Diplomarbeit

Verifikation von Firewall-Regeln mit dem Java Application Building Center

Stephan Windmüller

29. Juli 2008

Institution

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl 5 für Programmiersysteme

Gutachter

Prof. Dr. Bernhard Steffen
Prof. Dr. Tiziana Margaria-Steffen

Hiermit erkläre ich, Stephan Windmüller, dass ich die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Zitate habe ich stets kenntlich gemacht.

Dortmund, den 29. Juli 2008

_____

Stephan Windmüller

Inhaltsverzeichnis

Abkürzungsverzeichnis

DHCP Dynamic Host Configuration Protocol

DNS Domain Name System

DOM Document Object Model

DoS Denial of Service

DTD Document Type Definition

GPL GNU General Public License

HTTP Hypertext Transfer Protocol

ICMP Internet Control Message Protocol

IP Internet Protocol

jABC Java Application Building Center

NAT Network Address Translation

LSRR Loose Source and Record Route

SSRR Strict Source and Record Route

SAX Simple API for XML

SIB Service Independent Building Block

SMTP Simple Mail Transfer Protocol

SNMP Simple Network Management Protocol

SSH Secure Shell

TCP Transmission Control Protocol

UDP User Datagram Protocol

XML Extensible Markup Language

1 Einleitung

In einer perfekten Welt wären viele der heute alltäglichen Schutzmaßnahmen überflüssig. Ohne den Neid auf das Eigentum anderer ließe sich weder mit Schlüsselnoch mit Sicherheitsdiensten Geld verdienen. Ähnlich sähe es in der digitalen Welt aus: Die Wissenschaft der Kryptographie hätte ihren einzigen Zweck im Schutz der Privatsphäre und der Zugang zu Informationen und Rechnern jeder Art stünde allen Personen frei.

Allerdings sieht die Realität anders aus. Firmen, die das Schutzbedürfnis der Bürger und Firmen unterstützen, werden für ihre Dienstleistungen weiterhin entsprechend entlohnt. Vor allem mit Blick auf das Internet, in dem sich mit wenig Aufwand großer finanzieller Schaden anrichten lässt, sollten Firmen mit äußerster Sorgfalt auf die Absicherung ihrer eigenen Infrastruktur achten. Eine Komponente dieser Schutzvorrichtungen ist die von der breiten Bevölkerung oft missverstandene *Firewall*. Es handelt sich hierbei um ein Sicherheitskonzept, das auf verschiedene Arten umgesetzt werden kann.

In dieser Diplomarbeit wird zum einen der Weg vom Konzept zur funktionierenden Sicherheitslösung mit Hilfe eines grafischen Editors behandelt. Darauf aufbauend wurde zum anderen eine Softwarelösung entwickelt, mit welcher die Umsetzung dieses Konzepts darauf geprüft werden kann, ob die ursprünglichen Anforderungen eingehalten wurden.

1.1 Aufbau

Das erste Kapitel nach dieser Einleitung besteht aus einer Analyse der vorgegebenen Aufgabenstellung. Hierzu gehört neben einer kurzen Erklärung des Prinzips einer *Firewall* auch die Definition der erwünschten *Verifikation*.

Nach einer kurzen Einführung in den Aufbau von Datenpaketen im Internet wird im Abschnitt 3.2 der Begriff einer Firewall näher erläutert und deren Erstellung beschrieben. *Paketfilter* stellen einen elementaren Bestandteil dar und werden in Abschnitt 3.3 intensiver behandelt. Dazu werden vier verschiedene Paketfilter vorgestellt und ihre Eigenschaften verglichen.

Zur Umsetzung der in dieser Arbeit gesetzten Ziele wurden zwei Softwareprodukte – der Firewall Builder sowie das Java Application Building Center (jABC) – verwendet, auf die in Kapitel 5 eingegangen wird. Ihre Rolle im Gesamtkonzept wird dann zu Beginn des Kapitels 6 deutlich, das den Prozess der Implementierung erläutert.

Einen Einblick in die vorgesehene Arbeitsweise des fertigen Produktes liefert anschließend Kapitel 7. Anhand eines praxisnahen Beispiels wird hier der Aufbau eines

Paketfilters sowie dessen Überprüfung durchgeführt.

Die Diplomarbeit schließt mit einem Ausblick auf zukünftig mögliche Erweiterungen der entstandenen Software sowie einer Zusammenfassung der bisher durchgeführten Arbeiten.

2 Analyse der Aufgabenstellung

Um die im Rahmen dieser Diplomarbeit zu erfüllenden Aufgaben zu bestimmen, werden in diesem Kapitel zuvor der Begriff einer *Firewall* als auch das zu verwendende Konzept der *Verifikation* genauer definiert. Anschließend folgen eine Erläuterung der Rolle des jABC sowie die Erstellung des resultierenden Anforderungskatalogs.

2.1 Vom Konzept zur implementierten Firewall

Für die Absicherung eines Computernetzwerks ist im ersten Schritt die Erarbeitung eines Konzepts notwendig, das meist aus mehreren Anforderungen besteht. So könnten die beiden Anforderungen für das Netzwerk einer Firma etwa lauten, dass der eigene Webserver, auf dem die Firmenwebseite untergebracht ist, permanent weltweit erreichbar sein muss und die Mitarbeiter von ihrem Arbeitsplatz aus Zugriff auf externe Dienste im Internet haben. Dieses Szenario würde dem Konzept einer *Whitelist* entsprechen, bei der jeder nicht explizit zugelassene Datenverkehr das zu schützende Netzwek nicht erreichen beziehungsweise dieses nicht verlassen darf. Eine als Firewall bezeichnete und in Kapitel 3 näher beschriebene Komponente des Computernetzwerks sorgt für die Einhaltung dieser Anforderungen.

Während der Umsetzung eines solchen Konzepts werden die einzelnen (abstrakten) Anforderungen analysiert und in (konkrete) *Regeln*, die in der Konfigurationssyntax des jeweiligen Produktes verfasst sind, umgewandelt. Ein vollständiger Satz solcher Regeln kann anschließend benutzt werden, um die Firewall einzurichten.

Dieser komplexe Prozess der Umsetzung ist jedoch sehr fehleranfällig und kann zu unvorhergesehenen Ergebnissen führen. Zwar existieren Programme wie der im Kapitel 5 vorgestellte Firewall Builder, welche den Anwender bei diesem Vorgang unterstützen und ihm eventuell die Einarbeitung in die spezielle Syntax eines Produktes ersparen. So erlauben sie etwa die grafische Darstellung des zugrundeliegenden Netzwerks oder bieten fertige Konfigurationen für typische Einsatzszenarien an. Eine Überprüfung des Ergebnisses oder gar eine vollautomatische Erstellung sind jedoch nicht ohne Weiteres möglich. Wünschenswert ist daher eine Lösung, die – möglichst automatisiert – nach erfolgter Umsetzung auf eventuell verletzte Anforderungen testet. An dieser Stelle ist zudem ein generischer Ansatz hilfreich, der mehrere Firewallprodukte unterstützt und möglichst unabhängig vom verwendeten Betriebssystem funktioniert.

2.2 Überprüfung durch Verifikation und Validierung

Je nach Methode der Überprüfung eines Systems schwankt die Aussagekraft ihrer Ergebnisse. Aus diesem Grund muss zwischen den Verfahren *Verifikation* und *Validierung* unterschieden werden.

Verifikation

Mit Hilfe einer *formalen Verifikation* kann in der Informatik auf der Basis eines mathematischen Beweises die korrekte Umsetzung eines Systems gezeigt werden. Sofern dieser Beweis fehlerfrei durchgeführt wurde, sind keine weiteren Tests notwendig. Der Nachteil dieser Methode ist die schnell wachsende Komplexität der Beweisführung bei umfangreichen und komplexen Systemen, weswegen hier meist nur Tests einzelner Funktionen zum Einsatz kommen.

Ein bekanntes Verfahren zur Verifikation ist das sogenannte *Model Checking*, auf dessen Möglichkeiten in Kapitel 4 näher eingegangen wird.

Validierung

Im Gegensatz zur Verifikation zielt die *Validierung* nicht auf eine Aussage über alle möglichen Eingaben für ein Problem ab sondern für konkrete Testfälle. Zu diesem Zweck werden Testdaten benutzt, die das zu überprüfende System möglichst umfassend kontrollieren. Da nicht alle möglichen Eingaben berücksichtigt werden können, ist eine sorgfältige Auswahl der Testfälle notwendig.

2.3 Verwendung des Java Application Building Center

Bei dem jABC handelt es sich um eine Software zur Modellierung von Systemabläufen, die am Lehrstuhl V der Technischen Universität Dortmund entwickelt wird. Durch ihre grafische Darstellung von Systembeschreibungen eignet sie sich, um Firewall-Regeln und deren Testabläufe zu visualisieren. Mittels eines Plugins soll das jABC daher um Funktionalitäten zum Einlesen und Simulieren von Regelsätzen erweitert werden. Da die Software in der platformunabhängigen Sprache *Java* geschrieben ist, entfällt für den Anwender die Beschränkung auf ein Betriebssystem.

2.4 Anforderungskatalog

Damit die Überprüfung eines umgesetzten Firewall-Konzepts mit dem jABC durchgeführt werden kann, muss dieses um entsprechende Funktionen erweitert werden. Dies geschieht durch ein im Rahmen dieser Arbeit entwickeltes Plugin, welches sowohl das Einlesen und die grafische Darstellung von Regelsätzen als auch deren Simulation erlaubt. Im Folgenden werden daher die einzelnen Anforderungen angeführt, welche das Plugin erfüllen muss.

Importierung von Regelsätzen
Im ersten Schritt sollen existierende Regelsätze in das jABC importiert werden können. Um an dieser Stelle unabhängig vom gewählten Firewall-Produkt arbeiten zu können, bietet sich der Import eines Formats an, das mehrere Implementationen unterstützt.

Grafische Darstellung der importierten Regeln
Nach dem abgeschlossenen Import sollen die Regelsätze innerhalb des jABC grafisch dargestellt werden. Hierzu gehören eine Visualisierung der wichtigsten Eigenschaften sowie die Zusammenhänge der Regeln untereinander.

Erzeugung von virtuellen Paketen
Zur Simulation des Datenverkehrs soll dieser aus vorhandenen Dateien virtuell erstellt werden. Bei diesen Dateien kann es sich um verschiedene Quellen handeln, beispielsweise eine entsprechende XML-Struktur.

Simulation der Regelsätze mit Hilfe der virtuellen Pakete
Die eigentliche Validierung der erstellten Regelsätze geschieht in diesem Teil. Mit Hilfe des virtuellen Datenverkehrs soll der zuvor erstellte Graph durchlaufen und die Ergebnisse im jABC dargestellt werden.

Verifikation der Ergebnisse
Anhand der erzeugten Ergebnisse soll es möglich sein, eventuelle Fehler in der Umsetzung der Firewall festzustellen. Existieren keine solchen Fehler, gilt die Firewall als verifiziert.

3 Sicherheitskonzepte in Netzwerken

Zur Absicherung eines Netzwerks gegen Angriffe von innen oder außen ist ein grundlegendes Verständnis der verwendeten Technologien unerlässlich. Auf Basis der eigenen Anforderungen kann dann ein Konzept erstellt werden, das von einer Firewall umgesetzt wird. Dieses Kapitel erläutert die Grundlagen des Datenverkehrs in öffentlichen sowie privaten Netzen und zeigt exemplarisch die Funktionsweise einer Firewall. Im Anschluss daran werden verschiedene Implementierungen vorgestellt und ihr Funktionsumfang verglichen.

3.1 Datenverbindungen in Netzwerken

Möchten Kommunikationspartner über ein Computernetzwerk Daten austauschen, so geschieht dies mit Hilfe von *Datenpaketen*. Diese enthalten neben den eigentlichen Nutzdaten auch Informationen über Sender und Empfänger oder die Route, welche das Paket auf seinem Weg zurückgelegt hat. Große Datenströme werden dabei vor dem Versand auf mehrere kleine Pakete aufgeteilt und beim Empfänger wieder zusammengesetzt.

Sicherheitsprobleme ergeben sich bei diesem Konzept dadurch, dass Absender den Inhalt des Pakets frei bestimmen können und somit zum Beispiel in der Lage sind, die wahre Herkunft zu verschleiern. Diese Art von Angriff ist eines der Gebiete, auf welchen der Einsatz einer Firewall hilfreich ist.

3.1.1 Pakettypen

Zur Kommunikation im Internet wird derzeit am häufigsten das durch den Begriff *IP-Adresse* bekannte Internet Protocol (IP) verwendet. Ein solches Paket beginnt mit einem Kopfteil (Header), in dem diverse Optionen festgelegt sind. Dazu gehören neben den Adressen von Absender und Empfänger auch Angaben über Länge und Typ des folgenden Datenteils [Postel, 1981b]. Routing-Informationen beeinflussen zusätzlich das Verhalten von Routern, die an der Übertragung des Paketes beteiligt sind.

Während jedoch IP ausschließlich die Vermittlung übernimmt, sorgen darauf aufbauende Protokolle für den Datentransport. Die beiden wichtigsten sind das Transmission Control Protocol (TCP) für verbindungsorientierte sowie das User Datagram Protocol (UDP) für verbindungslose Dienste. Sie werden im Datenteil eines IP-Paketes übertragen und bestehen ihrerseits wiederum aus einem Header sowie einem Datenteil (siehe Abbildung 3.1). Zuätzlich zu den im IP-Header angegebenen

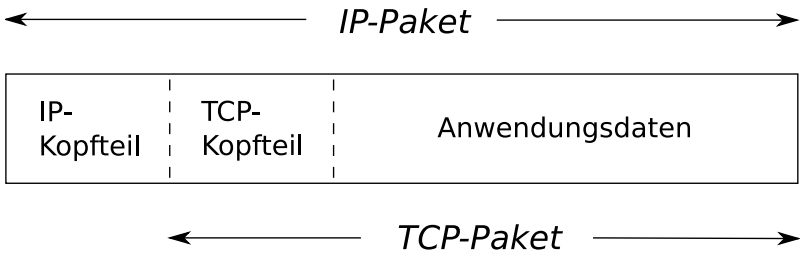


Abbildung 3.1: Aufbau eines TCP-Paketes, das innerhalb eines IP-Paketes transportiert wird

Adressen definieren beide Protokolle die Nummer eines Quell- und Zielports. Diese werden benötigt, da auf einem kommunizierenden Rechner mehrere Dienste laufen können, die eindeutig erreichbar sein müssen. Ein Dienst erhält daher einen oder mehrere Ports, die je nach Anwendung mit standardisierten Nummern belegt sind. Die Kombination aus IP-Adresse und und Port wird auch als *Socket* bezeichnet. Auf diese Weise sind mehrere verschiedene Verbindungen zwischen zwei Rechnern möglich.

Hilfreich zum Informationsaustausch beim Auftreten von Fehlern oder zur Diagnose ist weiterhin das Internet Control Message Protocol (ICMP). Innerhalb eines IP-Paketes versendet kann es festgelegte Statusmeldungen wie „Zielrechner nicht erreichbar“ übertragen [Postel, 1981a].

3.1.2 Anwendungsprotokolle

Der Aufbau des Datenteils eines TCP- oder UDP-Paketes richtet sich nach den Vorgaben des jeweiligen Anwendungsprotokolls. Sollen sich die Sicherheitsmaßnahmen auch auf die übertragenen Daten beziehen, so muss der Aufbau dieser Protokolle bekannt sein. Die wohl bekanntesten Beispiele sind das Hypertext Transfer Protocol (HTTP) zur Übertragung von Webseiten sowie das Simple Mail Transfer Protocol (SMTP) zum Versenden von E-Mails. Eine Sonderrolle nehmen verschlüsselte Protokolle ein, bei denen eine inhaltliche Analyse meist nicht möglich ist.

3.2 Firewalls

Im Verständnis vieler Computerbenutzer beschreibt der Begriff *Firewall* die sprichwörtliche „Feuerwand“, welche ungebetene Gäste wie zum Beispiel Hacker vom eigenen System fernhält. Zu diesem Bild haben nicht unwesentlich die Hersteller von Sicherheitslösungen für Privatanwender beigetragen, die ihre Produkte mit eben diesem Versprechen bewerben.

Tatsächlich besteht eine Firewall jedoch nicht aus einer Software, die nach der Installation einen lokalen Rechner sichert. Vielmehr handelt es sich um ein generelles Konzept zur Kontrolle des Datenverkehrs zwischen Netzwerken mit unterschiedli-

chen Sicherheitsstufen, welches durch eine Kombination aus Hard- und Softwarekomponenten umgesetzt wird.

Meist ist eines dieser Netze das öffentliche Internet, auf dessen Sicherheit praktisch niemand direkten Einfluss hat. Daher besteht die Hauptaufgabe vieler Firewall-Implementierungen darin, das eigene Netz vor dem Internet abzusichern und nur gewünschten Datenverkehr zuzulassen.

3.2.1 Aufgaben von Firewalls

Neben der Abwehr von Attacken aus externen Netzen kann eine Firewall noch weitere Funktionen erfüllen. Die Themen dieser Diplomarbeit konzentrieren sich dabei auf die Kontrolle des Netzwerkverkehrs und die Erkennung von gefälschten Paketen (sogenannte *Spoofing-Attacken*).

Kontrolle des Netzwerkverkehrs

Die primäre Aufgabe von Firewalls ist meist die Kontrolle des Netzwerkverkehrs durch sogenannte *Paketfilterung*. Dabei wird jedes an der Firewall ankommende oder von dort abgesendete Paket analysiert und seine Eigenschaften mit einem internen Regelwerk verglichen. Dieser Teil einer Firewall wird als *Paketfilter* bezeichnet.

Durch die Kontrolle ist es möglich, bestimmte Verbindungen gezielt zuzulassen, während jeglicher andere Datenverkehr blockiert wird. Dieser Ansatz kann als *Whitelist* bezeichnet werden und erfordert nur dann Änderungen an der Konfiguration, wenn neue Verbindungen hinzugefügt werden müssen. Andererseits ist es ebenso möglich, alle Verbindungen zu erlauben und nur einzelne, unerwünschte Pakete gezielt zu blockieren. Dies entspräche dem Prinzip einer *Blacklist*. Hier besteht allerdings die Gefahr, dass nicht alle Angriffsarten bedacht wurden. In den meisten Fällen wird daher der erste Ansatz gewählt.

Abwehr von Spoofing-Attacken

Paketfiler sortieren nach den Eigenschaften, die ein Datenpaket in sich trägt. Es ist jedoch problemlos möglich, Daten wie beispielsweise die Herkunft eines Pakets gezielt zu fälschen, um so den Paketfilter zu umgehen. Diese Art von Angriff wird als *Spoofing* bezeichnet. Eine Firewall sollte dieses erkennen und blockieren können, was auch als *Ingress Filtering* [Ferguson u. Senie, 2000] bezeichnet wird.

Eine häufige Art der Umsetzung eines solchen Schutzes besteht in der Überprüfung, ob ein Datenpaket zu einem Absender gehören soll, der über die empfangende Netzwerkschnittstelle gar keine Daten senden kann. So ist es beispielsweise nicht möglich, dass ein Paket aus dem internen Netzwerk über die Schnittstelle empfangen wird, welche nur mit dem öffentlichen Internet verbunden ist.

Network Address Translation

Aufgrund des immer knapper werdenden Raums an IPv4-Netzwerkadressen ist es heute nicht möglich, jeden Rechner mit einer festen Adresse zu versehen, über die er im Internet erreichbar wäre. Stattdessen erhalten viele Zugänge nur eine einzelne IP-Adresse, über die das gesamte angebundene Netzwerk mit dem Internet verbunden

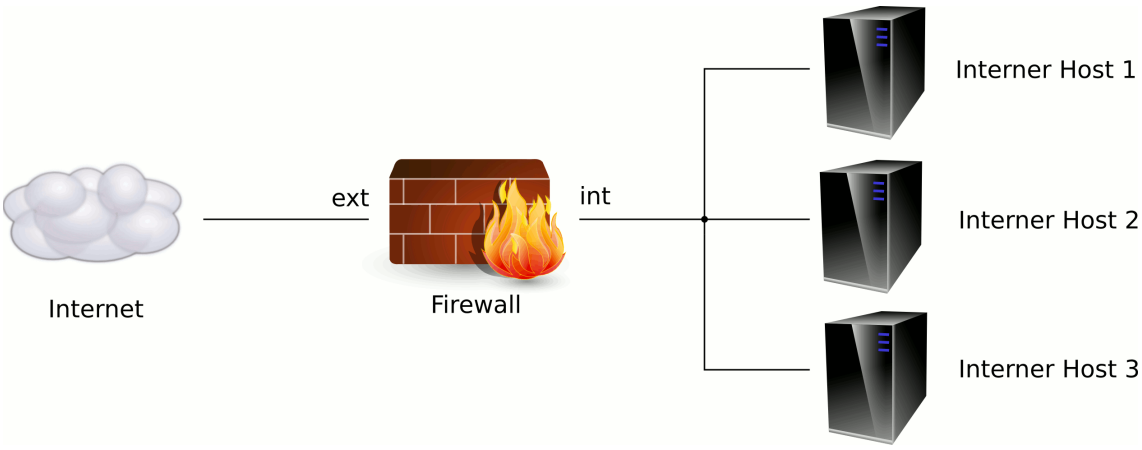


Abbildung 3.2: Schema einer exemplarischen Firewall

wird. Die Technik, mit der dies realisiert wird, trägt den Namen Network Address Translation (NAT).

Mit diesem Verfahren werden die Quellports von Paketen, die ins Internet geleitet werden, von der Firewall vor dem Versand umgeschrieben. Zusätzlich wird diese Zugehörigkeit in einer sogenannten NAT-Tabelle vermerkt. Antworten auf diese Pakete aus dem Internet werden beim Eintreffen dann wieder korrekt umgeschrieben und an den entsprechenden Rechner weitergeleitet.

Zusätzlich ist es auch möglich, feste Regeln zu definieren, was zum Beispiel den Betrieb eines Webservers hinter der Firewall erlaubt.

Filterung unerwünschter Inhalte

Viele Unternehmen gewähren ihren Mitarbeitern keinen unbeschränkten Internetzugang, da sie durch privates Surfen Leistungseinbußen befürchten oder aber gewisse Dienste als ungeeignet oder gar gefährlich betrachten. Auch diese Art der Filterung ist Teil einer Firewall, arbeitet jedoch im Gegensatz zum Paketfilter auch auf den Inhalten der Pakete. Eine mögliche Umsetzung könnte ein sogenannter *Proxy* darstellen. Dabei handelt es sich wie bei einem Paketfilter um einen Rechner, über den jeglicher Datenverkehr abgewickelt wird. Allerdings werden bei der Überprüfung nicht nur die Informationen der Header sondern auch der Datenteil der Pakete zur Entscheidungsfindung verwendet, etwa der Inhalt übertragener Webseiten.

Im privaten Bereich kann diese Technik benutzt werden, um Minderjährigen den Zugang zu Seiten zu verwehren, die für ihr Alter ungeeignet erscheinen.

3.2.2 Beispiel

Abbildung 3.2 zeigt die schematische Darstellung einer exemplarischen Firewall. Auf der linken Seite ist das öffentliche Internet mit seinen verfügbaren Diensten dargestellt. Die drei Rechner, welche rechts zu sehen sind, gehören zum internen Netzwerk und sollen auf eben diese Dienste zugreifen können. Geschützt werden sie

durch einen Firewall-Rechner in der Mitte, der durch seine zwei Netzwerkschnittstellen *int* und *ext* mit jeweils einem der beiden Netze verbunden ist. Eine direkte Verbindung besteht nicht, somit muss jeder Datenverkehr die Firewall passieren.

Ein rudimentäres Regelwerk, welches für jedes Paket von oben nach unten durchlaufen wird, könnte folgendermaßen aussehen:

1. Spoofing-Attacken von außen sollen unterbunden werden.
2. Rechner im internen Netz dürfen Verbindungen in das Internet aufbauen.
3. Der Firewall-Rechner kann vom internen Netz aus konfiguriert werden.
4. Alle anderen Verbindungen sind nicht zugelassen.

Die primäre Aufgabe dieser Beispiel-Firewall ist somit, einen sicheren Zugang der internen Rechner zum Internet zu gewährleisten. Ein Verbindungsaufbau soll nur durch die internen Rechner möglich sein, Pakete aus dem Internet werden nur als Antworten auf bereits bestehende Verbindungen zugelassen.

Zusätzlich soll es möglich sein, den Firewall-Rechner von einem internen Rechner aus zu administrieren. Zu diesem Zweck soll das SSH-Protokoll verwendet werden. Es handelt sich dabei um einen Dienst, der primär eine entfernte, verschlüsselte Terminalsitzung zu einem anderen Rechner bereitstellt [Ylonen u. Lonvick, 2006].

Um zu verhindern, dass sich Pakete aus dem Internet mit internen Adressen tarnen, sollen zusätzlich im ersten Schritt jegliche Spoofing-Versuche unterbunden werden.

3.3 Paketfilter

Wie bereits erwähnt ist die Filterung von Paketen primäre Aufgabe vieler Firewalls. Sie geschieht meist auf einem separaten Rechner, welcher zwischen den zu trennenden Netzen steht und wird durch eine Software – den Paketfilter – gesteuert. Mit einem entsprechenden Regelsatz ließe sich so unter anderem das Beispielkonzept aus dem Abschnitt 3.2.2 auf der vorherigen Seite ohne weitere Komponenten umsetzen.

Die im Abschnitt zuvor genannten Sicherheitslösungen für Heimanwender werden im Gegensatz dazu jedoch auf dem zu schützenden System selbst ausgeführt, weswegen sich hier die Bezeichnung *Hostbasierter Paketfilter* besser eignet.

3.3.1 Aufgaben

Zu den Aufgaben eines Paketfilters gehören neben der eigentlichen Filterung auch die (teilweise) Aufzeichnung des Datenverkehrs sowie eine eventuelle Veränderung der Pakete.

Paketfilterung

Bei der Paketfilterung wird jedes einzelne Paket betrachtet und anhand eines vorher definierten Regelwerks entschieden, was damit geschehen soll. Diese Entscheidung

Quelle	Ziel	Interface	Anwendung	Aktion
Internes Netz	Alle	ext	Alle	**Verwerfen**
Internes Netz	Externes Netz	int	Alle	**Erlauben**
Internes Netz	Firewall	int	SSH	**Erlauben**
Alle	Alle	Alle	Alle	**Verwerfen**

Tabelle 3.1: Mögliches Regelwerk zur Umsetzung des Beispiel-Konzepts mit einem Paketfilter

kann sowohl isoliert als auch im Kontext der bereits bestehenden Verbindungen getroffen werden.

Zu den überprüften Eigenschaften eines Pakets gehören neben Herkunft, Ziel und Typ eines Pakets auch Portnummern oder spezielle Flags wie das TCP-SYN. Weitere Details dazu finden sich in Unterabschnitt 3.3.4 auf Seite 18.

Logging

Mit Hilfe von Log-Dateien ist es möglich, einzelne oder auch alle Pakete im Nachhinein zu analysieren und so etwa Angriffe aufzudecken. Hilfreich sind diese Aufzeichnungen auch während der Entwicklung eines neuen Regelsatzes, um falsch behandelte Pakete aufzuspüren. Da jedoch der Datenteil eines Pakets zur Auswertung meist zweitrangig ist, werden nur die Header-Informationen protokolliert.

Veränderung von Paketen

Paketfilter können benutzt werden, um einzelne Pakete während der Verarbeitung zu verändern. So ist es möglich, Pakete aufgrund ihrer Eigenschaft für eine schnellere Verarbeitung zu markieren. Dies wird auch als *Quality of Service* bezeichnet.

Network Address Translation

Das in Abschnitt 3.2 auf Seite 8 beschriebene Verfahren der Network Address Translation wird von einigen Paketfilter-Implementationen direkt unterstützt.

3.3.2 Funktionsweise

Im einfachsten Fall erhält der Paketfilter für seine Entscheidungen ein Regelwerk, welches er schrittweise abarbeitet und bei passenden Regeln die entsprechende Aktion ausführt. So könnte ein Regelwerk für das Beispielkonzept aus dem Unterabschnitt 3.2.2 auf Seite 10 wie die Tabelle 3.1 aussehen.

Die erste zutreffende Regel entscheidet hierbei darüber, ob ein Paket verworfen wird oder passieren darf, daher ist die Reihenfolge von Bedeutung. Durch die letzte Regel werden alle Pakete abgefangen, welche keiner früheren zugeordnet werden können. Dies entspricht einer Umsetzung der Forderung des Konzepts, dass jeglicher Datenverkehr, der nicht explizit erlaubt wurde, unterbunden werden soll. Abbildung 3.3 zeigt den entsprechenden Ablauf innerhalb des Paketfilters in Form eines Graphen.

Es ist jedoch zu beachten, dass die im obigen Regelwerk spezifizierten Regeln bloß

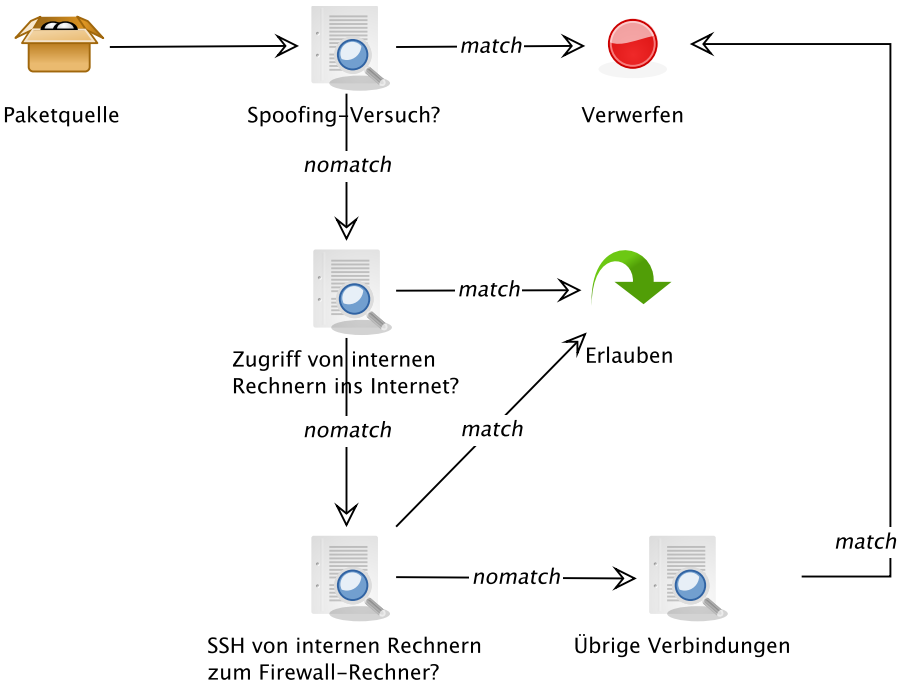


Abbildung 3.3: Darstellung des exemplarischen Ablaufs als Graph

eine Abstraktion darstellen. Würde man sie exakt so in einen Paketfilter übertragen, so ist es durchaus möglich, dass das Ergebnis nicht dem Gewünschten entspricht. Beispielsweise ist nicht festgelegt, wie mit Antworten auf die Anfragen der internen Rechner umzugehen ist. Eine Kommunikation mit Diensten im Internet ist jedoch kaum möglich, wenn die Gegenstelle nicht in der Lage ist, Antworten zurückzusenden.

Regelwerke müssen dabei nicht immer aus einer einzelnen Liste bestehen, die linear abgearbeitet wird. Viele Paketfilter können separate Listen verwalten, die bei der Abarbeitung ein mögliches Ziel anstatt einer Aktion darstellen. Sollen im obigen Beispiel alle verworfenen Pakete in einer Logdatei gespeichert werden, könnte eine entsprechende Regel erstellt werden, die alle behandelten Pakete unabhängig von sonstigen Parametern erst protokolliert und dann verwirft. Diese müsste dann nur noch in der ersten und vierten Regel statt der Aktion „Verwerfen“ als Ziel angegeben werden.

3.3.3 Implementierungen

Der Markt bietet eine große Anzahl von Paketfiltern, wobei sowohl kommerzielle Produkte als auch frei verfügbare Entwicklungen vorhanden sind. Im Folgenden wird eine Auswahl von vier bekannten Open-Source-Produkten beschrieben: Netfilter, IPFilter, ipfirewall und pf.

3.3.3.1 Netfilter

Seit Version 2.4 des Linux-Kernels ist Netfilter [Welte u. Ayuso, 2008] Bestandteil desselben und löste damit das vorher verwendete *ipchains* ab. Netfilter ist jedoch besser bekannt unter dem Namen seines Kommandozeilenwerkzeugs *iptables*, mit dem das Regelwerk angelegt und verwaltet wird. Die modulare Aufteilung von Netfilter innerhalb des Linux-Kernels erlaubt eine gezielte Aktivierung benötigter Features.

Netfilter verfügt über vier interne Tabellen, in die Regelketten eingefügt werden können. Diese haben unterschiedliche Aufgaben.

filter In dieser Tabelle geschieht die standardmäßige Filterung von eingehenden, ausgehenden und weitergeleiteten Paketen.

nat Regelketten für die Network Address Translation finden hier ihren Platz.

mangle Eine Veränderung von verarbeiteten Paketen findet in dieser Tabelle statt.

raw Diese Tabelle wird für ein erweitertes Connection Tracking benutzt.

Regelketten wiederum bestehen aus einer Liste von einzelnen Regeln, die nacheinander abgearbeitet werden. Fünf solcher Ketten sind fest in Netfilter integriert, eigene können angelegt werden. Welche Ketten ein Paket durchläuft, wird im ersten Schritt durch dessen Quelle und Ziel bestimmt.

INPUT Eingehende Pakete, welche an Dienste gerichtet sind, die auf dem Rechner des Paketfilters selbst laufen, werden durch diese Regel verarbeitet.

OUTPUT Ausgehende Pakete, die von einem lokalen Dienst erzeugt wurden

FORWARD Pakete, welche durch die Firewall mittels Routing weitergeleitet werden

PREROUTING Diese Kette wird für alle weiterzuleitenden Pakete abgearbeitet, bevor die Entscheidung zum Routing getroffen wird.

POSTROUTING Weiterzuleitende Pakete durchlaufen diese Kette unmittelbar bevor sie den Rechner verlassen.

Jede Regel innerhalb einer Regelkette erzeugt eine Aktion, falls die Regel auf das Paket zutrifft. Das Ziel einer solchen Aktion kann unter anderem eine selbstdefinierte Regelkette sein oder aber eine der vier fest spezifizierten Aktionen darstellen:

ACCEPT Akzeptiert das Paket

DROP Verwirft das Paket ohne Rückmeldung an den Absender

QUEUE Übergibt das Paket an einen lokalen Prozess

RETURN Springt zurück in die übergeordnete Regelkette

Zusätzlich stehen noch spezielle erweiterte Aktionen wie LOG (Logging von Paketen), DNAT (Network Address Translation), MARK oder REJECT zur Verfügung.

Beispiel

Im Folgenden werden die notwendigen iptables-Befehle zum Aufbau eines Paketfilters beschrieben, der die Anforderungen aus Unterabschnitt 3.2.2 auf Seite 10 erfüllt. Dabei erhalten die Rechner des internen Netzwerks in diesem Beispiel eine Adresse aus dem Raum 129.217.1.*, der Firewallrechner selbst die 129.217.1.1. Durch die nachfolgenden Kommandos wird für jede der drei Regelketten eine Default-Policy gesetzt, die in Kraft tritt, wenn keine andere Regel zutrifft.

```
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP
```

Im nächsten Schritt sollen alle Pakete zugelassen werden, die bereits existierenden Verbindungen zugeordnet werden können.

```
iptables -A INPUT   -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A OUTPUT  -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Neue Verbindungen sollen sowohl für Secure Shell (SSH) aus dem internen Netz zur Firewall als auch alle anderen Protokolle ins Internet möglich sein.

```
iptables -A INPUT -i in -p tcp -m tcp -s 129.217.1.0/24
     --dport 22 -m state --state NEW -j ACCEPT

iptables -A FORWARD -s 129.217.1.0/24 -m state --state NEW -j ACCEPT
```

Da sich die FORWARD-Kette nicht auf lokale Prozesse wie SSH bezieht, ist ein explizites Verbot aller anderen Verbindungen vom internen Netz zur Firewall nicht notwendig.

3.3.3.2 IPFilter

Im Gegensatz zu Netfilter ist IPFilter nicht fest an ein bestimmtes Betriebssystem gekoppelt und daher portabel einsetzbar. Die Liste der unterstützten Betriebssysteme[1] umfasst unter anderem Solaris, Linux und diverse BSD-Varianten.

IPFilter wird über eine Konfigurationsdatei angepasst, welche bei Aktivierung des Paketfilters eingelesen wird. Ein Nachteil dieser Methode ist die fehlende Möglichkeit, dynamisch Regeln einzufügen. Dies kann jedoch hilfreich sein, wenn zum Beispiel einzelne Rechner kurzzeitig „ausgesperrt“ werden sollen.

[1] `http://coombs.anu.edu.au/~avalon/ip-filter.html`

Eine weitere Unterscheidung zu anderen Paketfiltern besteht in der Abarbeitung der Regeln. Während die Hälfte der hier vorgestellten Produkte bei einer zutreffenden Regel die Verarbeitung mit der entsprechenden Aktion abbrechen, durchläuft IPFilter normalerweise alle Regeln bis zum Ende der Konfigurationsdatei. Auf diesem Weg können mehrere Regeln auf ein Paket zutreffen – die zuletzt gewählte Aktion wird dann angewendet.

Beispiel
Während bei anderen Paketfiltern die Default-Policy am Ende der Regeln oder als separate Einstellung eingebunden werden muss, wird dies bei IPFilter zu Beginn der Regeln definiert. Alle Pakete, auf die keine weiteren Regeln zutreffen, werden demnach abgelehnt.

```
block in            from any            to any
block out           from any            to any
```

Sollen Pakete hingegen expliziet erlaubt werden, so geschieht dies durch das Schlüsselwort *pass*. Die folgenden Regeln definieren dazu Optionen für die Pakete wie Übertragungsprotokoll und Quelladdresse.

```
pass in  proto icmp from 129.217.1.0/24 to any keep state
pass in  proto tcp  from 129.217.1.0/24 to any keep state
pass in  proto udp  from 129.217.1.0/24 to any keep state
pass in             from 129.217.1.0/24 to any
pass out proto icmp from 129.217.1.0/24 to any keep state
pass out proto tcp  from 129.217.1.0/24 to any keep state
pass out proto udp  from 129.217.1.0/24 to any keep state
pass out            from 129.217.1.0/24 to any
```

Allerdings sollen nicht alle Verbindungen aus dem internen Netz erlaubt werden, weshalb diese mit der nächsten Zeile wieder blockiert werden.

```
block in on int     from 129.217.1.0/24 to 129.217.1.1
```

Der letzte Eintrag der Konfiguration trägt den erlaubten SSH-Verbindungen aus dem internen Netz Rechnung.

```
pass  in on int     proto tcp from 129.217.1.0/24
                              to 129.217.1.1 port = 22 keep state
```

Vorzeitiger Abbruch der Regelverarbeitung
Der Nachteil dieser Art der Abarbeitung liegt in der Performance des Systems – alle Pakete müssen alle Regeln einmal passieren, bevor eine Entscheidung getroffen werden kann. Um Regeln, die häufig angewendet werden, möglichst frühzeitig anzuwenden, existiert das Schlüsselwort *quick*. Bei diesem wird die entsprechende Regelaktion sofort ausgeführt, die folgenden Regeln bleiben unberücksichtigt.

Soll also jeglicher SSH-Datenverkehr von den internen Rechnern zur Firewall sofort erlaubt werden, muss die folgende Regel zu Beginn der Konfiguration eingefügt werden.

```
pass in quick on int proto tcp from 129.217.1.0/24
                                to 129.217.1.1 port = 22 keep state
```

3.3.3.3 ipfirewall

Der Paketfilter ipfirewall (in der Kurzform ipfw) wurde für das Betriebssystem FreeBSD entwickelt, funktioniert jedoch auch unter MacOS X und in Teilen unter Microsoft Windows. Dabei steht die Bezeichnung sowohl für den Teil im Kernel des Betriebssystems als auch für das Kommandozeilenprogramm ipfw, mit welchem einzelne Regeln hinzugefügt werden.

Ebenso wie sich IPFilter in der Semantik (first rule counts) von allen anderen hier vorgestellten Paketfiltern abhebt, trifft dies auf ipfirewall und dessen Syntax zu. Anstatt einer Liste, die von oben nach unten durchgearbeitet wird, besteht ein Regelsatz für diesen Filter aus nummerierten Regeln, zwischen denen „Sprünge“ möglich sind.

Beispiel
Zu Beginn der Regelverarbeitung wird in diesem Beispiel auf eine bereits vorhandene, zuvor erlaubte Verbindung geprüft. Gehört ein Paket zu solch einer Verbindung, wird es durchgelassen.

```
$IPFW add 1 check-state ip from any to any
```

Im nächsten Schritt werden ausschließlich SSH-Verbindungen zum Firewall-Rechner hin erlaubt. Dabei ist zu beachten, dass der Rechner, auf dem der Paketfilter installiert ist, mit *me* bezeichnet wird. Das Schlüsselwort *keep-state* erlaubt im Zusammenhang mit der zuvor angelegten Regel Nummer 1 alle zukünftigen Pakete dieser Verbindung.

```
$IPFW add 10 allow tcp from 129.217.1.0/24 to me  22
in via int setup keep-state

$IPFW add 20 deny  all from 129.217.1.0/24 to me  in via int
```

Mit den letzten beiden Regeln werden zum einen alle übrigen Verbindungen der internen Rechner erlaubt und zum anderen eine Default-Policy gesetzt, die den Rest des Datenverkehrs blockiert.

```
$IPFW add 30 allow all from 129.217.1.0/24 to any keep-state

$IPFW add 40 deny  all from any            to any
```

3.3.3.4 pf

Auf ihrer Homepage [OpenBSD project, 2008] werben die Entwickler des Betriebssystems OpenBSD mit der Aussage „Only two remote holes in the default install, in more than 10 years!“. Man kann davon ausgehen, dass die Entwicklung des eigenen Paketfilers pf nicht weniger strengen Kontrollen unterlag.

Ursprünglich enthielt OpenBSD den bereits erwähnten Filter IPFilter. Aus Lizenzgründen [Hansteen, 2008] wurde dieser im Mai 2001 entfernt und mit Erscheinen von OpenBSD 3.0 durch den von Daniel Hartmeier entwickelten pf ersetzt. Beide Filter ähneln sich jedoch stark in der verwendeten Syntax.

pf ist nicht auf OpenBSD beschränkt, sondern auch Bestandteil von FreeBSD, NetBSD sowie DragonFly.

Gegenüber den anderen Paketfiltern bietet pf einige Alleinstellungsmerkmale. So ist es zum Beispiel möglich, als Filterkriterium das verwendete Betriebssystem eines Rechners zu verwenden. Dies wird durch kleine Unterschiede in der Implementierung von TCP/IP-Stacks erreicht, die pf erkennen kann. Auf diese Weise kann eine Regel wie „Verbiete Verbindungen mit Rechnern auf Port 139, die Microsoft Windows einsetzen“ umgesetzt werden.

Zusätzlich bietet pf die Möglichkeit, eingehenden Datenverkehr zu normalisieren, also unter anderem fragmentierte Pakete noch vor der Filterung zusammenzusetzen.

3.3.4 Unterschiede im Funktionsumfang

Jeder der oben genannten Paketfilter beherrscht grundlegende Funktionen zur Paketfilterung wie zum Beispiel einen Test auf Quell- und Zieladresse oder aber das verwendete Protokoll. Darüber hinaus existieren diverse erweiterte Funktionen, die nicht von allen Paketfiltern oder zumindest nicht im vollen Umfang unterstützt werden. Da keines der Produkte alle Möglichkeiten implementiert, hängt die Auswahl des verwendeten Filters stark von den eigenen Anforderungen ab. Aus diesem Grund werden im Folgenden die einzelnen Merkmale vorgestellt und der Funktionsumfang der Paketfilter gegenübergestellt.

3.3.4.1 Stateful Filtering

Verbindungen über das TCP können einen von drei Zuständen annehmen. Zu Beginn startet der Client den *Verbindungsaufbau*. Nachdem dieser erfolgt ist, können Daten gesendet werden. Initiiert einer der beiden Seiten einen *Verbindungsabbau*, so signalisiert sie damit das Ende der Kommunukation.

Hängt die für ein TCP-Paket zu treffende Entscheidung nicht von dem vorherigen Datenverkehr und somit vom Zustand einer TCP-Verbindung ab, so spricht man von *Stateless Filtering*. Die Behandlung von Antwortpaketen muss dann separat geschehen, beispielsweise über die Betrachtung der gesetzten TCP-Flags.

Im Gegensatz hierzu merkt sich ein Paketfilter beim *Stateful Filtering* den Status einer Verbindung und behandelt die zugehörigen Pakete entsprechend. Auf diese Weise ist es möglich, Antwortpakete auf bestehende Verbindungen generell zu erlauben, ohne für jeden Dienst entsprechende Regeln konfigurieren zu müssen.

In diesem Punkt unterscheidet sich die Qualität der verfügbaren Paketfilter deutlich. Während einige der Produkte interne Daten über existierende Verbindungen führen, entscheiden andere nur aufgrund der gesetzten TCP-Flags.

Beim Einsatz dieser Technik sollte jedoch bedacht werden, dass Angreifer mit Hilfe von SYN-Floods, die im RFC 4987 [Eddy, 2007] beschrieben werden, einen Paketfilter außer Gefecht setzen können. Wird nämlich der Aufbau einer TCP-Verbindung (Drei-Wege-Handshake) nicht vollständig durchlaufen, so existiert dennoch bereits eine Information über die (mögliche) Verbindung in der internen Datenbank. Geschieht dies mit einer hohen Frequenz, so kann durch diese Denial of Service (DoS)-Attacke die Datenbank „überlaufen“ und neue Verbindungen somit unmöglich machen. Eine entsprechende Gegenmaßnahme stellt hierbei der Einsatz von SYN-Cookies dar, bei deren Einsatz sich der betroffene Rechner den Zustand der Verbindung nicht merken muss. Stattdessen sendet er ein Cookie zurück, in welches der Zustand eincodiert ist. Nur wenn die Gegenstelle auf dieses korrekt antwortet, kommt eine Verbindung zustande.

Eigenschaften und Funktionsmerkmale				
	Netfilter	**IPFilter**	**ipfirewall**	**pf**
Generell				
Portabel	✗	✓	✓	✓
Erste Regel entscheidet	✓	✗	✓	✗
IP-Optionen				
IP LSRR	✓	✓	✓	✗
IP SSRR	✓	✓	✓	✗
IP Record Route	✓	✓	✓	✗
IP Timestamp	✓	✓	✓	✗
IP Fragmentation	✓	✓	✓	✓
IP Short Fragments	✗	✓	✗	✓
Lokale Prozesse				
UID senden/empfangen	✓/✗	✓/✓	✗/✗	✗/✗
GID senden/empfangen	✓/✗	✓/✓	✗/✗	✗/✗
PID senden/empfangen	✓/✗	✗/✗	✗/✗	✗/✗
Sonstiges				
Stateful Filtering	✓	✓	✓	✓
TCP Control Bits	✓	✓	✓	✓
MAC-Adresse	✓	✗	✓	✓
Paketquantitäten	✓	✗	✗	✗
IPv6	✓	✓	✓	✓
Zeitintervalle	✓	✗	✗	✗

Tabelle 3.2: Unterstützung einzelner Funktionsmerkmale in den vorgestellten Paketfiltern

Weder das UDP- noch das ICMP-Protokoll verfügen über einen internen Zustand der Verbindung, daher wird deren Status im Normalfall nicht erfasst. Einige Paketfilter merken sich jedoch akzeptierte UDP-Pakete und lassen folgende Antworten in der Gegenrichtung zu. Weiterhin ist es möglich, dass ICMP-Pakete, die sich auf eine vorhandene TCP-Sitzung beziehen, vom Stateful Filtering erfasst werden und passieren dürfen.

3.3.4.2 Verhalten beim Ablehnen von Paketen

Wird ein Paket vom Paketfilter zurückgewiesen, so muss entschieden werden, ob und wie der Absender darüber informiert werden soll.

Viele Personen argumentieren für ein kommentarloses Ablehnen (DROP) der Pakete, da Angreifer so nicht unterscheiden können, ob die Pakete blockiert oder überhaupt nicht angekommen sind. Dies trifft jedoch auch auf die erwünschten Benutzer zu, die beim DROP keine entsprechende Reaktion auf ihre Anfrage erhalten, was unter anderem ident-Verbindungen ausbremst [siehe Donnerhacke, 2008].

Als REJECT bezeichnet man die Ablehnung eines Paketes mit einem entsprechenden Hinweis, dass eine Verbindung nicht möglich ist. Wie genau dieser Hinweis aussieht, lässt sich je nach Protokoll konfigurieren.

3.3.4.3 Optionen des Internet Protocols (IP)

Der RFC des Internet Protocols [siehe Postel, 1981b] sieht eine Reihe von Optionen für jedes IP-Datenpaket vor.

Vorgabe einer Route

Im Normalfall entscheiden Router dynamisch, auf welchem Weg sie ein Paket weiterleiten. Mit den Optionen Loose Source and Record Route (LSRR) sowie Strict Source and Record Route (SSRR) kann der Absender jedoch im Vorfeld entscheiden, welche Router bei der Übertragung beteiligt sein sollen.

Aufzeichnung der Route (Record Route)

Mit dieser Option kann der Weg eines Pakets zurückverfolgt werden. Dafür trägt jeder Router seine eigene Adresse an die durch einen entsprechenden Zeiger spezifizierte Stelle ein. Der verfügbare Speicherplatz reicht jedoch nur für neun Einträge aus. Außerdem ist der Eintrag durch einen Router freiwillig und muss daher nicht erfolgen.

Aus dem Blickwinkel der Sicherheit betrachtet ist diese Option kritisch, da hiermit das interne Netzwerklayout an Außenstehende preisgegeben werden kann.

Zeitstempel (Timestamps)

Ähnlich wie bei der Aufzeichnung der Route können bei dieser Option Router angewiesen werden, Zeitstempel in das IP-Paket einzufügen, die bei Bedarf auch die Adresse des Routers enthalten.

Grundsätzlich ergeben sich hier dieselben Probleme und Sicherheitsbedenken wie bei der vorigen Option, weshalb eine Filterung dieser Pakete – vor allem, wenn sie aus dem eigenen Netzwerk stammen – sinnvoll sein kann.

Fragmentierung

Große IP-Pakete können vor dem Versenden in kleinere Pakete aufgeteilt werden, welche dann einzeln übertragen werden. Dieser Vorgang wird *Fragmentierung* genannt. Die ein-

zelnen Pakete enthalten dann Informationen über die Zugehörigkeit, welche zum späteren Zusammensetzen benutzt werden.

Einige Paketfilter können diese Optionen ebenfalls verarbeiten und entscheiden, wie mit den einzelnen Fragmenten vorgegangen werden soll. So kann etwa ab dem zweiten Fragment nicht mehr festgestellt werden, welches der Quell- oder Zielport des ursprünglichen Paketes war und entsprechende Regeln des Paketfilters greifen nicht mehr. Eine gebräuchliche Methode ist hier, eine Regel für alle Fragmente ab dem zweiten zu schreiben.

Jeder der vier vorgstellten Paketfilter erlaubt eine entsprechende Überprüfung. Im Falle von pf werden die Fragmente noch vor der Regelüberprüfung zusammengesetzt. IPFilter hingegen erlaubt zusätzlich einen Test auf Fragmente, die zu klein sind, um auch nur einen IP-Header zu beinhalten.

Beachtet werden sollte hier auch der Umstand, dass gewisse Kombinationen von Framgentierungsoptionen nicht zugelassen sind. Allerdings wurden diese bereits in Angriffen eingesetzt[Ziemba u. a., 1995], weshalb sich auch hier eine Filterung anbietet.

3.3.4.4 TCP Control Bits

Jedes TCP-Paket enthält in seinem Header sechs sogenannte *Control Bits* [siehe Postel, 1981c] mit unterschiedlichen Bedeutungen:

URG Der Zeiger *Urgent* ist gültig

ACK Das Feld *Acknowledgment* ist gültig

PSH Der Empfänger soll das Paket baldmöglichst an die Anwendung weiterreichen

RST Zurücksetzen der Verbindung

SYN Synchronisierung der Sequenznummern

FIN Es folgen keine weiteren Daten vom Sender

Einige Kombinationen dieser Bits sind nicht erlaubt und können daher bei der Verarbeitung zu unvorhergesehenen Ergebnissen führen. Ein Beispiel hierfür ist der sogenannte *Null Scan*, bei dem keines der Control Bits gesetzt ist.

Aus diesem Grund ist es in vielen Paketfilter möglich, auf bestimmte Kombinationen zu testen und diese entsprechend zu behandeln. Alle oben vorgstellten Paketfilter unterstützen diese Art von Test.

3.3.4.5 Auswertung der MAC-Adresse

Jede Netzwerkschnittstelle verfügt über eine von der Hardware vorgegebene, eindeutige Adresse. Diese wird vom Hersteller festgelegt und als *MAC-Adresse* bezeichnet. Im OSI-Modell wird sie der Sicherungsschicht zugordnet und dient der eindeutigen Identifizierung von Komponenten in lokalen Netzwerken. Aufgrund dieser Zuordnung ist sie allerdings auf das lokale Netz beschränkt, da sie nicht mit IP-Paketen über das Internet übertragen wird.

Die MAC-Adresse eignet sich jedoch, um ungewollten Veränderungen an lokalen Rechnern vozubeugen. Wurde beispielsweise ein Notebook unberechtigterweise mit einem entsprechenden Anschluss verbunden, so kann ein Paketfilter aufgrund der unbekannten

Hardware-Adresse den Zugang blockieren. MAC-Adressen können jedoch leicht durch Software gefälscht werden [Wikipedia, 2008]. Ihre Blockierung sollte daher höchstens eine zusätzliche Schutzmaßnahme bilden.

Sowohl Netfilter als auch ipfirewall unterstützen diese Art der Überprüfung direkt im Paketfilter. Zwar bietet pf die gleiche Funktionalität, allerdings müssen Pakete vorher außerhalb des Paketfilter entsprechend markiert werden. Einzig IPFilter ist nicht in der Lage, auf MAC-Adressen zu filtern.

3.3.4.6 Limitierung von Paketquantitäten

In einigen Situationen – beispielsweise zur Abwehr von DoS-Angriffen – ist es hilfreich, die Anzahl erlaubter Pakete eines bestimmten Typs künstlich zu limitieren. Kann ein Rechner etwa die Flut ankommender ICMP-Pakete nicht mehr bewältigen, so lassen sich diese auf eine Frequenz wie zehn Pakete pro Minute drosseln. Durch diese Maßnahme wird der Dienst nicht vollständig blockiert, sondern auf ein beherrschbares Maß reduziert.

Betrachtet man die reinen Paketfilter, so ist nur Netfilter zu einer direkten Spezifizierung einer solchen Limitierung fähig. Die restlichen drei Produkte sind jedoch in Kombination mit einem entsprechenden Traffic Shaper zu ähnlichen Funktionen in der Lage. Dabei ist jedoch meist nur eine Limitierung der Datengröße und nicht der Anzahl an Datenpaketen möglich.

3.3.4.7 Lokale Prozesse

Sollen auf dem Rechner, welcher den Paketfilter ausführt, Prozesse aufgrund ihrer Zugehörigkeit zu Nutzern oder Gruppen beschränkt werden, so lässt sich dies ebenfalls mit einigen Produkten umsetzen. Auf Unix- und Linux-Systemen läuft jedes Programm unter einer eindeutigen Benutzer- (UID) oder Gruppenkennung (GID). Dabei besitzt es zusätzlich eine eigene Prozess-ID (PID). Jede dieser drei Kennungen kann zur Filterung von ein- und ausgehendem Datenverkehr benutzt werden.

Da diese Informationen nicht Teil der eigentlichen Datenpakete sind, ist eine solche Beschränkung nur schwer und auch nicht immer ganz fehlerfrei möglich. Netfilter erlaubt die Limitierung auf alle drei IDs, jedoch nur in ausgehender Richtung. Ipfirewall kennt nur UID und GID, kann jedoch auch eingehende Verbindungen erkennen.

Sowohl pf als auch IPFilter lassen eine solche Funktion hingegen vermissen.

3.3.4.8 IPv6

Das in Unterabschnitt 3.2.1 auf Seite 9 beschriebene Verfahren der Network Address Translation bremst zwar das Problem der immer knapper werdenden IP-Adressen, kann es jedoch auf Dauer nicht lösen. Derzeit laufen daher die Bemühungen einer möglichst baldigen Umstellung auf das neue IPv6 auf Hochtouren. Mit diesem im RFC 4291 [Hinden u. Deering, 2006] beschriebenen Protokoll existieren ca. $3,4 \cdot 10^{38}$ Adressen.

Aktuelle Betriebssysteme sollten daher komplett IPv6-fähig sein. Alle vorgestellten Paketfilter unterstützen die Erkennung und Filterung entsprechender Pakete.

3.3.4.9 Zeitintervalle

Für bestimmte Regeln kann eine Einschränkung durch ein vorher definiertes Zeitintervall sinnvoll sein. Erlaubt eine Regel beispielsweise den Zugang von Mitarbeitern auf die Firmenrechner, so bietet sich eine Beschränkung auf Werktage an. Ein anderes Szenario wäre das Protokollieren von möglichen Angriffen nur während der Nacht, um Fehlalarme durch Mitarbeiter auszublenden.

Derzeit ist Netfilter der einzige Paketfilter mit dieser Funktionalität.

3.3.5 Überprüfung

Im Anschluss an die Erstellung oder Änderung eines Regelsatzes für einen Paketfilter empfiehlt sich eine Kontrolle, ob das zugrundeliegende Konzept (weiterhin) korrekt abgebildet wird. So kann es beispielsweise geschehen, dass eine neu eingeführte Regel unabsichtlich eine andere außer Kraft setzt. In einem anderen Fall ist es möglich, dass eine Regel keine Auswirkung hat, da alle möglicherweise zutreffenden Pakete bereits von einer früheren Regel abgefangen wurden.

Die gängigste Methode zur Überprüfung eines Regelsatzes stellen derzeit sogenannte Paketgeneratoren wie FTester [Barisani, 2007] dar. Während der eine Teil dieser Software als *Sniffer* hinter dem Paketfilter akzeptierte Pakete aufzeichnet, sendet ein *packet injector* speziell bereitete Datenpakete. Ein späterer Abgleich der Protkolldaten beider Programme liefert Aufschluss darüber, ob der Filter den Anforderungen genügt.

Als großer Nachteil dieser Methode kann der Aufwand angesehen werden, der für diesen Test notwendig ist. So muss der entsprechende Firewall-Rechner bereits vollständig konfiguriert und mit dem realen Netz verbunden sein. Tauchen Fehler in der Konfiguration auf, sind unter Umständen Dienste temporär ungesichert oder notwendige Verbindungen können nicht aufgebaut werden. Tests in einer anderen, weniger kritischen Umgebung lassen hingegen möglicherweise Spezialfälle außer acht. Wünschenswert ist daher eine Validierung der Regelsätze vor deren Implementation.

3.4 Zusammenfassung

In diesem Kapitel wurden die Aufgabenbereiche einer Firewall erläutert und näher auf die hierfür erforderlichen Paketfilter eingegangen. Eine Gegenüberstellung verschiedener Produkte zeigte jedoch, dass sich weder im Funktionsumfang noch in der Syntax ein Standard abzeichnet. Die Wahl des richtigen Paketfilters hängt daher stark von den umzusetzenden Anforderungen ab.

Eines der Probleme an dieser Stelle ist die notwendige Einarbeitung in die Details des gewählten Filters. Soll dieser zu einem späteren Zeitpunkt ausgetauscht werden, ist ein Wechsel nicht ohne weiteres möglich. Ein anderes Problem stellt die fehlende Möglichkeit zur Überprüfung umgesetzter Regelsätze dar, die ohne eine reale Implementierung auskommt.

4 Verifikation von Firewalls

Nachdem im letzten Kapitel der Aufbau sowie die Funktionsweise einer Firewall im Detail betrachtet wurden, widmet sich dieses Kapitel der Verifikation des dabei verwendeten Paketfilters. Hierzu werden die Möglichkeiten des *Model Checking* erläutert und mit denen einer Validierung verglichen.

4.1 Model Checking

Bei dem Verfahren des *Model Checking* handelt es sich um eine Methode zur Verifikation von Systembeschreibungen, wie sie etwa mit dem jABC erzeugt werden können. Die Überprüfung eines Systems erfolgt dabei in drei Schritten. Zuerst muss eine abstrakte Darstellung $\mathcal{M}$ des konkreten Systems erzeugt werden, beispielsweise durch die Überführung in einen Automaten. Anschließend ist eine formale Spezifikation ϕ der zu verifizierenden Eigenschaft festzulegen, was meist durch die Angabe einer temporallogischen Formel erfolgt. Ein spezielles Programm, der *Model Checker*, überprüft dann, ob das Modell die Spezifikation erfüllt, notiert als $\mathcal{M} \models \phi$. Das Verfahren läuft automatisch und benötigt daher keine Interaktion durch den Benutzer.

Schlägt die Verfikation dieser Eigenschaft für das Modell fehl, so können auf diese Weise Fehler im Modell aufgedeckt werden. Kann der Model Checker jedoch die Erfüllung der Spezifikation bestätigen, so gilt das Modell als verifiziert.

4.1.1 Spezifikation von Modellen

Eine häufig verwendete Art der Abstraktion eines realen Systems ist die eines endlichen Automaten $\mathcal{A} = \langle Q, T, q_0, l \rangle$, wobei Q die Menge der Zustände und $T \subseteq Q \times Q$ die möglichen Übergänge zwischen diesen bezeichnet. Essenziell für das Model Checking ist zusätzlich die Zuordnung $l : Q \to 2^{AP}$, die für jeden Zustand aus Q eine Menge von atomaren Eigenschaften spezifiziert, welche in diesem gelten.

In Abbildung 4.1 wird das Modell eines Überwachungssystems gezeigt. Der Automat besitzt die drei Zustände q_0, q_1 und q_2, von denen q_0 den Startzustand darstellt. Von dort kann das System bei Erkennung eines Problems in den Zustand q_1 wechseln, der eine Warnung repräsentiert. Verschlimmert sich das Problem weiter, so wird dies durch den Zustand q_2 ausgedrückt. Nach Beseitigung der Störung kann aus den Zuständen q_1 und q_2 direkt zurück in den Startzustand gewechselt werden. Mögliche Pfade in diesem Modell sind zum Beispiel (q_0, q_1, q_2, q_0) und (q_0, q_1, q_0).

Jeder Zustand ist mit einer Teilmenge von atomaren Eigenschaften aus der Menge $\{\texttt{CRIT}, \texttt{OK}, \texttt{WARN}\}$ versehen. Über diese Eigenschaften lassen sich Formeln definieren, die von einem Model Checker überprüft werden können. Eine zu verifizierende Eigenschaft, die für dieses System von Interesse ist, ist die Frage, ob der kritische Zustand nur erreicht werden kann, wenn vorher der Warnungs-Zustand aktiv war. Auch kann bewiesen werden,

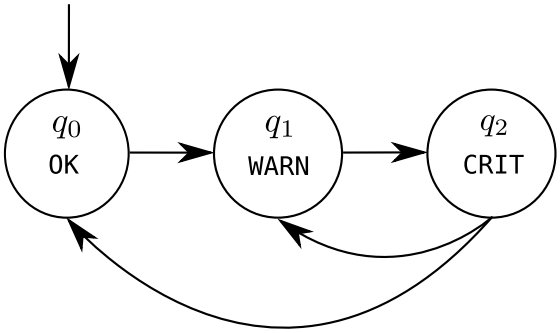


Abbildung 4.1: Einfacher Automat, der die Zustände eines Paketfilters darstellt

dass es keine Zustände gibt, aus denen das System nicht mehr zurück in den Startzustand gelangen kann. Für die Beantwortung solcher Fragen, die sich auf einen zeitlichen Ablauf beziehen, wird eine *Temporallogik* benötigt.

4.1.2 Temporallogiken

Im Gegensatz zu einer klassischen Logik wie der Aussagen- oder Prädikatenlogik lassen sich in einer Temporallogik Aussagen über zeitliche Abläufe definieren und auswerten. Die zu betrachtenden Zeitpunkte werden hierbei diskret behandelt, das heißt es existiert keine Auswertung der Systeme in Realzeit. Bei dem in Abbildung 4.1 gezeigten Automat ist daher die kleinste mögliche Zeiteinheit der Übergang von einem Zustand in den nächsten.

Eine Temporallogik enthält neben den Operatoren für Negation ($\neg$), Konjunktion ($\wedge$) und Disjunktion ($\vee$) auch modale Operatoren, die Aussagen über Pfade ermöglichen. So erlaubt etwa der Operator F (Finally) eine Aussage über die Eigenschaften zukünftiger Zustände. Die Formel F `OK` besagt zum Beispiel, dass einer der folgenden Zustände die Eigenschaft `OK` erfüllt und wir somit immer den Startzustand erreichen können. Analog dazu lassen sich mit dem Operator X (Next) Aussagen über den nächsten Zustand treffen. Der Operator U (Until) hingegen erlaubt Aussagen wie „bis zum Auftreten von A gilt B“.

Da auf einem Automaten verschiedene Pfade möglich sind, benötigt man für temporallogische Formeln weiterhin *Pfadquantoren*. Diese ermöglichen Aussagen über die Menge aller Pfade wie „es existiert ein Pfad“ (E für Exists) oder „auf allen Pfaden gilt“ (A für All). Um also zu überprüfen, ob das Erreichen eines Zustandes mit der Eigenschaft `OK` immer möglich ist, müsste die Formel EF `OK` für das Modell verifiziert werden.

4.1.3 Modellierung von Paketfiltern

Für die Verifikation einer Firewall beziehungsweise des verwendeten Paketfilters ist zuvor die Erstellung eines entsprechenden Modells notwendig. Zu diesem Zweck bietet sich eine grafische Repräsentation der Filterregeln an, wie sie exemplarisch in Abbildung 4.2 dargestellt ist.

Der Startpunkt des Graphen wird mit dem Zustand i gekennzeichnet, welcher ein neu eingehendes, zu verarbeitendes Netzwerkpaket symbolisiert. Zuerst wird für dieses im nächsten Zustand s ermittelt, ob es einer bereits existierenden Verbindung zugeordnet

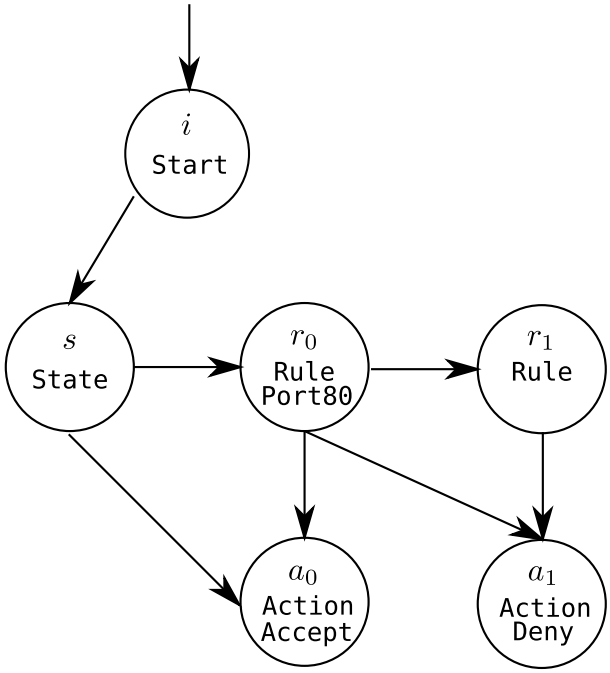


Abbildung 4.2: Modell eines Paketfilters in Form eines Graphen

werden kann. Ist dies nicht der Fall, müssen die Regeln des Paketfilters ausgewertet werden. In diesem Beispiel beschränkt sich die Anzahl auf zwei Regeln: Nur eingehende Pakete, die an den internen Webserver adressiert sind, werden akzeptiert, der Rest hingegen abgelehnt. Beide Regeln werden jeweils über einen Knoten r_0 beziehungsweise r_1 dargestellt. Je nach getroffener Entscheidung des Paketfilters endet der Pfad in einem der Knoten a_0 oder a_1, die ein Akzeptieren oder Ablehnen des Paketes repräsentieren.

Alle Regeln in diesem Modell wurden mit der atomaren Eigenschaft `Rule` versehen, während Aktionen die Eigenschaft `Action` erhielten. In diesem Modell kann davon ausgegangen werden, dass ein Pfad immer im Startzustand mit der Eigenschaft `Start` beginnt und in einem `Action`-Knoten endet. Dies entspräche der temporallogischen Formel

$$\texttt{Start} \Rightarrow AF(\texttt{Action} \wedge \neg EX\ \texttt{true})$$

Aussagen über die Struktur des Graphen sind zwar hilfreich, jedoch lassen sich hiermit noch keine Aussagen über das Verhalten bei der Verarbeitung von Paketen machen. Um dies zu ermöglichen, müssen die Regeln zuvor entsprechend klassifiziert werden. Die Eigenschaft `Port80` in obigem Modell soll etwa bedeuten, dass diese Regel Pakete beschreibt, die an den Webserver gerichtet sind. Schränkt man dann die Klasse der möglichen Pakete so ein, dass sie genau diese gewünschten Parameter aufweisen, so ließe sich mit folgender Formel beweisen, dass es einen Pfad gibt, auf dem diese Pakete akzeptiert werden:

$$\texttt{Start} \Rightarrow AF(\texttt{Port80} \wedge EX\ \texttt{Accept})$$

Um die Anwendung einer solchen Formel zu ermöglichen, müssen im Vorfeld alle Regeln, die auf diese Art von Paketen zutreffen, mit der Eigenschaft `Port80` versehen werden. Davon betroffen sind auch Regeln, die beispielsweise alle TCP-Pakete ohne Rücksicht auf

den gesetzten Port abdecken. Andernfalls könnte eine solche Regel, die im Pfad vor der akzeptierenden liegt, entsprechende Pakete vorher ablehnen.

Der verfizierende Model Checker arbeitet auf dem statischen Modell des Paketfilters und hat daher keine Daten über die zur Laufzeit ermittelten Informationen. Diese umfassen unter anderem auch die Verbindungen, welche vor Eintreffen eines Paketes etabliert wurden. Da Pakete jedoch bei Zugehörigkeit zu einer bereits existierenden Verbindung automatisch akzeptiert werden, kann diese Funktion so nicht mit Model Checking verifiziert werden.

4.2 Validierung

Für die Überprüfung eines realen Systems mit Hilfe einer Validierung ist im Gegensatz zur Verifikation nicht die Abstraktion in ein Modell sondern die Spezifikation von Testfällen notwendig. Jeder Testfall spezifiziert dabei eine erwünschte Reaktion des Systems. Entspricht die während des Tests ermittelte Reaktion dann der erwarteten, gilt der Test als bestanden.

Bei dieser Vorgehensweise ist eine große Sorgfalt während der Auswahl der Testfälle notwendig. Werden etwa spezielle Vorgänge oder Anforderungen nicht bedacht, so bescheinigen die übrigen Tests eine korrekte Funktionalität, die jedoch im praktischen Betrieb nicht gegeben ist.

Überträgt man den allgemeinen Vorgang der Validierung auf Paketfilter, so bestehen die zu spezifizierenden Testfälle aus einzelnen Netzwerkpaketen. Nach der Auswertung eines Paketes steht die Aktion fest, mit welcher der Paketfilter reagieren würde. Entspricht diese Aktion der gewünschten, so gilt der Test als erfolgreich. Ein erster Ansatz zur Ermittlung solcher Pakete stellen die Anforderungen dar, welche zu Beginn im Konzept der Firewall definiert wurden.

Betrachtet man etwa die Anforderungen der fiktiven Firma, welche zu Beginn des Kapitels 2 genannt wurden, so lassen sich daraus leicht entsprechende Testfälle ableiten. So muss einerseits die Verbindung zum Webserver als auch der Mitarbeiterzugang zum Internet überprüft werden. Ein weiterer Test könnte jedoch zusätzlich auch die Verbindung zum Webserver von den Arbeitsplätzen der Mitarbeiter abdecken. Diese kommt zwar im ursprünglichen Konzept nicht vor, soll aber wahrscheinlich möglich sein, da keine entsprechende Ausnahme existiert. Durch Aufnahme dieses Testfalls wird verhindert, dass spätere – und möglicherweise komplexe – Änderungen am Konzept oder der Firewall diese Verbindung unbemerkt blockieren.

4.3 Zusammenfassung

Das Verfahren des *Model Checkings* stellt ein effizientes Mittel zur Verifikation von Systemmodellen dar. Anhand spezieller Formeln lassen sich mit Hilfe einer automatisiert laufenden Software Aussagen über die Korrektheit des zu prüfenden Systems ermitteln.

Dieses Prinzip scheint allerdings nicht die optimale Lösung für das in dieser Arbeit spezielle Problem darzustellen. Aus diesem Grund wird im Folgenden der Ansatz der *Validierung* weiter verfolgt, dessen Aussagekraft jedoch in Abhängigkeit von der Qualität der Testfälle steht.

5 Verwendete Komponenten

Die Software, welche im Rahmen dieser Diplomarbeit entwickelt wurde, basiert auf zwei bereits exstierenden Produkten. Zum einen ist dies der *Firewall Builder*, mit dem Firewall-Regeln in einem grafischen Editor unabhängig vom verwendeten Paketfilter zusammengestellt werden können. Darstellung und Simulation erfolgen zum anderen im *Java Application Building Center* unter Zuhilfenahme bestehender Plugins.

5.1 Konfigurationseditor für Paketfilter

Betrachtet man die Konfigurationsbeispiele aus Abschnitt 3.3, so wird schnell deutlich, dass die Konfiguration und Wartung von Paketfiltern über deren eigene Syntax ein komplexer und mitunter auch fehleranfälliger Prozess ist. Anstatt die Funktionsweise eines Paketfilters im Detail zu verstehen, möchten sich viele Nutzer die notwendigen Regeln möglichst einfach „zusammenklicken“. Doch auch Administratoren mit dem nötigen Hintergrundwissen können eine unterstützende grafische Oberfläche zu schätzen wissen, die zusätzlich einen Überblick über die vorhandene Netzwerktopologie verschafft.

An diesem Punkt setzen mehrere auf dem Markt befindliche Softwareprodukte mit unterschiedlicher Funktionsvielfalt an (siehe Tabelle 5.1). Die einfachste Form stellt beispielsweise das Programm *Firestarter* dar, welches auf einem Linux-System mit wenigen Mausklicks einen Paketfilter auf Basis von Netfilter nach den Vorgaben des Benutzers zusammenstellt und einrichtet. Allerdings ist diese Konfiguration eher für Einzelplatzrechner und einfache Router gedacht. Mit dem ebenfalls unter Linux lauffähigen *Knetfilter* erhält man ein Frontend für Netfilter, das mehr Möglichkeiten bereitstellt, jedoch auch eine stärkere Einarbeitung in den Paketfilter erfordert.

Einen anderen Ansatz verfolgt der unter Microsoft Windows lauffähige *Solsoft Firewall Manager*, der eine grafische Spezifizierung von komplexen Netzen mit mehreren verschiede-

Konfigurationseditoren				
	Firestarter	**KNetfilter**	**Solsoft FM**	**Firewall Builder**
Mehrere Paketfilter	✗	✗	✓	✓
Betriebssystemunabhängig	✗	✗	✗	✓
Grafische Netzwerkdarstellung	✗	✗	✓	✗
Open Source	✓	✓	✗	✓
XML als Datenformat	✗	✗	✗	✓

Tabelle 5.1: Funktionsübersicht der Konfigurationseditoren für Paketfilter

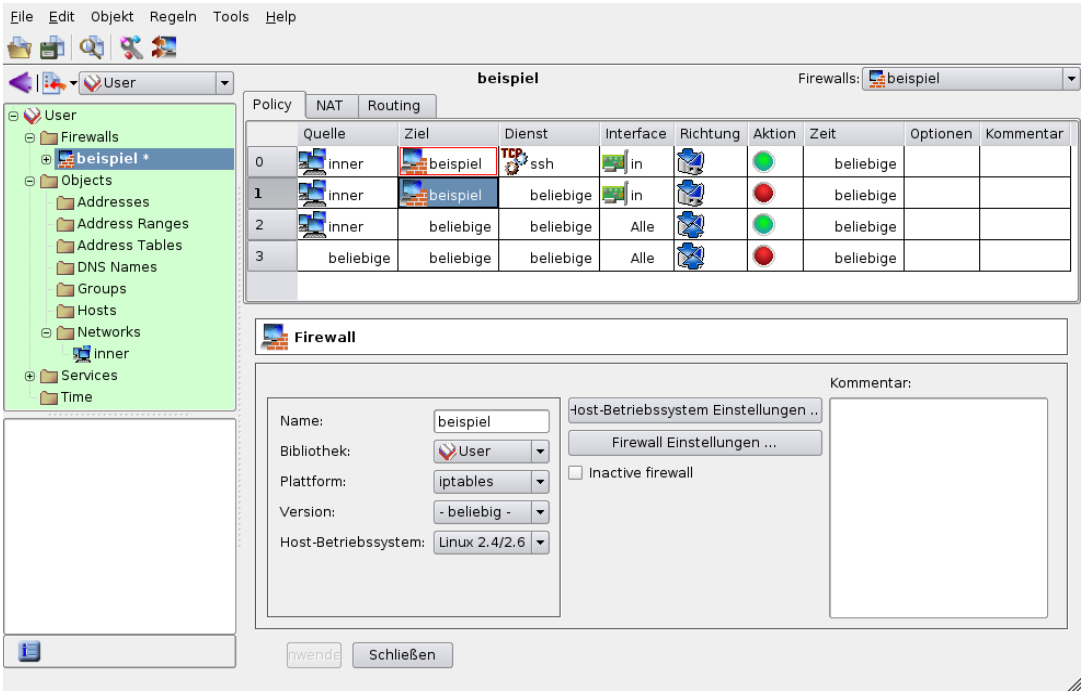


Abbildung 5.1: Screenshot des Firewall Builders

nen Paketfiltern ermöglicht. Als Nachteil dieses kommerziellen Produktes ist jedoch neben den Lizenzkosten die fehlende Möglichkeit zur einfachen Weiterverarbeitung der Daten in externen Programmen zu sehen.

In dieser Diplomarbeit wird daher der *Firewall Builder* verwendet. Es handelt sich um eine unter der GNU General Public License (GPL) entwickelte Kombination aus grafischer Oberfläche zum Definieren der Netzwerkkomponenten und einer Reihe von Compilern, die aus den eingegebenen Regeln den entsprechenden Paketfilter-Code generieren. Die Vorteile dieses Produkts liegen unter anderem in dem offenen Dateiformat sowie der breiten Unterstützung für frei verfügbare Paketfilter. Zusätzlich funktioniert der Firewall Builder sowohl unter Linux, Windows und Mac OS X, was dem Wunsch nach einer Betriebssystemunabhängigkeit entgegenkommt.

5.1.1 Graphischer Editor

Abbildung 5.1 zeigt die Eingabemaske des Firewall Builders. Über diese kann der Anwender Objekte wie zum Beispiel Hosts oder Netzwerkdienste erzeugen und in einer Tabellenstruktur zu Regeln zusammenfügen. Dabei wird von dem verwendeten Paketfilter abstrahiert. So erfolgt beispielsweise die Abarbeitung der Regeln immer von oben nach unten, wobei die erste zutreffende Regel – im Gegensatz zu IPfilter – die auszuführende Aktion für das Paket festlegt.

Jede Zeile der Tabellenstruktur stellt eine Regel dar und besteht dabei aus bis zu neun Spalten. Ob eine Spalte verwendet werden kann, hängt vom verwendeten Paketfilter ab. Beispielsweise ist die Limitierung auf eine Zeitspanne nur bei Netfilter möglich, bei anderen Paketfiltern wird diese Spalte ausgeblendet. Die folgende Liste erläutert die Bedeutungen

der einzelnen Spalten:

Quelle Eine Liste von Rechnern, Interfaces, oder Netzwerken, von deren IP-Adressen ein Datenpaket abgesendet wird

Ziel Eine Liste von Rechnern, Interfaces oder Netzwerken, die ein Datenpaket empfangen und anhand ihrer IP-Adressen identifiziert werden

Dienst Eine Liste von Netzwerkdiensten wie zum Beispiel SSH oder DHCP

Interface Angabe einer genauen Netzwerkschnittstelle der Firewall, über die der Datenverkehr gesendet oder empfangen wird

Richtung Eingehend, ausgehend oder beides. Hierbei ist nur die Richtung relativ zum Paketfilter und nicht zum Netzwerk gemeint. Verbindungen vom internen LAN ins Internet sind etwa sowohl eingehend (bis zum Paketfilter) als auch ausgehend (auf dem restlichen Weg).

Aktion Die durchzuführende Aktion für den Fall, dass die Regel auf das Paket zutrifft. Hier stehen zum Beispiel die Möglichkeiten *Accept*, *Deny* oder *Reject* zur Auswahl.

Zeit Eine Zeitspanne, während derer diese Regeln gültig ist

Optionen Steuert unter anderem das Logging

Kommentar Ein Freitextfeld für eigene Bemerkungen

Im Einzelnen bietet der Editor unter anderem folgende Möglichkeiten:

- Zur Erhöhung der Übersicht lassen sich Objekte zu Gruppen zusammenfassen, welche ihrerseits ebenfalls Teile von Gruppen sein können.
- Informationen über die eigene Konfiguration können über das Simple Network Management Protocol (SNMP) ermittelt werden.
- Mehrere Firewalls können gleichzeitig verwaltet werden und nutzen so nur einen Datenbestand.
- Verwendung einer mitgelieferten Standard-Bibliothek, welche diverse vordefinierte Elemente wie zum Beispiel häufig benutzte Protokolle enthält.

5.1.2 Konverter

Aus den zuvor angelegten Regeln erzeugt der Firewall Builder mit Hilfe des entsprechenden Konverters eine Konfigurationsdatei für den Paketfilter. Zusätzlich zu den in Unterabschnitt 3.3.3 vorgestellten Produkten unterstützt der Firewall Builder seit der Version 2.1.18 auch Cisco PIX und Cisco IOS, welche zuvor nur als kostenpflichtige Module erhältlich waren.

Innerhalb des Firewall Builders und in dessen Dokumentation wird diese Konvertierung durchgehend als *kompilieren* bezeichnet und die entsprechenden Module als *Compiler* betrachtet. Diese Begriffe wurden für diese Arbeit übernommen, um eine einheitliche Bezeichnung zu gewährleisten.

Während der Erzeugung führen die Compiler zusätzlich eine Überprüfung der Regelsätze durch. Beispielsweise wird auf das Vorhandensein von sogenannten Shadow-Regeln geprüft. Dabei handelt es sich um Regeln, die sich als redundant herausstellen und demnach entfernt werden können.

Im Anschluss an die Übersetzung in das Format des jeweiligen Paketfilters kann die Datei auf Wunsch automatisch durch den Firwall Builder auf dem entsprechenden Rechner installiert werden.

5.1.3 Datenformat

Zur Speicherung der eigenen Datenstruktur verwendet der Firewall Builder eine Datei, die mit Hilfe der Extensible Markup Language (XML) erzeugt wird und in der alle Objekte und Regeln abgelegt werden. Dies ermöglicht es unter anderem Entwicklern anderer Software, die Daten ohne großen Aufwand einzulesen.

Das folgende Beispiel bezeichnet eine Gruppe mit dem Namen „Networks“, die aus einem Netzwerk mit der Adresse 129.217.1.0 besteht:

```
<ObjectGroup id="id4784F8405569" name="Networks">
    <Network comment="" id="id4784F8575569" name="inner"
      address="129.217.1.0" netmask="255.255.255.0"/>
</ObjectGroup>
```

5.2 Java Application Building Center

Seit Mitte der neunziger Jahre wird am Lehrstuhl V des Fachbereichs Informatik an der Technischen Universität Dortmund ein Framework zur Modellierung von Systemabläufen entwickelt. Während die ursprüngliche Version mit C++ geschrieben wurde, basiert das aktuelle jABC [Steffen u. a., 2006] auf der plattformunabhängigen Sprache Java. Die modulare Architektur ermöglicht eine einfache Entwicklung von zusätzlichen Plugins, von denen einige am selben Lehrstuhl implementiert wurden und mit dem jABC ausgeliefert werden.

Bei der Modellierung von Abläufen mit diesem Framework werden zwei Nutzergruppen unterschieden. Zum einen gibt es *Anwendungsexperten*, die über detaillierte Kenntnisse des abzubildenden Systems verfügen, jedoch keine Programmierkenntnisse benötigen. In der grafischen Oberfläche (siehe Abbildung 5.2 auf der nächsten Seite) des jABC stellen sie Abläufe aus speziellen Komponenten dar. Eine solche Komponente wird als Service Independent Building Block (SIB) bezeichnet und kann sowohl ein- als auch ausgehende Kanten besitzen, über die es mit anderen SIBs verbunden ist. Die Implementierung der SIBs wird zum anderen von der zweiten Nutzergruppe, den *SIB-Experten*, durchgeführt.

Eigenschaften der SIBs werden unter anderem im sogenannten *SIB-Inspektor* dargestellt. Inspektoren sind spezielle Bereiche im jABC, die Kontextinformationen zu SIBs oder Graphen darstellen, jedoch auch durch Plugins selbst definiert werden können.

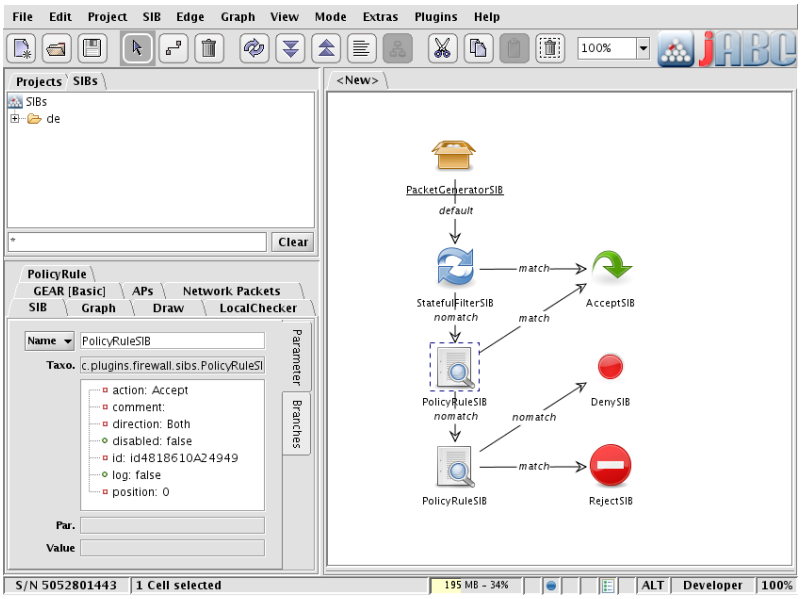


Abbildung 5.2: Oberfläche des jABC

5.2.1 LocalChecker

Mit Hilfe des LocalChecker-Plugins[Neubauer, 2008] von Johannes Neubauer ist es möglich, lokale Eigenschaften von SIBs nach vorgegebenen Kriterien zu überprüfen. Zu den überprüfbaren Elementen gehören sowohl die SIB-Parameter als auch eventuelle Kanten zu anderen SIBs. Bereits implementierte Standardtests umfassen beispielsweise Prüfungen auf Kanten, deren zweiter Endpunkt keinem SIB zugewiesen ist. Zusätzlich ist es möglich, eigene Tests für das entsprechende SIB einzubinden. Abbildung 5.3 zeigt die Ansicht für einen Testfall innerhalb des jABC.

Eine Überprüfung kann entweder manuell gestartet oder permanent durchgeführt werden. Die Ergebnisse werden im Inspektor des Plugins dargestellt, wobei eine farbliche Codierung die Schwere des Fehlers visualisiert.

5.2.2 Tracer-Plugin

Bei dem Tracer-Plugin, welches von Markus Doedt entwickelt wurde, handelt es sich um eine Erweiterung für das jABC, die eine Ausführung der erstellten Modelle erlaubt (siehe [Doedt, 2006]). Wahlweise kann diese komplett oder auch schrittweise erfolgen. Daten zum aktuellen Durchlauf können in dem sogenannten *Context* abgelegt werden und stehen den nachfolgenden Schritten zur Verfügung. Der Tracer ermöglicht außerdem eine Anzeige des ausgeführten Pfades in der grafischen Oberfläche des jABC.

5.2.3 GEAR

Anders als der LocalChecker überprüft das Plugin für „Game-based, Easy and Reverse model-checking“ (GEAR) von Marco Bakera und Clemens Renner nicht nur die Eigenschaften direkt zusammenhängender SIBs, sondern erlaubt Aussagen über ganze Pfade.

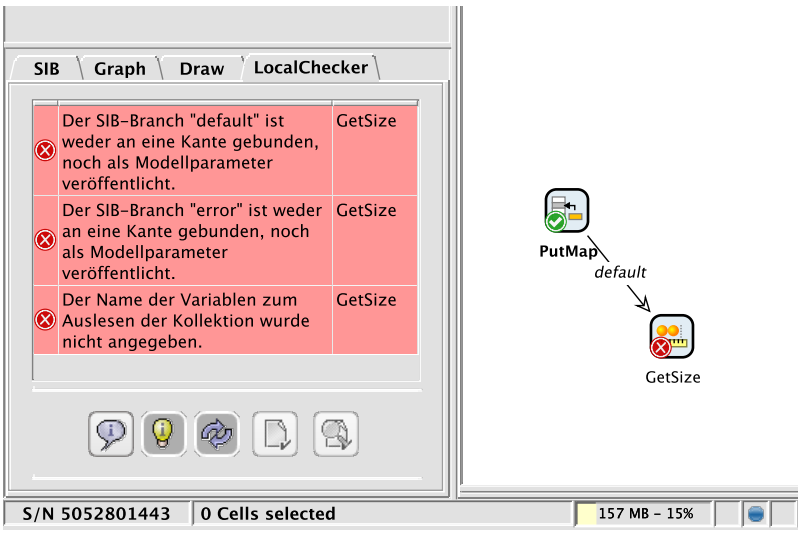


Abbildung 5.3: Das LocalChecker-Plugin im jABC. Grüne und rote Icons zeigen den Status einzelner SIBs, Warnmeldungen erscheinen im entsprechenden Inspektor.

Dies wird erreicht, indem es einen *Model Checker* bereitstellt, der Verifikationen ermöglicht wie sie in Kapitel 4 beschrieben wurden (siehe [Bakera u. a., 2007]).

Während der Erstellung von Graphen im jABC lassen sich dazu einzelne SIBs mit atomaren Eigenschaften versehen. Zusätzlich zu einer manuellen Vergabe können auch andere Parameter eines SIBs benutzt werden, etwa der Name der implementierenden Klasse. Nach der Angabe einer temporallogischen Formel kann dann einfach überprüft werden, ob das Modell diese Spezifikation erfüllt.

5.3 Zusammenfassung

Damit die umzusetzenden Regelsätze unabhängig vom verwendeten Paketfilter editiert werden können, ist eine Software notwendig, die eine abstrakte Ansicht dieser Regeln ermöglicht. Der Firewall Builder erfüllt diese Anforderung und bietet zudem eine einfache Exportmöglichkeit der erzeugten Daten.

Über ein zu entwickelndes Plugin sollen diese Daten mit dem jABC bearbeitet werden. Eine Reihe von existierenden Plugins bieten dabei bereits die Möglichkeit, den entstehenden Graphen zu überprüfen.

6 Firewall-Plugin

Im Anschluss an die Definition der Anforderungen und die Auswahl der hierfür erforderlichen Komponenten konnte die eigentliche Entwicklung des Plugins beginnen. Nach Festlegung des Konzepts, in welcher Art und Weise der spätere Arbeitsablauf der Erstellung einer Firewall erfolgen soll, musste die Art der Darstellung von Regelsätzen innerhalb des jABC sowie die dafür benötigte Java-Datenstruktur festgelegt werden. Diese bilden die Basis für den Import der externen Datenquellen, welcher anschließend fertiggestellt wurde.

Der nächste Schritt bestand in der Definition einer Struktur für virtuelle Datenpakete, welche von der Simulation ausgewertet werden. Während der gesamten Entwicklungsphase wurden mehrere Testfälle entwickelt, mit denen die Funktionen der Importmodule und des Simulators einzeln oder in Kombination validiert werden können.

6.1 Übersicht des zu entwickelnden Konzepts

Ein primäres Ziel der Entwicklung bestand darin, den zusätzlichen Aufwand der Validierung bei Erstellung einer Firewall zu minimieren. Von einer bestehenden Definition des Zielsystems im Firewall Builder ausgehend, sollten unter anderem unnötige Arbeitsschritte vermieden werden.

Das Ergebnis dieser Überlegungen ist in Abbildung 6.1 auf der nächsten Seite aufgezeigt, welche die Kombination der zuvor vorgestellten Komponenten sowie die vorgesehenen Abläufe darstellt. Eine Trennung der drei Ebenen verdeutlicht die Zusammengehörigkeit einzelner Arbeitsschritte beziehungsweise die Aufteilung auf verschiedene Anwenderprofile.

Planung

In der obersten Ebene werden die sicherheitstechnischen Anforderungen formuliert und in einer nicht exakt spezifizierten Form an die zweite Ebene weitergereicht. Dabei kann es sich um eine bereits formale Darstellung in Form eines jABC-Graphen handeln. Denkbar ist jedoch auch eine ausführliche textuelle Beschreibung, eine Multimedia-Präsentation oder auch nur eine stichwortartige Auflistung. Aufgrund der vielfältigen Möglichkeiten ist bei diesem Vorgang jedoch keine spezielle Software vorgeschrieben. Zusätzlich kann die Beschreibung je nach Verfasser und Umfeld variieren. Dennoch muss das Ergebnis auf der zweiten Ebene korrekt aufgefasst und umgesetzt werden.

Stellen sich bei der Implementation Probleme oder Widersprüche heraus, so müssen diese in ähnlicher Weise an die erste Schicht übermittelt werden. In diesem Fall muss das Konzept angepasst oder ergänzt werden, damit eine korrekte Umsetzung möglich ist.

Umsetzung und Simulation

Ausgehend von den zuvor definierten Anforderungen werden auf der mittleren Ebene die notwendigen Objekte wie Hosts oder Services im Firewall Builder eingetragen sowie entsprechende Regeln formuliert. Bei diesem Schritt handelt es sich wie bereits erwähnt um

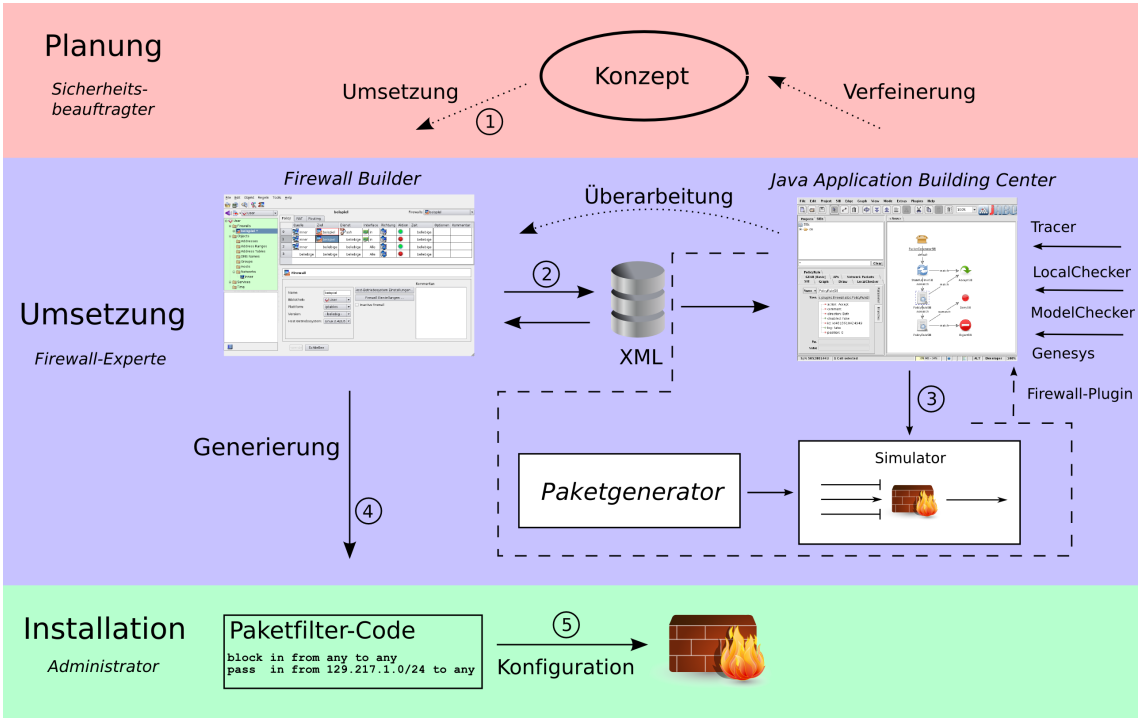


Abbildung 6.1: Kombination der verwendeten Komponenten

den fehleranfälligsten im gesamten Arbeitsablauf, weshalb das Ergebnis im Anschluss validiert werden sollte. Zu diesem Zweck werden die mit dem Firewall Builder erzeugten Daten im XML-Format abgespeichert und mit Hilfe eines zu entwickelnden Importfilters in eine Java-Datenstruktur eingelesen. Innerhalb des jABC können die Zusammenhänge der importierten Regelsätze dann in einer grafischen Ansicht durch entsprechende SIBs dargestellt werden.

Da eine Simulation des Paketfilters nicht ohne Testdaten geschehen kann, sind zuvor virtuelle Testpakete zu erstellen. Dies kann auf mehrere Arten geschehen, beispielsweise durch Anlegen einer entsprechenden XML-Datei oder den Import von realem Datenverkehr, welcher mit tcpdump aufgezeichnet wurde.

Nachdem alle notwendigen Daten mit dem jABC-Plugin geladen wurden, kann eine Simulation gestartet werden. Diese ermittelt sequenziell für jedes Paket, ob die im Firewall Builder angelegten Regeln die ursprünglichen Anforderungen korrekt umsetzen.

Sollte das Ergebnis nicht zufriedenstellend sein, so ist eine Überarbeitung der Regeln im Firewall Builder oder aber des gesamten Konzepts notwendig. Eventuelle Änderungen an der Reihenfolge der Regeln oder eine Änderung der ausgeführten Aktionen – etwa Annehmen oder Verwerfen – können direkt im jABC getestet werden. Mit Hilfe des LocalChecker-Plugins können an dieser Stelle Fehler wie Kanten, die ins Leere führen, erkannt und behoben werden. Der Einsatz des Model-Checker-Plugins *GEAR* erlaubt zudem eine Verifikation der möglichen Pfade im Modell.

Installation
Entspricht das Ergebnis letztendlich den Erwartungen, so wird der Regelsatz für den Paketfilter mit dem Firewall Builder erzeugt. Die Installation dieser fertiggestellten Datei erfolgt durch den Administrator des Firewall-Rechners und wird im Schaubild durch die dritte Ebene dargestellt.

6.2 Entwicklung

Zur Umsetzung des vorgestellten Arbeitsablaufs wurde im Rahmen dieser Diplomarbeit ein aus mehreren Einzelkomponenten bestehendes Firewall-Plugin für das jABC entwickelt, das in Abbildung 6.1 mit gestrichelten Linien gekennzeichnet ist und aus verschiedenen Komponenten besteht:

Import Zur Verarbeitung der Regeln wird eine Java-Datenstruktur benötigt. Diese soll mit den XML-Daten des Firewall Builders befüllt werden und alle dort eingegebenen Parameter aufnehmen können.

SIB-Palette Nach erfolgreichem Import sollen die Regelsätze im jABC grafisch dargestellt werden können. Zu diesem Zweck sind spezielle SIBs notwendig, die Knoten im Graphen wie einzelne Regeln oder Aktionen darstellen.

Paketgenerator Diese Komponente erzeugt aus verschiedenen Quellen virtuelle Pakete, welche vom Simulator überprüft werden.

Simulator Anhand der importierten Daten des Firewall Builders erkennt der Simulator für virtuelle Pakete, wie ein realer Paketfilter reagieren würde und stellt das Ergebnis grafisch innerhalb des jABC dar.

In diesem Abschnitt werden die einzelnen Arbeitsabläufe erläuert, welche für die Fertigstellung dieser Komponenten notwendig waren.

6.2.1 Semantik der jABC-Modelle

Zu Beginn der Entwicklung des Plugins stellte sich die Frage nach einer passenden Darstellung der Regelsätze innerhalb des jABC. Zu diesem Zweck musste in einem ersten Schritt festgelegt werden, auf welche Weise die Umwandlung in einen Graphen erfolgen soll.

Das Ergebnis dieser Überlegungen ist in Abbildung 6.2 auf der nächsten Seite dargestellt. Für jede im Firewall Builder angelegte Regel wird ein Knoten mit zwei ausgehenden Kanten erzeugt. Die Kanten decken jeweils den Fall ab, ob die Regel auf ein Paket zutrifft (Treffer) oder nicht (kein Treffer).

Beginnend mit dem obersten Knoten soll während der Simulation diese Frage für jedes Paket entschieden und den entsprechenden Kanten gefolgt werden. In den meisten Fällen zeigen diese bei einem *Treffer* auf eine Senke, welche eine Aktion wie *Akzeptieren* oder *Verwerfen* darstellt. An dieser Stelle ist die Simulation des Paketes beendet und die erreichte Aktion wird für das simulierte Paket vermerkt. Bei *Kein Treffer* hingegen landet das Paket meist bei der nächsten Regel. Die Verlinkung des letzten Knotens mit der Verwerfen-Senke setzt die standardmäßige Zurückweisung des Firewall Builders um, die alle Pakete betrifft, auf die keine Regel zutraf.

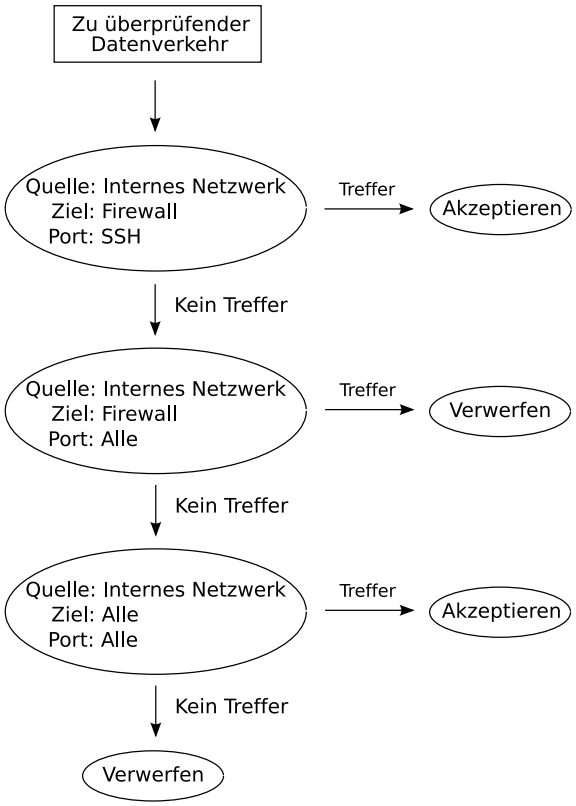


Abbildung 6.2: Darstellung eines Regelsatzes als Graph

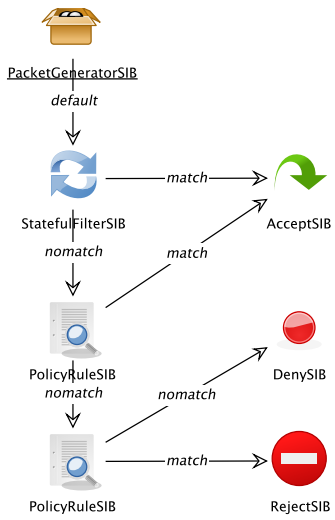


Abbildung 6.3: Importierter Graph, dargestellt als SIBs im jABC

An dieser Stelle sei jedoch angemerkt, dass die genannte Abbildung nur einen einfachen Fall demonstriert. Regelknoten müssen bei einem Treffer nicht zwingend auf eine Senke verweisen, sondern können auch in separate Regelketten münden. Zusätzlich muss eine Aktion nicht in jedem Fall als Senke dargestellt werden, beispielsweise beim Markieren von Paketen. Auf diese Weise können komplexe Graphen erzeugt werden, die – im ungünstigen Fall – nicht kreisfrei sein müssen.

6.2.1.1 SIB-Palette

Nach der Festlegung auf die Darstellung des Graphen begann die Planung der benötigten SIBs. Ein Beispielgraph aus diesen ist in Abbildung 6.3 dargestellt, wobei die Semantik mit der des Tracer-Plugins identisch ist: Jedes SIB ist *ausführbar* und spezifiziert so während eines Durchlaufs die nächste Kante der Ausführung.

Da jede Simulation einen Startpunkt besitzen muss, wurde das `PacketGeneratorSIB` entworfen, welches als Quelle der virtuellen Pakete dient und dessen einzige ausgehende Kante auf den ersten Regelknoten zeigt. Um die korrekte Funktionalität des Tracer-Plugins zu gewährleisten, wird dieses SIB als einziges mit dem Status *StartSIB* versehen.

Innerhalb der Datenstruktur wird jede Regel durch ein Objekt vom Typ `PolicyRule` abgebildet, das alle notwendigen Informationen zum Treffen einer Entscheidung über ein Paket besitzt. Daher werden die oben beschriebenen Regelknoten im Graphen des jABC durch ein `PolicyRuleSIB` repräsentiert, welches eine Referenz zur entsprechenden `PolicyRule` besitzt.

Eine Abart der Regelknoten stellt das `StatefulFilterSIB` dar. Hierbei handelt es sich um eine Abbildung des Verbindungskontextes, der in Unterabschnitt 3.3.4.1 auf Seite 18 erläutert wurde. Jede akzeptierte Verbindung wird automatisch in einer Datenbank aufgezeichnet, so dass weitere Pakete dieser Verbindung bei einer positiven Zuordnung automatisch angenommen werden können. Gewöhnlich steht dieses SIB direkt unterhalb des Paketgenerators und verweist bei fehlender Übereinstimmung mit dem Paket auf den ersten Regelknoten.

Senken im Graphen werden durch Aktionen markiert, beispielsweise das `AcceptSIB` oder `DenySIB`. Allerdings existieren auch Aktionen, die nicht zwingend das Ende der Simulation hervorrufen. Ein Beispiel hierfür ist das `TagSIB`, welches einkommende Pakete mit einer Marke versieht und an den nächsten Regelknoten weiterreicht.

6.2.2 Java-Datenstruktur

Zusätzlich zu den oben genannten SIBs, die Regelsätze eines Paketfilters darstellen, werden Objekte benötigt, welche die übrigen zur Simulation notwendigen Parameter aufnehmen. Zu diesen gehören etwa die im Firewall Builder angelegten Netzwerkrechner oder -dienste. Primär mussten daher sowohl die Kompatibilität zur Datenstruktur des Firewall Builders als auch eine effiziente spätere Verwendung im Simulator beachtet werden.

Da der Import eine zentrale Rolle für das zu entwickelnde Plugin spielt, wurden die in der Document Type Definition (DTD) des Firewall Builder beschriebenen Objekte möglichst exakt in Java-Klassen umgewandelt. Jedes Element erhielt eine gleichnamige Klasse, in der private Variablen die in der XML-Datei spezifizierten Attribute darstellen. Das Prinzip der Vererbung wurde fast ausschließlich für abstrakte Klassen benutzt, mit denen Objekte mit ähnlichen Eigenschaften – beispielsweise Netzwerkobjekte oder -protokolle – zusammengefasst wurden. Die Änderungen an der verarbeiteten DTD beliefen sich in den letzten Aktualisierungen des Firewall Builders auf wenige Zeilen, so dass die nach diesem Ansatz erstellten Klassen auch in Zukunft ohne aufwändige Änderungen mit aktuelleren Versionen der Software zusammenarbeiten können.

In der zugrundeliegeneden Datenstruktur besteht jede XML-Datei aus einer `FWObjectDatabase`, die eine Liste von `Library`-Objekten verwaltet. Innerhalb einer solchen `Library` befinden sich die im Firewall Builder angelegten Objekte wie Firewalls, Hosts oder Zeitintervalle. Jedes Objekt besitzt neben allgemeinen Attributen wie dem Namen oder einem zugewiesenen Kommentar eine ID, mit der es von anderen Objekten referenziert werden kann. Diese Referenzierung ist nicht auf die `Library` des referenzierenden Objekts beschränkt sondern kann auf ein beliebiges Objekt der gesamenten `FWObjectDatabase` verweisen. Zusätzlich können für jeden Typ – Netzwerkkomponente, Protokoll oder Zeitintervall – Gruppen existieren, welche beliebig viele entsprechende Objekte aufnehmen können.

Abbildung 6.4 auf der nächsten Seite stellt einen Ausschnitt der Datenstruktur dar. Hierbei handelt es sich um den Teil, der Objekte im realen Netzwerk abbildet. Zu diesen gehören Hosts, deren Netzwerkschnittstellen, ganze Netzwerke und auch die zu simulierende Firewall selbst. Alle Klassen erben hierbei von der abstrakten Klasse `NetworkObject`. Dies stellt sowohl die Existenz der allgemeinen Attribute wie `id` oder `name` als auch die Implementierung der Methoden für Import und Simulation sicher. Gruppierungen dieser Objekte sind mit Hilfe der Klasse `NetworkObjectGroup` realisierbar, wobei eine beliebig tiefe Verschachtetelung dieser Gruppen möglich ist.

Zentral ist an dieser Stelle die Klasse `Firewall` zu betrachten, die zwar auch als ein Objekt ähnlich eines `Hosts` benutzt werden kann, zusätzlich jedoch das Feld `policy` besitzt. In diesem werden alle Regelsätze, die im Firewall Builder für diese Firewall definiert wurden, abgelegt. Objektreferenzen verweisen innerhalb der Regeln unter anderem auf Objekte der in der Abbildung dargestellten Klassen. Auf diese Weise sind zyklische Referenzen möglich, beispielsweise wenn eine Firewall eine Policy mit einer Regel besitzt, in

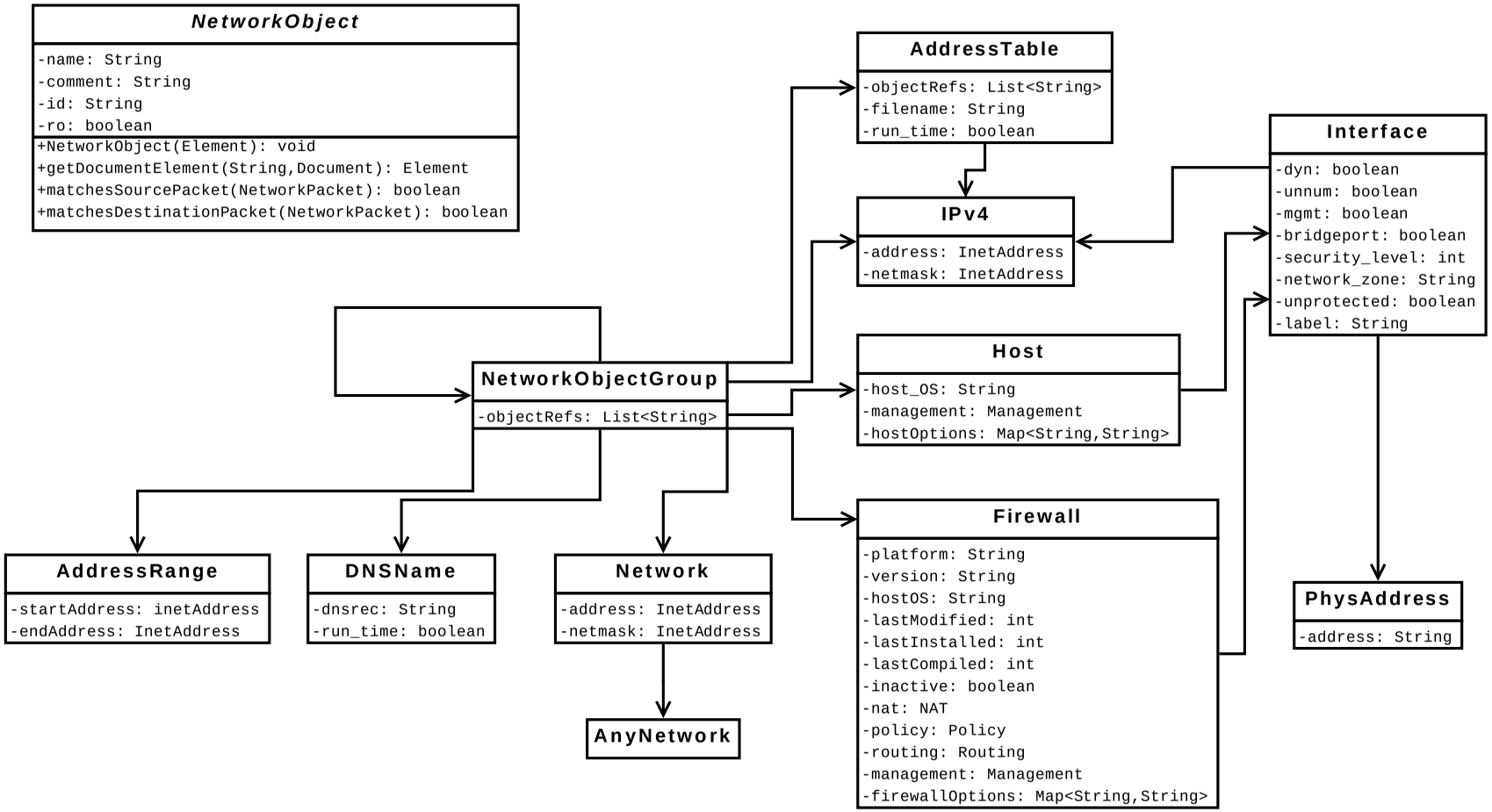


Abbildung 6.4: Aussschnitt der Datenstruktur, der Objekte im Netzwerk abbildet. Aus Gründen der Übersichtlichkeit wurde die Vererbung aller Klassen von der abstrakten Klasse `NetworkObject` nicht dargestellt.

der sie selbst referenziert ist. Dieses Problem musste beim Import entsprechend behandelt werden.

Da die Datenstruktur nicht nur zur Speicherung der Parameter sondern auch für die spätere Simulation verwendet werden soll, wurden entsprechende Methoden für die einzelnen Objekte implementiert. Eine Alternative zu diesem Vorgehen wäre eine separate Simulator-Klasse, welche mit Zugriff auf die Attribute der Objekte Entscheidungen trifft. Allerdings würden auf diese Weise Vorteile des objektorientierten Ansatzes ungenutzt bleiben. Für ein `NetworkObject` existieren daher besispielsweise die beiden Methoden `matchesSourcePacket` sowie `matchesDestinationPacket`, die für eine Komponente des Netzwerks ermitteln, ob es die Quelle oder das Ziel des simulierten Pakets darstellt. Die Komplexität der Implementierungen dieser Methoden reicht vom einfachen Zeichenkettenvergleich über Auswertungen von Netzwerkmasken bis hin zur Abfrage externer DNS-Server.

6.2.3 Import

Die Umwandlung von Regelsätzen des Firewall Builders in entsprechende SIBs ist Aufgabe des Imports. Um die Anzahl der zusätzlich benötigten Java-Bibliotheken dabei möglichst klein zu halten, wurde der bereits vom jABC benutzte Parser Xerces[Apache XML Project, 2008] verwendet. Allgemein lassen sich XML-Parser in zwei Kategorien aufteilen: Simple API for XML (SAX) und Document Object Model (DOM). Während SAX-Parser das gesamte Dokument sequentiell einliesen und während der Verarbeitung Ereignisse auslösen [Brownell, 2002], bauen DOM-Parser eine Datenstruktur in Baumform auf, in der die einzelnen Elemente separat behandelt werden können [W3C DOM Interest Group, 2005]. Xerces bietet für beide Arten eine Implementation, im Rahmen dieser Arbeit wurde der DOM-Parser eingesetzt.

Beim Einlesen durch die Klasse `DatabaseParser` wird daher im ersten Schritt aus der Quelldatei der DOM-Baum erzeugt. An der Wurzel steht – wie von der DTD des Firewall Builders vorgeschrieben – immer die `FWObjectDatabase`. Dieses Element wird der gleichnamigen Klasse im Konstruktor übergeben, wobei Kindelemente wiederum erkannt und mit deren entsprechenden Konstruktoren erzeugt werden. Nach Abschluss dieses rekursiven Prozesses enthält ein Objekt vom Typ `FWObjectDatabase` alle Informationen der XML-Datei in Form von Java-Klassen.

Dem Problem der zyklischen Referenzen begegnet der Import damit, dass er keinen Versuch unternimmt, die Referenzen aufzulösen. Ist ein Objekt kein Kind des Elternknotens, sondern nur über seine ID bekannt, so wird ausschließlich diese in der resultierenden Datenstruktur abgespeichert. Soll dann im späteren Programmverlauf auf das Objekt über seine ID zugegriffen werden, ermöglicht beispielsweise die Methode `getNetworkObject(String)` der Klasse `FWObjectDatabase` nach dem vollständigen Parsen das Holen der Objekte aus der Datenbank. Hierzu ist es jedoch erforderlich, dass die aktuelle Datenbank für alle Objekte des Plugins zugänglich ist. Im Anschluss an den erfolgreichen Import wird daher die resultierende `FWObjectDatabase` in einem statischen Feld der Klasse `FirewallPlugin` gespeichert. Jedes Objekt hat dann die Möglichkeit, diese Datenbank über eine öffentliche Methode abzurufen und entsprechende Anfragen nach referenzierten Objekten zu stellen.

6.2.3.1 Änderungen der importierten Daten

Auf eine fehlgeschlagene Überprüfung folgt meist die Änderung der betroffenen Regelsätze sowie eine erneute Simulation. Hilfreich wäre es an dieser Stelle, wenn der Benutzer das jABC nicht verlassen müsste, sondern Änderungen direkt an den importierten Daten durchführen kann. Um zu entscheiden, inwiefern dieses Ziel umgesetzt werden kann, mussten jedoch zuvor die entsprechenden Darstellungsmöglichkeiten des jABC näher untersucht werden.

Eine naheliegende Variante wäre an dieser Stelle der Einsatz eines speziellen Inspektors. Nach Auswahl einer Firewall-Regel könnte er beispielsweise alle Quellsysteme auflisten und eine Änderung ihrer Parameter erlauben. Allerdings unterstützt der Firewall Builder eine große Anzahl verschiedener Objekttypen mit jeweils unterschiedlichen Eigenschaften. Da bei einer solchen Umsetzung große Teile des Firewall Builders innerhalb des jABC nachgebildet werden müssten, wurde diese Idee nicht weiter verfolgt.

Die zweite untersuchte Variante bestand in dem Einsatz sogenannter *Hierarchien*. Mit diesen lassen sich mehrere Objekte innerhalb eines SIBs zusammenfassen und auf diese

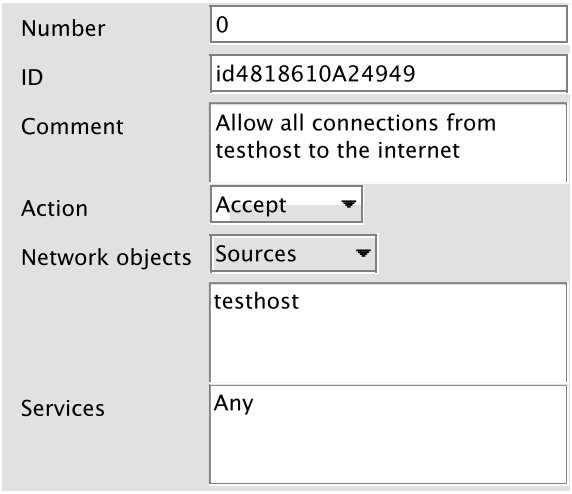


Abbildung 6.5: Der Inspektor für Regelsätze bietet einerseits eine grobe Darstellung einer Regel als auch eine Möglichkeit, die zu verwendende Aktion zu ändern.

Weise übersichtlich im Graph darstellen. So wäre es beispielsweise möglich, alle Teile einer Regel, wie sie in den Spalten des Firewall Builders definiert wurden, einzeln zu behandeln und auf der obersten Hierarchie als komplette Regel darzustellen. Das Problem an dieser Stelle besteht jedoch zum einen aus der Menge der zu implementierenden SIBs als auch – wie bereits bei der Lösung oben – der Entwicklung spezieller Inspektoren für jedes mögliche Objekt. Zwar unterstützt der mitgelieferte SIB-Inspektor bereits eine Reihe von Datentypen, die Adresse eines Netzwerkobjekts vom Typ `InetAddress` gehört jedoch beispielsweise nicht dazu.

Stattdessen ist der Anwender in der Lage, die importierten Regeln auf Ebene des Graphen zu manipulieren. Die möglichen Aktionen umfassen etwa eine Umsortierung und Entfernung von Regeln sowie die Verbindung mit anderen Aktionen, etwa *Accept* statt *Deny*. Letzteres ist sowohl über ein Umsetzen der Kanten als auch im Inspektor für Regelsätze (siehe Abbildung 6.5) möglich, der zudem auch eine Darstellung der wichtigsten Teile einer Regel bietet.

Damit der Graph nach den erfolgten Änderungen noch korrekt simuliert werden kann, sorgt das in Unterabschnitt 5.2.1 auf Seite 33 beschriebene Localchecker-Plugin für die Einhaltung festgelegter Regeln der verwendeten SIBs. Dieser prüft beispielsweise, dass jedes `PolicyRuleSIB` über mindestens eine eingehende Kante verfügt und die *match*- und *nomatch*-Kanten nicht ins Leere zeigen.

6.2.4 Simulation

Im Anschluss an den Import von Regelsätzen in das jABC kann eine Simulation erfolgen. Diese erfolgt wie bereits erwähnt mit Hilfe von vorher definierten virtuellen Testpaketen, deren Aufbau während der Entwicklung des Plugins festgelegt wurde.

6.2.4.1 Virtuelle Datenpakete

Für die Entscheidung eines Paketfilters, wie mit einem spezifischen Paket umgegangen werden soll, wird meist ausschließlich der Header ausgewertet. Aus diesem Grund beschränkt sich die erstellte Datenstruktur für virtuelle Pakete auf den Typ sowie gesetzte Felder im Header.

Als Datenformat wurde im ersten Schritt XML gewählt und eine entsprechende DTD erstellt. Der folgende Auszug definiert den Rahmen eines jeden Paketes.

```
<!ELEMENT NetworkPacket (EthernetOptions?,ICMPOptions?,
                         TCPOptions?,UDPOptions?,
                         IPOptions?,Date)>
<!ATTLIST NetworkPacket
    interfaceName       CDATA       #REQUIRED
    serviceType         CDATA       #REQUIRED
    expectedAction      %ACTION;    #IMPLIED
    matchedRuleNumber   CDATA       #IMPLIED
>
```

In dieser Datenstruktur sind alle notwendigen Informationen enthalten, über die auch der entscheidende Paketfilter verfügt. Neben den eigentlichen Paketdaten sind dies auch die Zeit der Ankunft sowie das Interface, auf welchem das Paket empfangen wurde. Damit nach einer Simulation entschieden werden kann, ob der Paketfilter korrekt arbeitet, existiert zusätzlich ein Feld für die ursprünglich erwartete Aktion.

Um die Verarbeitung der Paketdaten in Java zu ermöglichen, wurde parallel zur XML-Version die Klasse `NetworkPacket` mit identischen Eigenschaften entworfen. Mittels eines entsprechenden Konstruktors dieser Klasse kann das von Xerces geparste Element direkt bei der Erstellung eines entsprechenden Objektes benutzt werden. Eine Methode für den Export ermöglicht die Speicherung von Paketen im XML-Format, wenn diese beispielsweise aus einer anderen Quelle eingelesen wurden.

Während der eigentlichen Simulation werden die so importierten Pakete im Kontext der Regelsätze verarbeitet. Dabei muss die Reihenfolge der Pakete beibehalten werden, in der sie an einer Schnittstelle eintreffen würden, da ansonsten die Ergebnisse durch Funktionen wie das Stateful Filtering verfälscht werden könnten.

Import

Manuell erstellte Paketdaten in der oben beschriebenen XML-Syntax eignen sich in erster Linie für selbst definierte Tests, die ein Paketfilter absolvieren soll. Allerdings soll es auch möglich sein, reale Daten auszuwerten und innerhalb des Simulators zu benutzen.

Zum Mitschneiden des Netzwerkverkehrs eignet sich die Software tcpdump. Nach dem Aufruf versetzt sie die Netzwerkkarte in den sogenannten *promiscuous mode*, bei dem auch die Pakete aufgezeichnet werden, welche ursprünglich nicht an den betroffenen Rechner adressiert waren. Die Ausgabe von tcpdump enthält alle notwendigen Paketdaten, es fehlen jedoch der Empfangszeitpunkt sowie das betroffene Interface. Mit Hilfe der Klasse `TCPDumpParser` lässt sich diese Ausgabe in virtuelle Netzwerkpakete umwandeln.

Generierung

Für die Validierung komplexer Paketfilter reichen eine Handvoll Testpakete nicht aus. Zu leicht werden hier Spezialfälle nicht bedacht und kritische Fehler in der Umsetzung

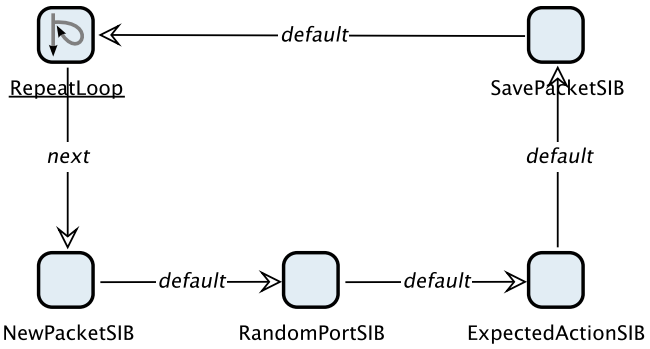


Abbildung 6.6: Ein aus SIBs bestehender Paketgenerator. Das `RepeatLoop`-SIB ermöglicht eine frei definierbare Anzahl von Durchläufen.

übersehen. Mit dem im vorigen Abschnitt beschriebenen tcpdump-Import lassen sich zwar eine große Anzahl Pakete automatisch einlesen, die Klassifizierung muss jedoch weiterhin von Hand geschehen. Im Gegensatz dazu wird die manuelle Erstellung mit einer XML-Datei schnell sehr aufwändig.

Abhilfe schafft hier ein im Rahmen dieser Diplomarbeit entwickelter Paketgenerator, der sich die Funktionalität des Tracer-Plugins zunutze macht. Mittels einzelner konfigurierbarer SIBs lässt sich ein Graph erstellen, der Pakete nach vorgegebenen Kriterien erzeugt. Im Einzelnen stehen dazu folgende SIBs zur Verfügung:

`NewPacketSIB`	Erstellung und Initialisierung eines Paketes vom Typ IP, TCP, UDP oder ICMP
`IPOptionsSIB`	Festlegung von IP-Optionen wie Quell- und Zieladresse
`TCPOptionsSIB`	Festlegung von TCP-Flags wie ACK oder Sync sowie Quell- und Zielport
`RandomPortSIB`	Zufällige Verteilung der Portnummern bei TCP-Paketen
`ExpectedActionSIB`	Definition der erwarteten simulierten Aktion für dieses Paket
`SavePacketSIB`	Hinzufügen des aktuell generierten Paketes zur Liste aller Testpakete

In Zusammenarbeit mit den *CommonSIBs*, die am selben Lehrstuhl wie das jABC entwickelt wurden und unter anderem mehrfache Durchläufe eines Graphen unterstützen, lassen sich so die gewünschten Pakete erzeugen und im Anschluss daran in eine XML-Datei exportieren. Abbildung 6.6 zeigt exemplarisch den Aufbau eines solchen Graphen.

Verwendung im Java Application Building Center

Bevor die virtuellen Pakete im Simulator verwendet werden können, müssen sie über das Plugin geladen werden. Ein eigens entworfener Inspektor, der in Abbildung 6.7 dargestellt ist, bietet für diese Aktion eine entsprechende Schaltfläche. Beim Einlesen wird der Benutzer nach der gewünschten Quelldatei gefragt und im Anschluss je nach Typ der entsprechende Parser aufgerufen. Wie schon beim Import der Regelsätze existiert in der Klasse `FirewallPlugin` eine statische Liste, in der die eingelesenen Daten abgelegt werden und so für alle Teile des Plugins zur Verfüfung stehen.

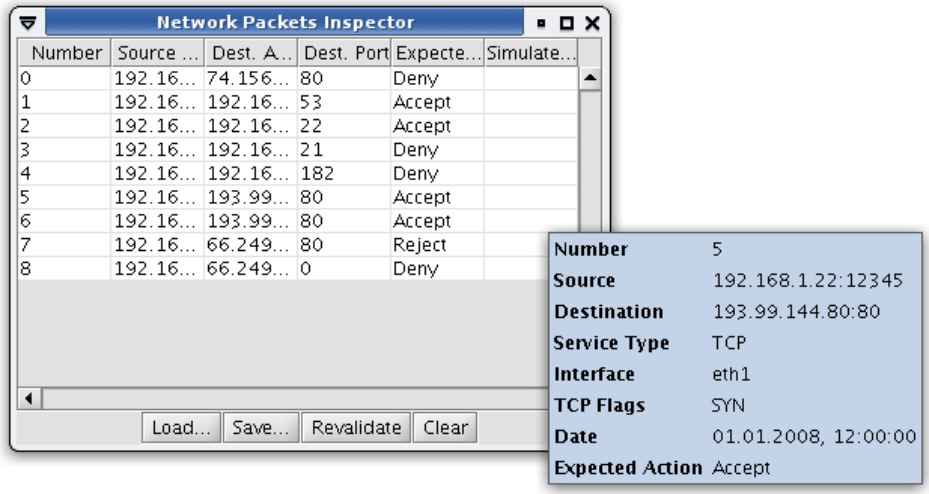


Abbildung 6.7: Inspektor für virtuelle Pakete nach dem Laden einer XML-Datei und vor der eigentlichen Simulation

Im restlichen Teil des Inspektors werden die Pakete in einer Tabellenstruktur dargestellt, wobei aus Gründen der Übersichtlichkeit eine Beschränkung auf die wichtigsten Daten erfolgt. Die Anzeige zusätzlicher Informationen wie der gesetzten IP-Flags erfolgt durch Tooltips beim Überfahren mit der Maus.

6.2.4.2 Simulator

Nach erfolgtem Import von Regelsätzen und virtuellen Paketen ist eine Simulation des Paketfilters möglich. Eine Betätigung der entsprechenden Schaltfläche im Inspektor der Netzwerkpakete startet einen Algorithmus mit folgendem Schema:

1. Erzeuge eine leere Datenbank zur Speicherung von Informationen über bestehende virtuelle Verbindungen (`StateDatabase`).
2. Führe folgende Schritte für alle virtuellen Pakete aus:
   a) Hole das nächste zu simulierende Paket aus der Liste aller Pakete in der Klasse `FirewallPlugin`.
   b) Sichere die aktuelle Datenbank zusammen mit dem Paket in derselben Klasse.
   c) Setze sowohl das Paket als auch die Datenbank in den *Context* des Tracer-Plugins.
   d) Führe einen Durchlauf des Tracer-Plugins aus und sichere die simulierte Aktion im virtuellen Paket.
3. Aktualisiere die Darstellung im Inspektor für Netzwerkpakete.

Der Grund für die Sicherung der aktuellen `StateDatabase` liegt in der Beschleunigung späterer Simulationen einzelner Pakete. Diese sollen im Anschluss an einen vollständigen Durchlauf mit Auswahl der entsprechenden Zeile in der Tabelle des Inspektors möglich sein.

Um nicht alle vorherigen Pakete erneut simulieren zu müssen, wird die Verbindungsdatenbank während des Gesamtdurchlaufs zwischengespeichert und dann bei Bedarf abgerufen.

Bei Ausführung des Tracers führt dieser für jedes SIB die `trace`-Methode aus, welche über die nächste zu besuchende Kante entscheidet. Kann ein solches SIB zwischen zwei möglichen ausgehenden Kanten wählen, so hängt die Aktion sowohl von den darin enthaltenen Daten als auch dem Inhalt des *Context*s ab. Im `StatefulFilterSIB` wird beispielsweise die aktuelle Verbindungsdatenbank daraufhin überprüft, ob die Kombination aus Quell- und Ziel-Socket des aktuellen Paketes dort verzeichnet oder aber Teil eines laufenden Verbindungsaufbaus ist. Ist dies der Fall, folgt der Tracer hier der *match*-Kante, die in diesem Fall auf ein `AcceptSIB` verweisen sollte.

Etwas komplexer gestaltet sich die entsprechende Methode der `PolicyRuleSIB`s, von denen jedes mit einer aus dem Firewall Builder importieren Regel verknüpft ist. Dabei handelt es sich im ein Objekt vom Typ `PolicyRule`, das über eine Methode mit der Signatur `matchesPacket(NetworkPacket)` verfügt. Diese Methode überprüft intern Parameter der Regel daraufhin, ob sie auf das zu simulierende Paket zutreffen. Ist dies der Fall, so gibt die aufrufende `trace`-Methode die *match*-Kante zurück, welcher das Tracer-Plugin zum nächsten SIB folgt. Im Normalfall zeigen *nomatch*-Kanten auf das `PolicyRuleSIB` der nächsten auszuwertenden Regel, während *match*-Kanten auf ein `ActionSIB` verweisen, das mit der auszuführenden Aktion verknüpft ist.

Endet die Ausführung einer Simulation an einem `ActionSIB`, so wird hier die entsprechende Aktion im virtuellen Paket eingetragen. Beim Erreichen eines `AcceptSIB`s wird zusätzlich die Verbindungsdatenbank aktualisiert. Bisher unbekannte Verbindungen werden hinzugefügt, während korrekt geschlossene Verbindungen entfernt werden.

Anhand der Darstellung im Inspektor kann schnell festgestellt werden, ob die Validierung erfolgreich war. Grün markierte Zeilen weisen auf eine Übereinstimmung von erwarteter und simulierter Aktion hin während eine rote Färbung das Gegenteil markiert. Unterscheiden sich diese Werte nur aus Sicht des Absenders – beispielsweise wenn die simulierte Aktion *Deny* statt *Reject* lautet – so kennzeichnet eine gelbe Färbung diesen Zustand.

Eine Auswahl der entsprechenden Zeile stellt wie bereits erwähnt den Verlauf des Paketes durch den Graphen mit Hilfe der zwischengespeicherten Verbindungsdatenbank dar. Auf diese Weise kann bei fehlerhaft zugeordneten Aktionen festgestellt werden, welche konkrete Regel dafür verantwortlich ist.

6.2.5 Funktionstests

Während der Entwicklung des Simulators wurde für einige fertiggestellte simulierte Funktionen ein Testfall erstellt, der bei späteren Änderungen verwendet werden konnte, um eine weiterhin bestehene Funktionsfähigkeit zu gewährleisten. Diese bestanden aus drei Elementen: Einer mit dem Firewall Builder erstellten XML-Datei, einer Definition von virtuellen Paketen sowie einer in Java geschriebenen Testklasse. Als Framework kam an dieser Stelle das häufig verwendete *JUnit* zum Einsatz.

Der trivialste Funktionstest prüfte beispielsweise anhand einer einzigen Regel, die lediglich aus der Anweisung „Erlaube alle Pakete“ bestand und diente nur der Kontrolle, ob der Simlator überhaupt Pakete richtig zuordnen kann. Etwas komplexer prüfte ein anderer Testfall, ob die Umsetzung des Stateful Filtering weiterhin korrekt funktioniert.

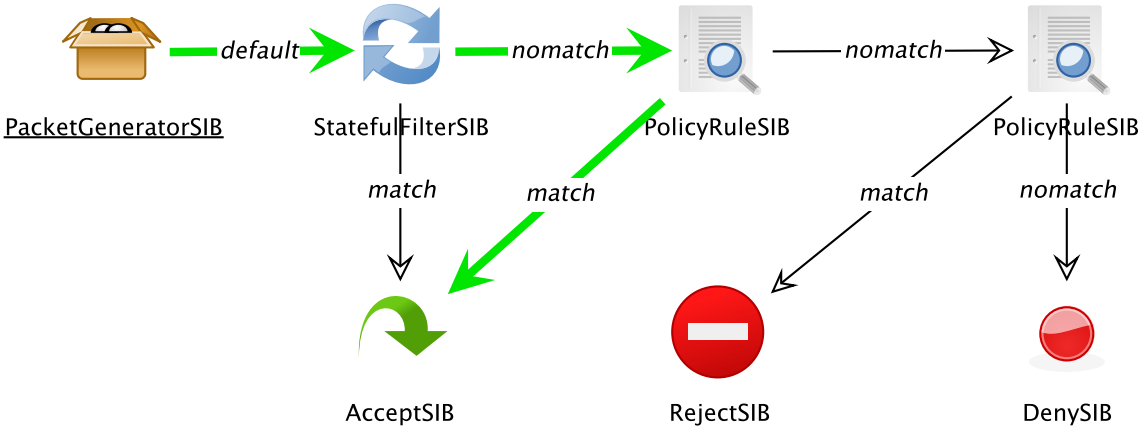


Abbildung 6.8: Grafische Darstellung des Testfalls für Stateful Filtering

Abbildung 6.8 zeigt den Graphen im jABC, der für diesen Test simuliert wurde. Die zugehörigen virtuellen Pakete validierten dabei folgende Aussagen, die für einen erfolgreichen Test allesamt bejaht werden mussten.

- Sind Verbindungen aus der Gegenrichtung vor Eröffnung der Verbindung verboten?
- Wird ein Paket, welches eine Verbindung aufbauen soll, akzeptiert?
- Darf die Gegenstelle auf dieses Paket antworten, obwohl noch keine Verbindung besteht?
- Wird das letzte Paket der Verbindungseröffnung akzeptiert?
- Sind Pakete aus beiden Richtungen im Kontext dieser Verbindung erlaubt?
- Sind weiterhin alle anderen Verbindungen aus der Gegenrichtung verboten?
- Kann die Verbindung korrekt geschlossen werden?
- Werden nach Beenden der Verbindung keine weiteren Pakete mehr zugelassen, die vorher der Verbindung zugeordnet werden konnten?

Zur Beantwortung dieser Fragen werden ähnlich wie bei einer normalen Simulation die errechnete Aktion mit der erwarteten, welche im Paket festgelegt ist, verglichen. Weichen diese Aktionen voneinander ab, so gilt der Test als nicht bestanden.

6.3 Zusammenfassung

Mit der Fertigstellung der einzelnen Komponenten und deren Zusammenschluss im Firewall-Plugin stehen nun die Funktionalitäten, die im Konzept zu Beginn dieses Kapitels gefordert wurden, zur Verfügung. Regelsätze lassen sich unabhängig vom gewählten Paketfilter in das jABC importieren und dort zusammen mit Testpaketen simulieren. Kann das Ergebnis die Erwartungen erfüllen, erzeugt der Firewall Builder die benötigten Konfigurationsdateien.

7 Praxisbeispiel

In diesem Kapitel wird die Verwendung des entwickelten Plugins anhand eines praxisnahen Beispiels demonstriert. Dazu werden anfangs die Struktur des abzusichernden Netzwerks vorgestellt, das Firewall-Konzept erläutert und die Umsetzung des Konzepts im Firewall Builder beschrieben. Es folgen die Generierung von Testpaketen, mit denen die Regelsätze validiert werden sowie der Import in das jABC. Eine Simulation der Daten liefert dann Rückschlüsse über das Verhalten der geplanten Firewall und ermöglicht eine Ausbesserung eventueller Fehler.

7.1 Beispiel-Netzwerk

Das fiktive Netzwerk dieses Beispiels besteht aus sechs Rechnern mit unterschiedlichen Aufgaben, die in Tabelle 7.1 aufgelistet sind. Abbildung 7.1 zeigt eine grafische Darstellung des Netzwerks.

Mit dem Internet selbst ist nur der Firewall-Rechner direkt verbunden, der den Namen *sonne* trägt. Er verfügt über drei Netzwerkschnittstellen, von denen zwei mit den internen Subnetzen kommunizieren. Alle drei Netzwerke sind ansonsten nicht miteinander verbdunden, *sonne* übernimmt somit die Funktion des Gateways, über den jeglicher Datenverkehr abgewickelt wird. Das installierte Betriebssystem ist Linux, als Paketfilter kommt Netfilter zum Einsatz.

Als Server im Subnetz mit der Adresse 129.217.42.0 fungieren die Rechner *neptun* (DNS), *jupiter* (Web) und *saturn* (Mail). Alle drei sollen Anfragen für ihre Dienste aus dem internen Netzwerk wie auch dem Internet beantworten können.

Die beiden Rechner *erde* und *mars* bezeichnen die Workstations, an denen Mitarbeiter sowohl die Dienste der Server nutzen können als auch nahezu unbeschränkten Zugang ins Internet genießen. Als Sicherheitsmaßnahme vor manipulierten DNS-Servern soll jedoch

Hostname	**Interface**	**Adresse(n)**	**Funktion**
	eth0	129.217.33.1	
sonne	eth1	129.217.42.1	Paketfilter
	eth2	129.217.43.1	
jupiter	eth0	129.217.42.50	Webserver
saturn	eth0	129.217.42.51	Mailserver
neptun	eth0	129.217.42.52	DNS-Server
erde	eth0	129.217.43.100	Workstation
mars	eth0	129.217.43.101	Workstation

Tabelle 7.1: Übersicht der Rechner des Beispiel-Netzwerkes und deren Funktion

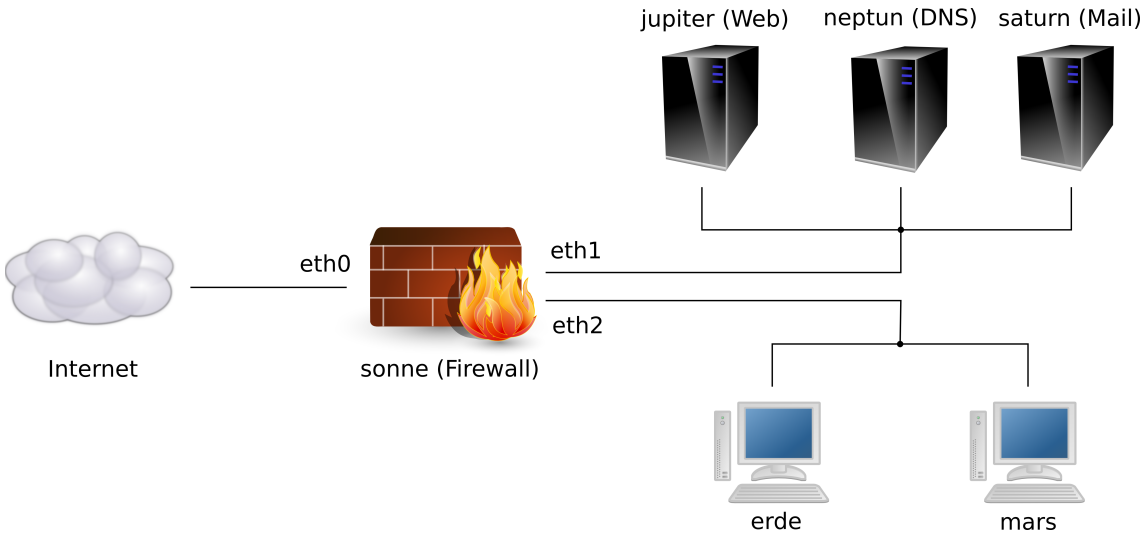


Abbildung 7.1: Schema des Beispiel-Netzwerks

ausschließlich *neptun* zur Namensauflösung benutzt werden. Zusätzlich soll von diesen Rechnern die Administration der Server über SSH möglich sein.

7.2 Konzept der Firewall

Nachdem der Aufgabenbereich der einzelnen Rechner festgelegt wurde, kann ein Konzept entworfen werden, das diese Aufgaben ermöglicht und zugleich das Netzwerk vor Angriffen schützt. Die entsprechenden Anforderungen lauten im Einzelnen:

- Schutz vor Spoofing-Attacken aus dem Internet
- DNS-Anfragen an *neptun* von außen
- DNS-Anfragen aus dem internen Netz nur an *neptun*
- Abruf der Webseiten von *jupiter* aus dem internen Netz und dem Internet
- Abruf und Versand von E-Mails an *saturn*
- Versand von E-Mails über *saturn* an Mailserver im Internet
- Zugriff der Workstations auf Dienste der Server und des Internets
- Blockierung aller nicht explizit zugelassenen Verbindungen

Tabelle 7.2 listet die aus diesen Anforderungen ermittelten Dienste beziehungsweise deren Anwendungsprotokolle, die jeder Rechner nach innen oder außen anbieten darf.

Hostname	Protokolle (intern)	Protokolle (extern)
sonne	SSH[1]	*keine*
jupiter	HTTP(S), SSH[1]	HTTP(S)
neptun	DOMAIN, SSH[1]	DOMAIN
saturn	IMAP(S), POP3(S), SMTP(S), SSH[1]	SMTP(S)
erde	*keine*	*keine*
mars	*keine*	*keine*

[1] Nur Workstations

Tabelle 7.2: Auflistung der erlaubten angebotenen Anwendungsprotokolle jedes Rechners

7.3 Umsetzung des Konzepts im Firewall Builder

Da alle angebotenen Dienste bereits im Firewall Builder vorkonfiguriert sind, ist nur die Erstellung der Objekte nötig, die Rechner im Netzwerk darstellen. Für eine bessere Übersicht und einfache spätere Wartbarkeit werden diese in drei Gruppen aufgeteilt: Die Server *jupiter*, *saturn* und *neptun* sowie die Firewall befinden sich in der Gruppe *Server* während *erde* und *mars* der Gruppe *Workstations* angehören. Diese beiden Gruppen bilden dann die dritte mit dem Namen *Interne Hosts*. Zuätzlich wird eine Gruppe *Mail-Protokolle* für Dienste angelegt, die neben dem Sendeprotokoll SMTP auch den Abruf über IMAP und POP3 – sowohl unverschlüsselt als auch verschlüsselt – berücksichtigt.

Im nächsten Schritt werden insgesamt zehn Regeln angelegt, über die das oben beschriebene Konzept umgesetzt wird und in Abbildung 7.2 auf der nächsten Seite gezeigt sind. Diese lauten im Einzelnen:

0. Verwirf und protokolliere Pakete, die scheinbar aus dem internen Netz kommen, jedoch über die Schnittstelle eth0 empfangen werden.
1. Erlaube jeglichen Datenverkehr über das lokale loopback-Interface.
2. Erlaube alle DNS-Anfragen an *neptun*.
3. Ermögliche alle DNS-Anfragen von *neptun* an andere Rechner.
4. Gewähre SSH-Zugang von den Workstations zu den Servern.
5. Erlaube jeden Zugriff über HTTP und HTTPS an *jupiter*.
6. Ermögliche das Senden und Holen von Mails über SMTP, IMAP und POP3 an *saturn*.
7. Damit *saturn* Mails nach außen verschicken kann, ermögliche einen Versand per SMTP an jeden Rechner.
8. Erlaube jeglichen Datenverkehr von den Workstations nach außen.
9. Blockiere jede nicht zutreffende Verbindung mit *ICMP port unreachable*.

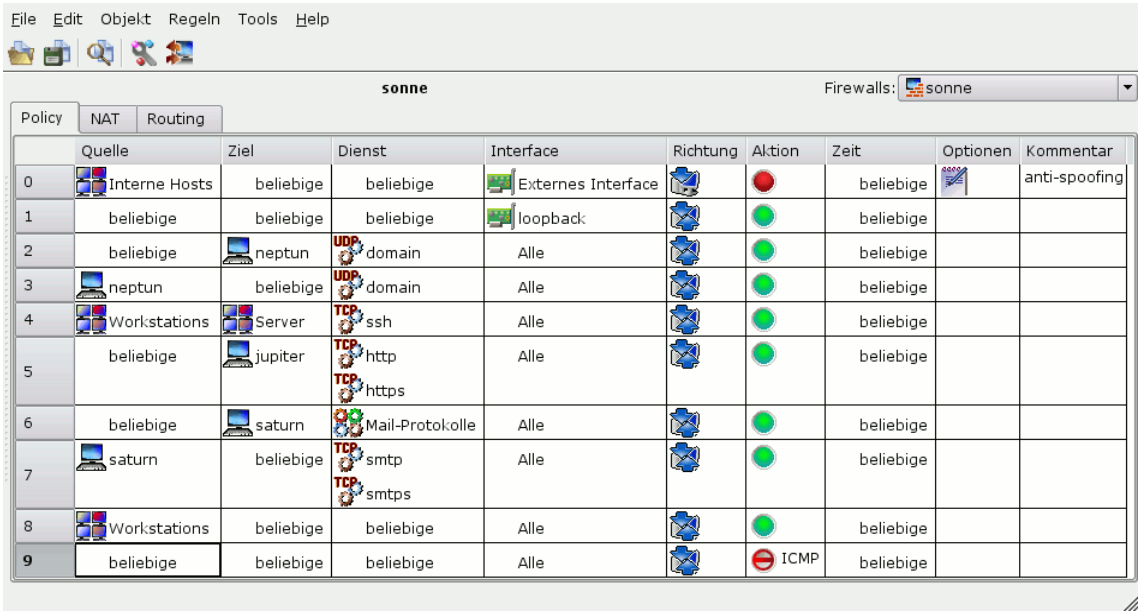


Abbildung 7.2: Hauptfenster des Firewall Builders mit dem Regelsatz der Firewall *sonne*

7.4 Erstellung von virtuellen Testpaketen

Für eine Simulation der zehn erstellten Regeln sind virtuelle Testpakete notwendig, mit denen die ursprünglichen Anforderungen möglichst genau geprüft werden können. Da der Paketfilter noch vor dem ersten Einsatz simuliert werden soll, stehen keine reellen Daten wie tcpdump-Ausgaben zur Verfügung. Tabelle 7.3 auf der nächsten Seite listet die Eigenschaften der Pakete auf, die aus diesem Grund manuell in Form einer XML-Datei spezifiziert sind. Für jedes Paket wurde zusätzlich die Anforderung vermerkt, die mit dem entsprechenden Paket validiert werden soll.

Die Anzahl der Pakete sollte zwar bei einem reellen, vollständigen Test durchaus höher sein, reicht jedoch zur Dokumentation im Rahmen dieses Beispiels aus.

Mit Hilfe der Pakete 1–13 wird die Kommunikation zu den erlaubten Diensten der Server überprüft. Dies umfasst etwa die DNS-Abfrage von *mars* an den internen DNS-Server oden der Zugriff auf den Webserver *jupiter* aus dem Internet. Eine vollständige Liste an Testpaketen könnte hier jede mögliche Kombination von zwei internen Rechnern sowie dem Internet für jeden betrachteten Dienst auflisten, worauf jedoch an dieser Stelle aus Gründen der Übersichtlichtkeit verzichtet wurde.

Die Regeln 14–18 hingegen beziehen sich auf den Datenverkehr von den Workstations ins Internet. Verbindungen sollen nur in eine Richtung aufgebaut werden dürfen, daher muss das Paket 14 im ersten Anlauf abgelehnt werden. Nachdem jedoch die Verbindung mit den Paketen 15–17 im Rahmen eines Drei-Wege-Handshake korrekt aufgebaut wurde, darf diesmal das vorher abgelehnte, zu Nummer 14 ähnliche Paket aufgrund des *Stateful Filtering* passieren.

Nr.	Typ	Interface	Quelle	Ziel	Aktion	Anforderung
0	TCP	eth0	129.217.43.100:24626	129.217.42.51:22	Deny	Anti-Spoofing
1	UDP	eth0	62.153.167.57:54619	129.217.42.52:53	Accept	DNS von außen an *neptun*
2	UDP	eth2	129.217.43.101:15736	129.217.42.52:53	Accept	DNS von innen an *neptun*
3	UDP	eth2	129.217.43.101:3672	208.67.222.222:53	Reject	DNS von innen an andere Server
4	UDP	eth1	129.217.42.52:2462	208.67.222.222:53	Accept	DNS von *neptun* an andere Server
5	TCP	eth2	129.217.43.100:46256	129.217.42.51:22	Accept	SSH von Workstations an Server
6	TCP	eth1	129.217.42.52:12631	129.217.42.51:22	Reject	SSH von Server an Server
7	TCP	eth0	57.146.83.147:34725	129.217.42.51:22	Reject	SSH von außen an Server
8	TCP	eth2	129.217.43.101:26824	129.217.42.50:443	Accept	HTTPS von Workstation an *jupiter*
9	TCP	eth0	48.146.251.14:42672	129.217.42.50:80	Accept	HTTP von außen an *jupiter*
10	TCP	eth0	215.124.51.35:46721	129.217.42.51:110	Accept	POP3 von außen an *saturn*
11	TCP	eth2	129.217.43.101:2862	129.217.42.51:143	Accept	IMAP von Workstation an *saturn*
12	TCP	eth2	129.217.43.101:52457	129.217.42.51:25	Accept	SMTP von Workstation an *saturn*
13	TCP	eth1	129.217.42.51:15773	217.72.192.157:25	Accept	SMTP von *saturn* an andere Server
14	TCP	eth0	216.34.181.45:80	129.217.43.101:24783	Reject	Verbindungen zu Workstations
15	TCP	eth2	129.217.43.101:24783	216.34.181.45:80	Accept	Verbindungen von Workstations
16	TCP	eth0	216.34.181.45:80	129.217.43.101:24783	Accept	Antworten auf Verbindungen
17	TCP	eth2	129.217.43.101:24783	216.34.181.45:80	Accept	Verbindungen von Workstations
18	TCP	eth0	216.34.181.45:80	129.217.43.101:24783	Accept	Verbindungen zu Workstations

Tabelle 7.3: Eigenschaften der virtuellen Pakete, mit denen die erstellten Regelsätze validiert werden sollen

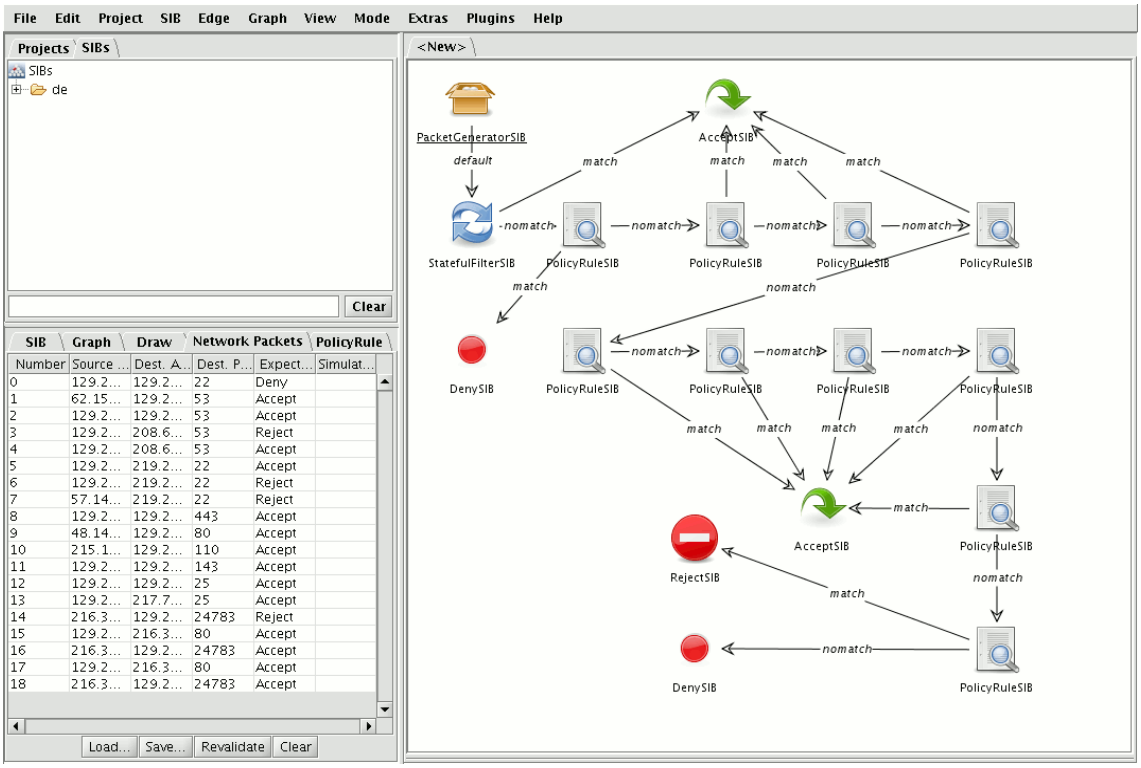


Abbildung 7.3: Die Ansicht des jABC nach Importierung der zuvor erstellten Datensätze

7.5 Import der Datensätze

Nachdem sowohl die Netzwerkstruktur im Firewall Builder als auch die virtuellen Pakete manuell im XML-Format erstellt wurden, müssen diese Datensätze zur Simulation mit Hilfe des jABC-Plugins importiert werden. Für beide Schritte existiert im Menü „Plugins“ des jABC ein entsprechender Punkt, der den Benutzer jeweils zur Angabe der einzulesenden Datei auffordert.

Im Falle der Daten des Firewall Builders erkundigt sich das Plugin weiterhin nach dem zu simulierenden Firewall-Rechner, sofern in der Datei mehrere spezifiziert sind. Daraufhin wird die XML-Datei importiert und der resultierende SIB-Graph im *Graph Canvas* dargestellt. Handelt es sich hingegen um die Daten der virtuellen Pakete, so erfolgt deren Darstellung im Inspektor mit dem Titel „Network Packets“. Abbildung 7.3 zeigt die resultierende Ansicht des jABC nach dem Import der vorher erstellten Daten.

7.6 Verifikation des importierten Graphen

Während des Imports wurden die SIBs automatisch mit atomaren Eigenschaften versehen. So erhielt jede Regel die Eigenschaft `Rule` und jede Aktion die Eigenschaft `Action`. Der Paketgenerator erhielt als Markierung für den Ursprung jedes Pfades die Eigenschaft `Start`.

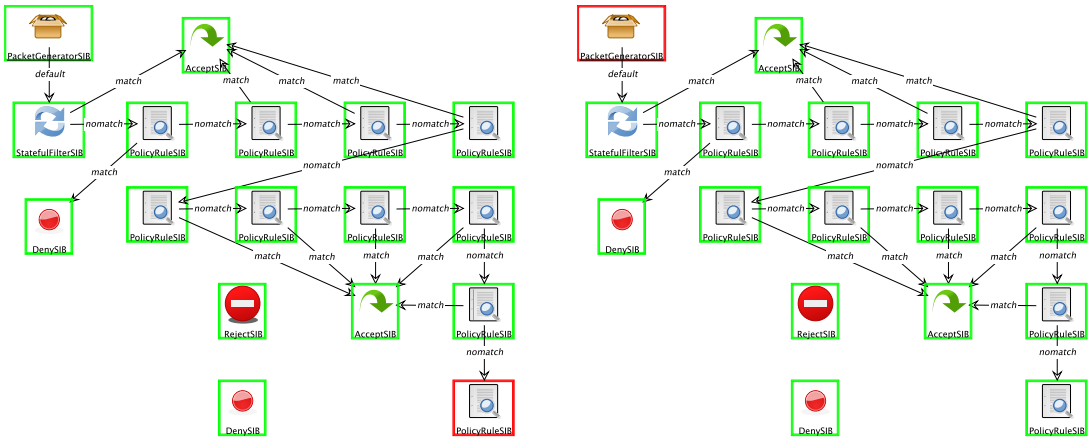


Abbildung 7.4: Verifikation des importierten Graphen mit dem Model-Checker-Plugin GEAR; Links der konstruierte Fehler, rechts das Ergebnis der Formel zur Suche des fehlerhaften SIBs

Zusätzlich enthält das Modell bereits eine vorbereitete Formel. Diese verifiziert, ob für jedes ankommende Paket eine Aktion ermittelt wird. Somit wird ausgeschlossen, dass die Verarbeitung etwa bei einer Regel ohne ausgehende Kanten endet oder einen Kreis durchläuft. Diese Spezifikation wird durch die temporallogische Formel

$$Start \Rightarrow AF(Action)$$

ausgedrückt. Direkt nach dem Import der obigen Daten ausgeführt markiert das Plugin alle SIBs des Graphen mit einem grünen Rahmen. Dies symbolisiert, dass das Modell die genannte Spezifikation erfüllt.

Um ein Beispiel zu konstruieren, bei dem mit Hilfe von GEAR ein Fehler entdeckt werden kann, wurden im Folgenden die beiden Kanten des letzten `PolicyRuleSIB`s entfernt. Das Ergebnis des Model Checkers ist in Abbildung 7.4 zu sehen: Da vom Paketgenerator aus nicht mehr in jedem Fall eine Aktion erreicht werden kann, erhält dieser einen roten Rahmen.

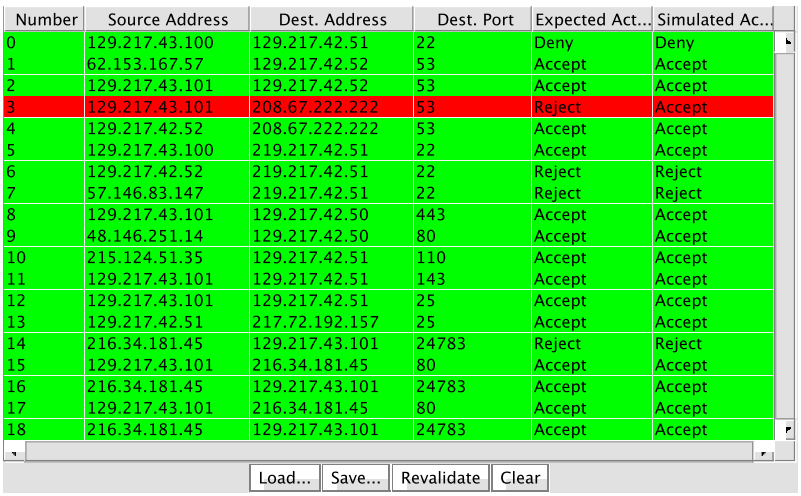
Eine weitere Formel eignet sich, um den Fehler im Modell zu finden. So muss es für jeden Regelknoten einen Pfad geben, der irgendwann zu einer Aktion führt. Temporallogisch lässt sich diese Aussage mit

$$Rule \Rightarrow AF(Action)$$

formulieren. Der rechte Teil der Abbildung 7.4 zeigt, dass einzig die letzte Regel diese Forderung nicht erfüllt.

7.7 Simulation

Im Anschluss an den Import der Datensätze kann die eigentliche Simulation gestartet werden. Dies geschieht mit Auswahl der Schaltfläche „Revalidate“ im Inspektor der Netzwerk-



Number	Source Address	Dest. Address	Dest. Port	Expected Act...	Simulated Ac...
0	129.217.43.100	129.217.42.51	22	Deny	Deny
1	62.153.167.57	129.217.42.52	53	Accept	Accept
2	129.217.43.101	129.217.42.52	53	Accept	Accept
3	129.217.43.101	208.67.222.222	53	Reject	Accept
4	129.217.42.52	208.67.222.222	53	Accept	Accept
5	129.217.43.100	219.217.42.51	22	Accept	Accept
6	129.217.42.52	219.217.42.51	22	Reject	Reject
7	57.146.83.147	219.217.42.51	22	Reject	Reject
8	129.217.43.101	129.217.42.50	443	Accept	Accept
9	48.146.251.14	129.217.42.50	80	Accept	Accept
10	215.124.51.35	129.217.42.51	110	Accept	Accept
11	129.217.43.101	129.217.42.51	143	Accept	Accept
12	129.217.43.101	129.217.42.51	25	Accept	Accept
13	129.217.42.51	217.72.192.157	25	Accept	Accept
14	216.34.181.45	129.217.43.101	24783	Reject	Reject
15	129.217.43.101	216.34.181.45	80	Accept	Accept
16	216.34.181.45	129.217.43.101	24783	Accept	Accept
17	129.217.43.101	216.34.181.45	80	Accept	Accept
18	216.34.181.45	129.217.43.101	24783	Accept	Accept

Abbildung 7.5: Ergebnis der Simulation im Inspektor für Netzwerkpakete

pakete. Im Hintergrund werden nun alle Pakete nacheinander simuliert und als Ergebnis in der Ansicht des Inspektors farbig dargestellt. Hierbei exisieren drei Abstufungen:

- Grüne Einträge zeigen eine Übereinstimmung des erwarteten und des simulierten Verhaltens.
- Gelbe Einträge lassen eine Abweichung im simulierten Verhalten erkennen, die jedoch für den Empfänger des Paketes keinen Unterschied macht (beispielsweise *Deny* statt *Reject*).
- Rote Einträge zeigen eine gravierende Abweichung im simulierten Verhalten und sollten daher genauer untersucht werden.

Das Ergebnis der Simulation im Rahmen dieses Beispiels ist in Abbildung 7.5 dargestellt. Abgesehen von einer Ausnahme wurden alle Pakete korrekt kategorisiert. Allerdings widerspricht in der vierten Zeile die simulierte Aktion *Reject* dem ursprünglich erwarteten *Accept*.

Ein Blick in Tabelle 7.3 auf Seite 55 verrät, dass es sich um die DNS-Anfrage eines internen Rechners an einen externen DNS-Server handelt. Diese sollten ursprünglich unterbunden werden, um das Unterschieben falscher Namensauflösungen zu verhindern. Eine Auswahl des entsprechenden Paketes im Inspektor (siehe Abbildung 7.6 auf der nächsten Seite) verrät jedoch, weshalb dieser Zugriff nicht wie erwartet blockiert wurde: Die vorletzte Regel erlaubt Workstations den Zugriff auf alle Dienste des Internets und somit auch Verbindungen mit externen DNS-Servern.

Dieser Fehler entstand während der Umsetzung des Sicherheitskonzeptes im Firewall Builder. Zwar wurde der DNS-Zugriff an keiner Stelle explizit zugelassen, allerdings wurde übersehen, dass er ebenfalls von einer anderen Regel mit eingeschlossen wird. Aus diesem Grund ist zur Korrektur eine separate Regel notwendig, die vorher eingefügt werden muss. Der folgende Ausschnitt zeigt den überarbeiteten Regelsatz:

7. Damit *saturn* Mails nach außen verschicken kann, ermögliche einen Versand per SMTP an jeden Rechner.

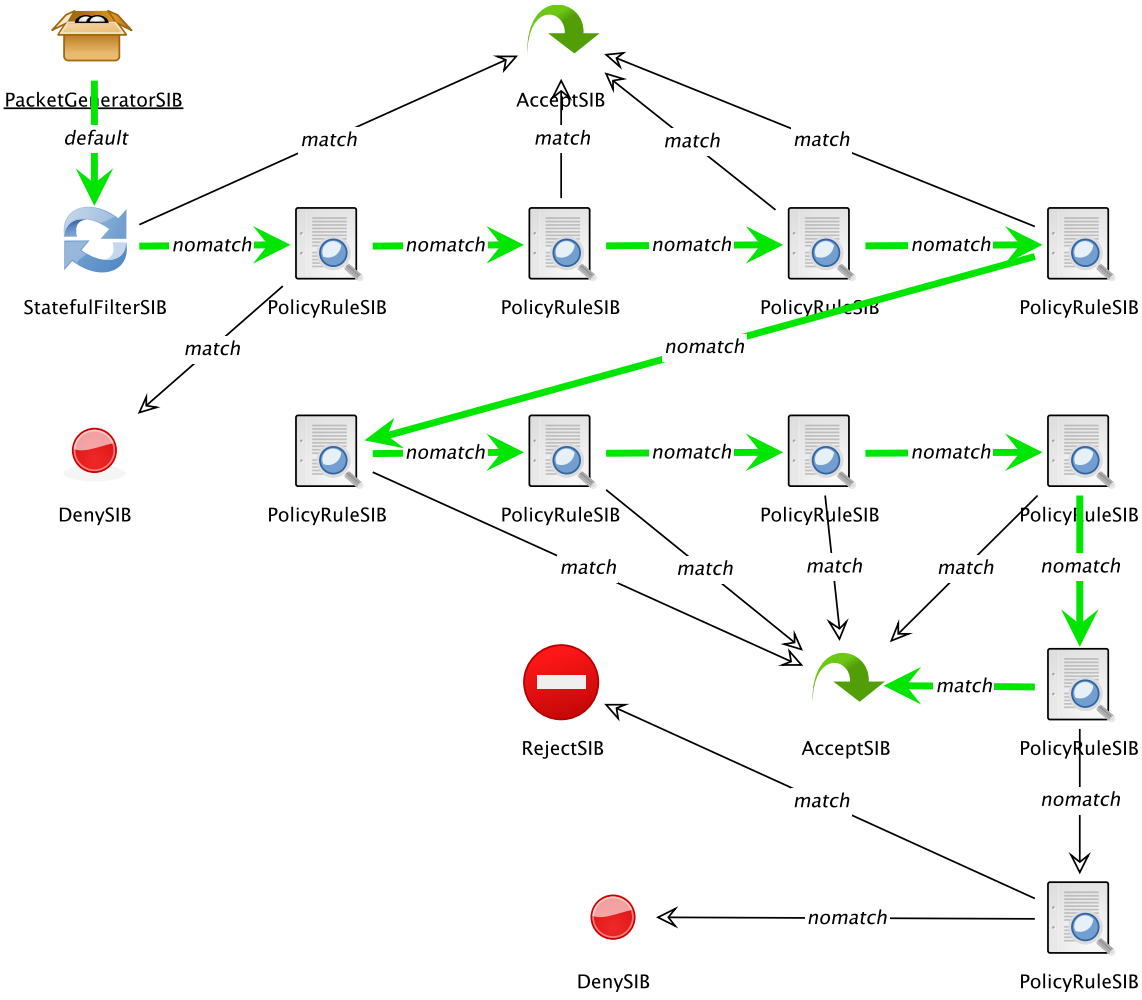


Abbildung 7.6: Einzelne Simulation des als fehlerhaft markierten Paketes

8. Verbiete DNS-Anfragen von internen Rechnern an externe DNS-Server.
9. Erlaube jeglichen Datenverkehr von den Workstations nach außen.
10. Blockiere jede nicht zutreffende Verbindung mit *ICMP port unreachable*.

Nachdem diese Änderungen mit dem Firewall Builder eingepflegt sind, erfolgt eine erneute Validierung der Datensätze. Im Falle eines positiven Ergebnisses kann nun die Konfigurationsdatei im Firewall Builder automatisiert erstellt und auf dem Firewall-Rechner installiert werden.

7.8 Ergebnis

Obwohl das vorgestellte Beispiel nur auf wenigen Regeln beruht, so lässt sich dennoch das in dieser Arbeit vorgestellte Konzept nachvollziehen. Mit Hilfe des Firewall Builders wurden die initialen Sicherheitsforderungen umgesetzt und im jABC grafisch dargestellt. Nach der Definition von ausgewählten virtuellen Paketen, welche die Anforderungen einzeln überprüfen sollen, wurden die Datensätze mit dem entwickelten Plugin geladen und eine Simulation gestartet.

Ein während der Simulation aufgetretener Fehler konnte in der grafischen Ansicht nachvollzogen und die Ursache ermittelt werden. Nach einer Änderung der Daten im Firewall Builder konnte dieser Fehler behoben werden, noch bevor die Konfiguration des Firewall-Rechner abgeschlossen wurde.

8 Schlussbemerkungen

Dieses Kapitel beginnt mit einem Ausblick auf mögliche zukünftige Erweiterungen, mit denen der Funktionsumfang des entworfenen Plugins noch vergrößert werden könnte. Abschließend werden die in der Anforderungsanalyse gestellten Teilaufgaben aufgegriffen und das Ergebnis ihrer Entwicklung zusammengefasst.

8.1 Ausblick

Während der Entwicklung des Plugins wurden mögliche Erweiterungen der Software deutlich. Diese konnten zwar im Rahmen dieser Arbeit nicht umgesetzt werden, liefern jedoch Material für zukünftige Entwicklungen.

8.1.1 Änderung und Export der Regelsätze

Obwohl der Firewall Builder bereits eine vollständige Umgebung für die Bearbeitung der Firewall-Regeln bereitstellt, wäre eine direkte Anpassung der Parameter im jABC komfortabler. Im Abschnitt 6.2.3.1 auf Seite 43 wurde erläutert, dass für eine vollständige Umsetzung jedoch große Teile des Firewall Builders innerhalb des jABC neu implementiert werden müssten. Allerdings birgt der Ansatz der *Hierarchien* das Potenzial, die Ergebnisse der Simulation noch aussagekräftiger zu machen. So könnte ein Nutzer nicht nur sehen, welche Regeln ein Paket passiert, sondern aus welchem Grund es mit den Parametern einer bestimmten Regel zusammenpasst.

Mit der Ermöglichung komplexer Änderungen wird zusätzlich ein Export dieser Datenstrukturen notwendig. Die notwendigen Java-Methoden wurden bereits im Rahmen des Imports teilweise implementiert und müssten daher bei Bedarf nur in die Oberläche des Plugins eingebaut werden.

8.1.2 Zeitnahe Analyse des Netzwerkverkehrs

Virtuelle Netzwerkpakete können derzeit in verschiedenen Datenformaten importiert werden. Dabei handelt es sich jedoch entweder um künstlich erzeugte Daten oder aber Mitschnitte des Datenverkehrs, die möglicherweise schon einige Zeit zurückliegen.

Hilfreich wäre an dieser Stelle ein Monitoring des aktuellen Datenverkehrs auf den Filterregeln. Dazu müsste das jABC direkt mit den Daten der Netzwerkschnittstelle versorgt werden. Datenpakete könnten auf diese Weise „live“ beobachtet und das Verhalten des Paketfilters analysiert werden.

Weitere Möglichkeiten ergeben sich an dieser Stelle für Techniken des *maschinellen Lernens*. Eine entsprechende Komponente könnte den Datenverkehr kontrollieren, auf diese Weise unbekannte Angriffsmuster erkennen und entsprechend reagieren.

8.1.3 Erzeugung von Konfigurationsdateien für andere Paketfilter

Der Firewall Builder unterstützt derzeit die Übersetzung von Regelsätzen in die Syntax verschiedenster Paketfilter. Allerdings ist diese Liste nicht vollständig, vor allem in Hinblick auf kommerzielle Produkte. Eine Überlegung wäre daher, diese Funktion auf der grafischen Ebene des jABC nachzurüsten.

Für diesen Zweck böte sich das von Sven Jörges entwickelte Code-Generator-Plugin *Genesys* an. Es handelt sich dabei um eine Erweiterung, mit der Quellcode für Hochsprachen aus SIB-Graphen erzeugt werden kann[Jörges u. a., 2007].

8.2 Zusammenfassung

Mit Rückblick auf den zu Beginn vorgestellten Anforderungskatalog wurden folgende Funktionalitäten im Rahmen dieser Arbeit umgesetzt.

Importierung von Regelsätzen
Der erste Berührungspunkt mit den zu verarbeitenden Datensätzen besteht aus dem Import der XML-Dateien, die vom Firewall Builder erzeugt werden. Dieser Vorgang wurde mit Hilfe einer Java-Datenstruktur sowie einem entsprechenden Parser umgesetzt.

Durch die Wahl des Firewall Builders und des jABC als verarbeitende Komponenten ist die Platformunabhängigkeit gewährleistet.

Grafische Darstellung der importierten Regeln
Eine eigens erzeugte Palette von SIBs übernimmt nach erfolgreichem Import die Darstellung der Firewall-Regeln. Zusammenhänge können direkt im jABC analysiert und kleine Änderungen auch dort vorgenommen werden. Da das jABC bereits Funktionen zur Darstellung von Graphen bietet, konnten diese übernommen mussten diese nicht erneut implementiert werden.

Erzeugung von virtuellen Paketen
Die für eine Simulation erforderlichen Testpakete können auf verschiedene Arten erzeugt und dann mittels des Plugins eingelesen werden. Derzeit unterstützt sind das Einlesen von entsprechenden XML-Dateien, tcpdump-Mitschnitten sowie die automatisierte Erstellung mit Hilfe des Tracer-Plugins.

Innerhalb des jABC übernimmt ein spezieller Inspektor die Darstellung der Paketdaten. Über ihn lassen sich ebenfalls Simulationen starten und deren Ergebnisse auswerten.

Simulation der Regelsätze mit Hilfe der virtuellen Pakete
Nach dem Einlesen der zu validierenden Datensätze kann im jABC die Simulation gestartet werden. Eine farbige Markierung im Inspektor der Netzwerkpakete gibt einen schnellen Überblick, welche Tests möglicherweise fehlgeschlagen sind. Durch die nachträgliche Simulation einzelner Pakete und deren grafischer Darstellung ist eine weitergehende Analyse möglich.

Verifikation der Ergebnisse
Anhand der Ergebnisse, welche die Simulation geliefert hat, ist eine Einschätzung der Qualität der Umsetzung des Paketfilters ohne dessen vorherige Installation möglich.

Literaturverzeichnis

[Apache XML Project 2008] Apache XML Project: *Xerces2 Java Parser Readme*. `http://xerces.apache.org/xerces2-j/`. Version: 2008

[Bakera u. a. 2007] Bakera, Marco ; Margaria, Tiziana ; Renner, Clemens D. ; Steffen, Bernhard: Verification, Diagnosis and Adaptation: Tool supported enhancement of the model-driven verification process. In: *ISoLA*, 2007, S. 85–97

[Barisani 2007] Barisani, Andrea: *FTester – Firewall and IDS Testing tool*. `http://dev.inversepath.com/trac/ftester`. Version: Juni 2007

[Brownell 2002] Brownell, David: *SAX2*. O'Reilly Media, 2002. – ISBN 0596002378

[Doedt 2006] Doedt, Markus: *Erweiterung der jABC-Framework Bibliothek um eine modular anpassbare Ausführungsschicht*, Universität Dortmund, Diplomarbeit, Mai 2006

[Donnerhacke 2008] Donnerhacke, Lutz: de.comp.security.firewall FAQ – Ist REJECT oder DENY sinnvoller? (2008). `http://www.iks-jena.de/mitarb/lutz/usenet/Firewall.html#Deny`

[Eddy 2007] Eddy, W.: *TCP SYN Flooding Attacks and Common Mitigations*. RFC 4987 (Informational). `http://www.ietf.org/rfc/rfc4987.txt`. Version: August 2007 (Request for Comments)

[Ferguson u. Senie 2000] Ferguson, P. ; Senie, D.: *Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing*. RFC 2827 (Best Current Practice). `http://www.ietf.org/rfc/rfc2827.txt`. Version: Mai 2000 (Request for Comments). – Aktualisiert durch RFC 3704

[Hansteen 2008] Hansteen, Peter N. M.: *Firewalling with OpenBSD's PF packet filter*. `http://home.nuug.no/~peter/pf/en/pf-firewall.pdf`. Version: 2008

[Hinden u. Deering 2006] Hinden, R. ; Deering, S.: *IP Version 6 Addressing Architecture*. RFC 4291 (Draft Standard). `http://www.ietf.org/rfc/rfc4291.txt`. Version: Februar 2006 (Request for Comments)

[Jörges u. a. 2007] Jörges, Sven ; Kubczak, Christian ; Pageau, Felix ; Margaria, Tiziana: Model Driven Design of Reliable Robot Control Programs Using the jABC. In: *EASE '07: Proceedings of the Fourth IEEE International Workshop on Engineering of Autonomic and Autonomous Systems*. Washington, DC, USA : IEEE Computer Society, 2007. – ISBN 0–7695–2809–0, S. 137–148

[Neubauer 2008] Neubauer, Johannes: *jABC Plugins – LocalChecker*. `http://jabc.cs.tu-dortmund.de/opencms/en/plugins/local_checker.html`. Version: März 2008

[OpenBSD project 2008] OpenBSD project: *OpenBSD Homepage*. http://openbsd.org/. Version: 2008

[Postel 1981a] Postel, J.: *Internet Control Message Protocol*. RFC 792 (Standard). http://www.ietf.org/rfc/rfc792.txt. Version: September 1981 (Request for Comments). – Aktualisiert durch RFCs 950, 4884

[Postel 1981b] Postel, J.: *Internet Protocol*. RFC 791 (Standard). http://www.ietf.org/rfc/rfc791.txt. Version: September 1981 (Request for Comments). – Aktualisiert durch RFC 1349

[Postel 1981c] Postel, J.: *Transmission Control Protocol*. RFC 793 (Standard). http://www.ietf.org/rfc/rfc793.txt. Version: September 1981 (Request for Comments). – Aktualisiert durch RFC 3168

[Steffen u. a. 2006] Steffen, Bernhard ; Margaria, Tiziana ; Nagel, Ralf ; Jörges, Sven ; Kubczak, Christian: Model-Driven Development with the jABC. (2006), November

[W3C DOM Interest Group 2005] W3C DOM Interest Group: *Document Object Model (DOM)*. http://www.w3.org/DOM/. Version: Januar 2005

[Welte u. Ayuso 2008] Welte, Harald ; Ayuso, Pablo N.: *The netfilter.org project*. http://netfilter.org/. Version: 2008

[Wikipedia 2008] Wikipedia: *MAC-Filter*. http://de.wikipedia.org/w/index.php?title=MAC-Filter&oldid=42466134. Version: Februar 2008

[Ylonen u. Lonvick 2006] Ylonen, T. ; Lonvick, C.: *The Secure Shell (SSH) Protocol Architecture*. RFC 4251 (Proposed Standard). http://www.ietf.org/rfc/rfc4251.txt. Version: Januar 2006 (Request for Comments)

[Ziemba u. a. 1995] Ziemba, G. ; Reed, D. ; Traina, P.: *Security Considerations for IP Fragment Filtering*. RFC 1858 (Informational). http://www.ietf.org/rfc/rfc1858.txt. Version: Oktober 1995 (Request for Comments). – Aktualisiert durch RFC 3128