP-Socket

Die Übertragung mit Datagramm-Sockets benötigt keine feste Verbindung zum Zielrechner. Sie beruht auf dem Verschicken von Datagrammen. (UDP).

Im Gegensatz zu Streamsockets, die auf einen bestimmten Host fixiert sind und nur mit diesem Daten austauschen können, sind Datagramm-Sockets flexibler. Über einen Datagramm-Socket kann mit verschiedenen Rechnern kommuniziert werden. Der jeweils

zu kontaktierende Host wird nicht bei der Initialisierung des Sockets festgelegt, sondern im abgeschickten Datagramm vermerkt. Ein Datagramm-Socket wird in Java durch die Klasse DatagramSocket repräsentiert.

Eine Instanz von DatagramSocket erzeugt man mit:

```
socket = new DatagramSocket();
```

Der Konstruktor bekommt in diesem Fall keine Parameter übergeben. Im Unterschied zu Stream-Sockets, die scheinbar direkt mit einem anderen Rechner verbunden sind, ist ein Datagramm-Socket nur an einen Port am lokalen Rechner gebunden.

Die Port-Nummer, an die der Socket gebunden ist, wird auf diese Weise aus der Liste der freien Port-Nummern beliebig gewählt. Diese Art der Initialisierung wird vor allem bei der Programmierung von Clients gewählt.

Es ist jedoch auch möglich, dem Konstruktor die Port-Nummer explizit zu übergeben, z. B.:

```
socket = new DatagramSocket(port);
```

Hierbei muß jedoch darauf geachtet werden, daß der gewählte Port nicht schon von einem anderen Dienst belegt ist. Ist dies der Fall, wird eine SocketException ausgelöst. Diese Technik wird hauptsächlich bei der Server-Programmierung auf Datagrammbasis verwendet, da ein Server üblicherweise unter einer bestimmten Port-Nummer zu finden ist. Das ist nur gewährleistet, wenn die Port-Nummer explizit bei der Initialisierung übergeben wird.

Zum Verschicken eines Datagramms braucht man ein Exemplar der Klasse InetAddress. Dieses beinhaltet die Adresse des Server-Hosts. Allgemein kann man die einem Host-Namen zugehörige InetAddress durch Aufruf der statischen Methode getByName(String) der Klasse InetAddress ermitteln.

Die Kommunikation über Datagramm-Sockets erfolgt mit sogenannten Datagrammpaketen. Java stellt hierfür die Klasse DatagramPacket zur Verfügung. Eine Kommunikation ist erst möglich, wenn ein Exemplar dieser Klasse vorhanden ist:

```
packet = new DatagramPacket(buffer, 256, address, port);
```

Dem Konstruktor werden folgende Parameter übergeben:

- Bytepuffer
- Größe des Bytepuffers
- Adresse des Zielrechners
- Port-Nummer des Dienstes, der auf dem Zielrechner erreicht werden soll

Seit dem JDK 1.2 verfügt die Klasse DatagramSocket über die Methoden connect() und disconnect().
connect() besitzt zwei Parameter: Ein Exemplar von InetAddress und eine Portnummer. Nach dem Aufruf von connect() können nur noch Datagramme an die angegebene Adresse verschickt und empfangen werden. Nach dem Aufruf von disconnect() können Pakete wieder an beliebige Adressen verschickt bzw. von beliebigen Adressen empfangen werden.

Das Datagramm kann mit der Methode send() verschickt werden:

```
socket.send(packet);
```

Ist das Paket einmal abgeschickt, wartet das Programm auf die Antwort des Servers:

```
socket.receive(packet);
```

receive() erwartet ein Exemplar der Klasse DatagramPacket als Parameter. Die Programmausführung wird so lange blockiert, bis ein Paket eingetroffen ist.

Wenn ein Applet z. B. Datagramme nur erhält, aber nicht verschickt, ist es nicht nötig, Port-Nummer und InetAddress eines Rechners anzugeben. Zu diesem Zweck besitzt DatagramPacket einen zweiten Konstruktor:

```
packet = new DatagramPacket(buffer, 256);
```

Ihm wird nur der Bytepuffer und die Größe dieses Puffers übergegeben.

Auch send() und receive() können eine IOException verursachen.
Wird der Versuch unternommen, Pakete zu empfangen, deren Inhalt größer als die Kapazität des DatagramPacket-Objektes ist, so werden alle Daten nach Überschreitung der Kapazitätsgrenze verworfen. Die Paketgrößen zwischen Sender und Empfänger sollten also übereinstimmen.

Ist ein Paket angekommen, kann man mit verschiedenen Methoden auf dessen Daten zugreifen:

getAddress()
Liefert die InetAddress des Host.
getPort()
Liefert die Port-Nummer des Dienstes.
getData()
Liefert Daten in Form eines Arrays von Bytes.
getLength()
Liefert die Größe des Paketes.

Wie auch Streamsockets sollten Datagramm-Sockets immer wieder geschlossen werden.

```
socket.close();   // Schließen des Sockets
```

8.4 Multicast-Socket

Mit der Klasse DatagramSocket ist es auch möglich, Multicast-Pakete zu versenden. Hierzu muß man einfach als Zieladresse eine Multicast-Adresse angeben. Zum Empfangen von Multicastpaketen muß die Klasse MulticastSocket benutzt werden. Dort sind Methoden definiert, die es erlauben, sich Multicast-Gruppen anzuschließen bzw. zu verlassen.

8.5 Client – / Serveranwendung

Ein Server stellt anderen Programmen einen Dienst zur Verfügung. Er wartet auf Anfragen (lokal oder innerhalb eines Netzes) und antwortet mit der gewünschten Aktion. Ein http-Server z. B. wartet auf Anfragen eines Browsers. Trifft eine Anfrage ein, schickt der Server dem Browser das entsprechende Dokument. Er »bedient« sozusagen den Client (in diesem Fall den Browser). Die meisten Server können gleichzeitig mit mehreren Programmen kommunizieren.

Ein Client hingegen ist ein Programm, das den Dienst, den der Server bietet, benutzt (z. B. Browser). Der Browser z. B. fordert vom Server ein Dokument an.

Prinzipiell gibt es zwei verschiedene Techniken, wie man einen Server realisieren kann. Bei beiden Techniken wird vorausgesetzt, daß ein Server gleichzeitig mehrere Clients bedienen kann:

iterativ

Bei dem iterativen Prinzip werden die einzelnen Clients vom Server jeweils sequentiell bedient. Der Server überprüft in bestimmten Zeitabständen jeweils, ob von den einzelnen Clients Daten angekommen sind, und bearbeitet die Anfragen gegebenenfalls. Alle Clients werden in diesem Fall im selben Thread bedient und somit sequentiell abgearbeitet.

konkurrierend (multithreaded)

Bei dem konkurrierenden Prinzip wird jeder Client vom Server in einem eigenen Thread bedient. Die Bedienung der einzelnen Clients erfolgt in diesem Fall nebenläufig.

In allen Fällen, in denen die Anfrage eines Clients lange dauern kann, ist ein konkurrierender (multithreaded) Server einem iterativen Server vorzuziehen. Das liegt daran, daß ein konkurrierender Server Clients nebenläufig dienen kann. Besonders einfach ist die Architektur eines konkurrierenden Servers, wenn zwischen den Clients, die gleichzeitig mit dem Server verbunden sind, keine Daten ausgetauscht werden müssen. Das ist z. B. bei Diensten wie ftp der Fall. Findet ein Datenaustausch zwischen Clients statt, muß eine Synchronisation zwischen den Clients stattfinden.

9 Schlußwort

Das Hauptproblem scheint mir zu sein, daß es immer noch Implementationsfehler gibt. Die meisten hier beschriebenen Probleme sind zwar behoben, aber keiner weiß wieviele Fehler noch existieren und wieviele neue Fehler durch die Fixes entstanden sind.
Fehler in der Spezifikation sind bisher nicht bekannt, so daß man auf ein sicheres Java in der Zukunft hoffen kann.

Mögliche Gründe für die Implementationsfehler sind z.B.:

- die enorme Komplexität des Java Security Model (was ich auch bei der Erstellung dieser Arbeit merken mußte)
- die moderne Softwareentwicklung steht unter einem hohen Marktdruck (was modernen Marketingstrategien zu verdanken ist)

So entschließt man sich entweder Java im Browser zu deaktivieren und keine Java Applikationen zu benutzen (Was der einzig sichere Weg ist!).

Oder man beachtet folgende Regeln:

Regeln für Anwender:

U1. Immer die neusten Browser verwenden und auf Sicherheitswarnungen der Hersteller achten!
U2. Nur Applets von vertrauenswürdigen Sites herunterladen!
U3. Die java.policy im Benutzerverzeichnis sollte nur editiert werden, wenn man genau weiß was man tut!

Regeln für Admins:

A1. Firewall richtig konfigurieren, so das Verbindungen nur von innen nach außen aufgebaut werden können (SYN!)
A2. User kontrollieren, so daß immer neuste Browser und Sicherheitspatches verwendet werden!
A3. java.policy in den Benutzerverzeichnissen kontrollieren oder diese möglichst sperren!

Regeln für Entwickler [McGraw 99]:

D1. Der Entwickler sollte sich nicht auf die Initialisierung seiner Objekte verlassen! Es ist auch ohne Konstruktor möglich ein Objekt zu erzeugen, das dann nicht initialisiert ist. Eine Klasse sollte also so geschrieben werden, das ein Objekt, bevor es irgend etwas tut, überprüft, ob es initialisiert ist.

D2. Den Zugriff auf Klassen, Methoden und Variable einschränken, diese also möglichst *private* deklarieren.

D3. Möglichst alles *final* deklarieren um ein Überladen zu verhindern!

D4. Man sollte sich nicht darauf verlassen, das seine private oder protected Klassen in einem package sicher seien. Ein Angreifer könnte eine maliziöse Klasse in das package einbringen und so Zugriff erlangen.

D5. Keine inneren Klassen benutzen! Viele Javabücher behaupten, diese seien nur von ihrer äußeren Klasse zugreifbar. Dies stimmt so nicht! Der Compiler übersetzt sie in normale Klassen, da der Bytecode nicht das Konzept der inneren Klassen kennt .

D6. Verhindere Code zu signieren! Nicht signierter Code hat keine besonderen Privilegien und kann damit auch keinen Schaden anrichten.

D7. Werden spezielle Privilegien benötigt, so das der Code signiert werden muß, sollte der ganze Code in ein File geschrieben werden um eine mix-and-match Attacke (wird in dieser Arbeit nicht weiter ausgeführt) zu erschweren.

D8. Klassen nicht Klonbar machen! Ein Angreifer könnte ansonsten neue Instanzen der Klasse erzeugen, ohne den Konstruktor der Klasse zu benutzen.

D9. Klassen nicht serialisierbar machen! Die innere Struktur eines Objekts ist sonst leicht einsehbar.

D10. Klassen nicht deserialisierbar machen, um Manipulationen zu vermeiden.

D11. Keine Klassen mit ihren Namen vergleichen, ansonsten besteht wieder die Gefahr einer mix-and-match Attacke.

D12. Keine schützenswerte Information in den Code einbringen, diese können leicht ausgelesen werden.

10 Abbildungsverzeichnis

[Baccala 97] Brent Baccala, Connected: An Internet Encyclopedia, www.freesoft.org/CIE/index.htm

[firewall1] Internet Ireland - Intranet and Security Services, http://ios.internet-ireland.ie/network/intranet.html

11 Quellenangaben

[Bellovin 89] S.M. Belloin, “Security Problems in the TCP/IP Protocol Suite”, Computer Communication Review, Vol. 19, No. 2, pp. 32-48, April 1989

[Berg 00] Clifford J. Berg, "Advanced Java 2 development for enterprise applications", Sun Microsystems Press/Prentice Hall PTR 2000

[Hoque 00] Faisal Hoque, "E-enterprise", Cambridge Univ. Press 2000

[Huegen 98] Craig A. Huegen, “The Latest in Denail of Service Attacks: “Smurfing”, Description and nformation to Minimize Effects”, April 1989

[Kerner 92] Helmut Kerner, „Rechnernetze nach OSI“, Addison-Wesley 1992

[Kyas 98] Othmar Kyas, "Sicherheit im Internet", Internat. Thomson Publ. 1998

[Li Gong 99] Li Gong, "Inside java 2 platform security", Addison-Wesley 2000

[Lienemann 00] Gerhard Lienemann, "TCP/IP-Grundlagen", Heise 2000

[Mahmoud 00] Qusay H. Mahmoud, "Distributed programming with Java", Manning 2000

[McGraw 99] Gary McGraw, Edward W. Felten, "Securing Java", Wiley 1999

[Middendorf 99] Stefan Middendorf, Reiner Singer, “ Java Programmierhandbuch und Referenz füt die Java-2-Plattform”, dpunkt 1999

[Monson-Haefel 99] Richard Monson-Haefel, "Enterprise JavaBeans", O'Reilly 1999

[Oaks 98] Scott Oaks, "Java security", OReilly 1998

[Orfali 98] Robert Orfali, Dan Harkey, "Client/Server programming witsch Java and CORBA", Wiley 1998

[Seshadri 99] Govind Seshadri, Gopalan Suresh Raj, "Enterprise Java computing" Cambridge univ. Press 1999

[Slama 99] Dirk Slama, Jason Garbis, Perry Russel, "Enterprise CORBA", Prentice hall 1999

[Smith 98] Richard E. Smith, "Internet-Kryptographie", Addison-Wesley Longmann 1998
[Sohr 00] Karsten Sohr, “Sandkastenspiele”, ct-Magazin 11/2000 S. 226

[Vitek 99] Jan Vitek, "Secure Internet programming", Springer 1999

[Vogel 98] Andreas Vogel, Keith Duddy, "Java programming with CORBA", Wiley 1998

Internetquellen (alle links vom 22.05.2000)

[Kimera 00] Kimera: A Java System Architecture, http://kimera.cs.washington.edu/

[Lee 99] Berners-Lee: Weaving the Web, http://www.w3.org/People/Berners-Lee/Weaving/Overview.html

[McQueen 00] Miles McQueen Java Virtual Machine Security and the Brown Orifice Attack, http://www.sans.org/infosecFAQ/java_sec.htm

[mladue 00] Hostile Applets Home Page, http://metro.to/mladue/hostile-applets/index.html

[Nachenberg 99] Carey Nachenberg, JavaApp.BeanHive , http://www.symantec.com/avcenter/venc/data/javaapp.beanhive.html

[Nachenberg 98] Carey Nachenberg, Eric Chien, Stephen Trilling, JavaApp.Strange Brew, http://www.symantec.com/avcenter/venc/data/javaapp.strangebrew.html

[RFC 1883] S. Deering, Xerox PARC, Internet Protocol, Version 6 (IPv6) Specification, December 1995

[RFC 1884] S. Deering, Xerox PARC, IP Version 6 Addressing Architecture, December 1995

[RFC 791] Jon Postel, Internet Protocol - DARPA Internet Program Protocol Spezification, September 1981

[RFC 792] Jon Postel, Internet Control Message Protocol, September 1981

[RFC 793] Jon Postel, Transmission Control Protocol - DARPA Internet Program Protocol Spezification, September 1981

[RFC1034] P. Mockapetris, Domain Names – Concepts and Facilities, November 1987

[RFC1035] P. Mockapetris, Domain Names – Implementation and Specification, November 1987

[RFC768] Jon Postel, User Datagram Protocol, August 1980

[SecurityFocus bid 1121] MS IE 5.01 JSObject Cross-Frame Vulnerability, http://www.securityfocus.com/bid/1121

[SecurityFocus bid 1209] Microsoft Internet Explorer for Macintosh java.net.URLConnection Vulnerability, http://www.securityfocus.com/bid/1209

[SecurityFocus bid 1339] Microsoft Internet Explorer for Macintosh getImage and classloader Vulnerabilities, http://www.securityfocus.com/bid/1339

[SecurityFocus bid 957] ,Microsoft Java Virtual Machine getSystemResource Vulnerability, http://www.securityfocus.com/bid/957

[SecurityFocus bid1336] Multiple Vendors java.net.URLConnection Applet Direct Connection Vulnerability, http://www.securityfocus.com/bid/1336

[SecurityFocus bid1337] Multiple Vendors HTTP Redirect Java Applet Vulnerability, http://www.securityfocus.com/bid/1337

[SecurityFocus bid1545] Multiple Vendor Java Virtual Machine Listening Socket Vulnerability, http://www.securityfocus.com/bid/1545

[SecurityFocus bid1546] Netscape Communicator URL Read Vulnerability, http://www.securityfocus.com/bid/1546

[SecurityFocus bid1754] Microsoft Virtual Machine com.ms.activeX.ActiveXComponent Arbitrary Program Execution Vulnerability, http://www.securityfocus.com/bid/1754

[SecurityFocus bid1812] Microsoft Virtual Machine Arbitrary Java Codebase Execution Vulnerability, http://www.securityfocus.com/bid/1812

[SecurityFocus bid2051] JRE Disallowed Class Loading Vulnerability, http://www.securityfocus.com/bid/2051

[Sun Java History] JAVA(TM) TECHNOLOGY: THE EARLY YEARS, http://java.sun.com/features/1998/05/birthday.html

[Sun SecCron 00] Security FAQ - Java(TM) Software Technology, http://java.sun.com/sfaq/chronology.html

Bugtraq archives for 4th quarter (Oct-Dec) 1997: Another way to exploit local classes in Java, http://www.dataguard.no/bugtraq/1997_4/0055.html

Check Point Software Technologies, http://www.checkpoint.com/

Computer Security Information, http://www.alw.nih.gov/Security/

Counterpane Internet Security, Inc., http://www.counterpane.com/

Covalent Technologies, http://www.covalent.net/raven/ssl/index.php

Cracking DES: Secrets of Encryption Research, Wiretap Politics, and Chip Design, http://cryptome.org/cracking-des.htm

David Hopwood, http://www.users.zetnet.co.uk/hopwood/,

DFN-CERT GmbH, DFN-PCA und DFN-FWL, http://www.cert.dfn.de/

Distributed Programming with Java, http://www.manning.com/Mahmoud/index.html

EFF DES Cracker Project , http://www.eff.org/descracker.html
Electronics Engineers, Inc.

FreeS/WAN Project: Home Page, http://www.xs4all.nl/~freeswan/

Higher Learning : For all the real hackers & phreakers, http://www.xs4all.nl/~l0rd/

IBM alphaWorks, http://www.alphaworks.ibm.com/Home/

IBM Developer Connection, http://www.developer.ibm.com/devcon/titlepg.htm

IBM developerWorks : Java technology, http://www.ibm.com/developer/java/

IBM Software : Application Development : VisualAge Developer Domain, http://www7.software.ibm.com/vad.nsf/

IETF Home Page, http://www.ietf.org/

Inet CERT® Coordination Center, http://www.cert.org/

iPlanet Developer, http://developer.iplanet.com/index.html

Java / MSIE / Netscape Cache Exploit - Jan '97, BobbyRite , http://www.alcrypto.co.uk/java/

Java Security, whitepaper by J. Steven Fritzinger, Marianne Mueller http://java.sun.com/security/whitepaper.ps

Java(TM) 2 SDK Documentation, http://java.sun.com/products/jdk/1.2/docs/index.html

Java(TM) Security API, http://java.sun.com/security/

jGuru.com(Home): Your view of the Java universe, http://www.jguru.com/portal/index.jsp?tab=1

Kimera: A Java System Architecture, http://kimera.cs.washington.edu/

Matt Blaze's cryptography resource on the web, http://www.crypto.com/

Poison Java, Richard Comerford, http://csl.iit.edu/~java/papers_others/java_security/jav.html, The Institute of Electrical and

Rob van der Woude's Scripting Pages: Batch Files for DOS, Windows (all flavours) and OS/2; Rexx; JavaScript; HTML, http://home.wanadoo.nl/r.woude/robmain.html

RSA Security Inc., http://www.rsasecurity.com/

Secure Internet Programming Laboratory, http://www.cs.princeton.edu/sip/

SUN Frequently Asked Questions - Applet Security, http://java.sun.com/sfaq/

SUN Java Developer Connection, http://developer.java.sun.com/

SUN Java Security Architecture: Contents, http://java.sun.com/products/jdk/1.2/docs/guide/security/spec/security-spec.doc.html

SUN Java(TM) Security API, http://java.sun.com/security/

SUN JDK 1.1.x Documentation, http://java.sun.com/products/jdk/1.1/docs/index.html

SUN The Source for Java(TM) Technology, http://java.sun.com/

W3C - The World Wide Web Consortium, http://www.w3.org/