DIPLOMARBEIT

OPEN SOURCE SECURITY TOOLS

Ausgeführt zum Zweck der Erlangung des akademischen Grades eines
Diplom-Ingenieurs (FH) für Telekommunikation und Medien
am Fachhochschul-Diplomstudiengang Telekommunikation und Medien St. Pölten

unter der Leitung von:
Dipl.-HTL-Ing. Andreas Schaupp MSc MAS

Zweitbegutachtung:
FH-Prof. Dipl.-Ing. Georg Barta

ausgeführt von:

Peter Brachtl
tm021015

Wien, im September 2006

Ehrenwörtliche Erklärung

Ich versichere, dass

- ich diese Diplomarbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfe bedient habe.
- ich dieses Diplomarbeitsthema bisher weder im Inland noch im Ausland einem Begutachter/einer Begutachterin zur Beurteilung oder in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

Diese Arbeit stimmt mit der von den Begutachtern beurteilten Arbeit überein.

.....	.....
Ort, Datum	Unterschrift

Kurzfassung

Sicherheit in IT-Systemen hat in der heutigen Zeit einen stetig wachsenden Stellenwert (*„When we face a choice between adding features and resolving security issues, we need to choose security“*, (Gates 2002)). So ist es für Unternehmen heute unumgänglich, ein gewisses Maß an Sicherheit herzustellen, um Informationssysteme und Netzwerke gegen unbefugte Zugriffe zu schützen. Die Implementierung kommerzieller IT Security Lösungen kann jedoch für Unternehmen nicht unbeträchtliche Kosten mit sich bringen.
Ziel dieser Arbeit war es nun, festzustellen, ob sich Sicherheitslösungen realisieren lassen, die rein auf Open Source Software basieren, und ob diese eine Alternative zu kommerziellen Security Produkten bieten können. Zusätzlich sollte noch erörtert werden, ob der Aufwand, der betrieben werden muss, um eine reine Open Source Lösung zu implementieren, den von kommerziellen Lösungen übersteigt.

Zur Beantwortung dieser Fragen war es zuerst notwendig, die theoretischen Grundlagen von Open Source und Security zu erforschen, um den Rahmen zu definieren, in dem sich die Arbeit bewegt. Dazu wurden die geschichtlichen Hintergründe von Open Source, die Definition von Open Source, sowie verschiedene Lizenzen beleuchtet. Anschließend wurde die Theorie der Informationssicherheit untersucht, sowie die grundlegenden Prinzipien von Security und die Ziele der IT Sicherheit analysiert.
Nach Erarbeitung der theoretischen Grundlagen befasste sich die Arbeit mit der Analyse vorhandener Open Source Security Programme aus verschiedensten Kategorien. Es wurden Produkte aus den Bereichen Firewalls, Intrusion Detection Systeme, Port Scanner, Vulnerabiliy Scanner, Anti-Viren Software sowie Verschlüsselungstechnologien herangezogen. Ziel dieses Vorgehens war es, festzustellen, welche Produkte verfügbar sind, und in welchen Bereichen Defizite bestanden, um anschließend aus jedem Bereich stellvertretend ein Programm jeder Kategorie für die praktische Implementierung auszuwählen. Allerdings zeige die Analyse, dass sich die Verfügbarkeit der Produkte der verschiedenen Bereiche stark unterscheidet. Während eine große Zahl an Firewall Lösungen zur Verfügung standen, war es nicht möglich, mehr als nur einen vollwertigen Vertreter zu finden.
Der letzte Teil der Arbeit widmete sich der praktischen Implementierung der zuvor analysierten Security Anwendungen. Ziel war es, ein rein auf Open Source Software basierendes System aufzubauen, das dennoch einen mit kommerziellen Lösungen vergleichbaren Sicherheitsstandard bieten kann.

Die im Zuge der Arbeit gewonnen Erkenntnisse führten zu dem Schluss, dass der Aufbau eines Sicherheitssystems unter ausschließlicher Zuhilfenahme von Open Source Software möglich ist, und dass die meisten der implementierten Programme einen sehr hohen Standard sowie umfangreiche Einsatzmöglichkeiten bieten. Ferner hat sich der Aufwand, der zur Planung und Installation eines solchen Sicherheitssytems notwendig ist, als nicht signifikant höher herausgestellt, als dies bei einem kommerziellen System der Fall wäre.

Abstract

Security in information systems is among the most important issues in corporate security. For corporations it has become inevitable to implement security measures in order to protect information systems and networks against unauthorized access and unwanted tampering.

The motivation behind this thesis was to research, whether it was possible to build open source based security systems, that are able to compete with their commercial counterparts. In addition, the research aimed at determining, if the work put into building such a system exceeds the effort needed for building a commercial system, hence canceling out the financial advantage.
The first step in the course of the research was to give an overview of the theoretical background of open source and the basic aspects of security. Secondly, open source products of all major areas of security software have been analyzed, reaching from firewalls to encryption tools, in order to determine the spread of open source software. Additionally, one product of each group was chosen to be implemented in the practical part of the thesis. In order to determine the usefulness of the different kinds of tools a security system has been built and tested for different vulnerabilities.

The research done in the course of the thesis has led to the conclusion, that it is possible to build a secure system, based solely on open source software.

Inhaltsverzeichnis

1 Einleitung

Informationssicherheit in vernetzen Systemen hat in den letzten Jahren enorm an Bedeutung gewonnen. Durch das ständige Wachsen des Internets und der immer stärkeren Verbreitung von IT Systemen ist es heutzutage unabdingbar, Rechner sowie Netze entsprechend abzusichern. (vgl. Howlett 2004)
Um eine möglichst gute Absicherung gegenüber unbefugten Zugriffen zu gewährleisten, ist es jedoch notwendig, vernetzte Systeme auf mehreren Ebenen mit Hilfe verschiedenster Tools zu schützen. Dazu zählen zum Beispiel Firewalls, Virenscanner oder Intrusion Detection Systeme sowie auch Sicherheitsmaßnahmen auf Betriebsystemebene.
Eine Implementierung der genannten Maßnahmen kann für ein Unternehmen jedoch zu erheblichen Kosten führen, wenn auf kommerzielle Sicherheitslösungen zurückgegriffen wird. Alternativ hierzu können Open Source Produkte eingesetzt werden. Diese sind oft kostenlos erhältlich und gerade im Bereich des Internets besonders präsent. So gibt es eine Vielzahl an nicht kommerziellen Security Produkten, die unter Open Source Lizenzen herausgegeben werden. Die Handhabung solcher Tools ist jedoch nicht immer so einfach und komfortabel, wie man dies von vielen kommerziellen Produkten gewohnt ist. Oft ist es für den User auch notwendig, sich wesentlich tiefgehender mit der Materie auseinanderzusetzen, um einen reibungslosen Betrieb der Software zu ermöglichen. Dies könnte für viele Benutzer eine Barriere darstellen.

1.1 Ziele und Fragestellung

Ziel dieser Arbeit soll es nun sein, vorhandene Open Source Security Produkte zu analysieren, und festzustellen, ob diese von ihrem Leistungsumfang geeignet sind, kommerzielle Produkte vollständig zu ersetzen und somit eine kostengünstige Alternative zu bieten. Des weiteren gilt es zu erörtern, ob der allfällige Mehraufwand zum Betrieb dieser Produkte die Kostenersparnis wert ist.

1.2 Abgrenzungen

Der Begriff der IT Security ist ein sehr weit gefächerter. So zählen zum Beispiel auch Maßnahmen wie Zugangskontrollen zu den Räumlichkeiten, in denen sich vernetzte Systeme befinden, zum Bereich der Informationssicherheit. Da sich diese Arbeit jedoch mit Software Security Tools befasst, werden diese *physischen Sicherheitsmaßnahmen* nicht behandelt.
Weiters wird sich diese Arbeit ausschließlich mit Software Produkten für das Betriebsystem Linux auseinandersetzen. Zwar gibt es auch für andere Betriebsysteme (z.B. für Windows) Open Source Applikationen, jedoch sollen in dieser Arbeit nur *reine* Open Source Lösungen analysiert werden. Dies inkludiert auch das Betriebsystem.

1.3 Aufbau der Arbeit

Im folgenden Kapitel soll zunächst eine allgemeine Einführung in die Materie gegeben werden. Dabei wird zuerst der Begriff *Open Source* definiert, sowie kurz auf die entstehungsgeschichtlichen Hintergründe und die heutige Situation von Open Source Software eingegangen. Des Weiteren wird ein Überblick über die verschiedenen Open Source Lizenzen und deren Inhalte gegeben.
Anschließend befasst sich das Kapitel mit den Grundlagen der Informationssicherheit. Es soll vermittelt werden, was unter dem Begriff *IT Security* zu verstehen ist, und welche Teilbereiche für diese Arbeit relevant sind.

Kapitel 3 befasst sich zuerst mit grundlegenden Sicherheitsmaßnahmen auf Betriebsystemebene, und zeigt, warum ein robustes Betriebsystem als Grundlage für weitere Security Tools notwendig ist.
Anschließend wird ein Überblick über verschiedenste Open Source Security Tools gegeben. Behandelt werden Firewalls, Intrusion Detection Systeme, Port Scanner und Vulnerability Scanner, Anti-Viren Software sowie Verschlüsselungstechnologien.

In Kapitel 4 werden die in Kapitel 3 behandelten Applikationen in einer praktischen Teststellung implementiert und auf ihre Fähigkeiten hin analysiert. Anschließend wird ein Penetrationstest durchgeführt und die gesammelten Daten ausgewertet.

Das letzte Kapitel widmet sich der Zusammenfassung der Arbeit sowie der Beantwortung der Forschungsfragen.

2 Einführung in Open Source Security

Ziel dieser Arbeit ist es, einen Überblick über vorhandene Open Source Security Produkte zu geben und diese auf ihre Tauglichkeit als Alternative zu kommerziellen Security Produkten hin zu analysieren. Hierzu ist es jedoch zuerst notwendig, den Forschungsgegenstand zu definieren, diesen in einen entsprechenden Kontext zu setzen und Abgrenzungen zu schaffen.
So soll im folgenden Kapitel einleitend erklärt werden, worum es sich bei *Open Source* und *Security* eigentlich handelt, und welche Bereiche für diese Arbeit relevant sind.
Im ersten Abschnitt wird die Entstehung und Bedeutung von Open Source untersucht und anschließend auf die verschiedenen Software Lizenzen eingegangen.
Der zweite Teil beschäftigt sich mit dem Begriff Security, und soll die grundlegenden Konzepte von IT Security ergründen.

2.1 Open Source

> *„Free software is a matter of liberty, not price. To understand the concept, you should think of 'free' as in 'free speech', not 'free beer'."*
>
> – Richard Stallman

Der Begriff Open Source sollte hinlänglich bekannt sein. Jedoch ist Vielen oft nicht klar, worum es sich dabei überhaupt handelt, und was sich hinter diesen beiden Worten verbirgt. So ist *Open Source* oder *Free* Software keinesfalls zwingend kostenlos. Das *Free* bzw. *Open* bezieht sich vielmehr auf die freie Zugänglichkeit zum Quellcode sowie auf die Möglichkeit, diesen verändern und weitergeben zu dürfen, und hat mit *Freeware* (gratis Software) nichts zu tun.
Um verständlich zu machen, was Open Source Software ist, und wofür es steht, soll an dieser Stelle zuerst ein Überblick über die Entstehungsgeschichte gegeben werden, um ein besseres Verständnis für die Ideen hinter Open Source Software zu schaffen.
Anschließend werden die zwei wichtigsten Open Source Software Lizenzen (die GNU GPL sowie die BSD Lizenz) analysiert und miteinander verglichen.

2.1.1 Entstehung und Bedeutung

Die Wurzeln von Open Source Software liegen bereits in den späten 60er Jahren des letzten Jahrhunderts, bei UNIX. Von den AT&T Bell Labs entwickelt, wurde es, aufgrund von Regulativen, die AT&T daran hinderten, UNIX kommerziell zu vermarkten, an Universitäten weitergegeben. Dort wurde begonnen, den Code zu modifizieren. Dies ging an manchen Universitäten so weit, dass vollkommen eigenständige Systeme entstanden. So zum Beispiel an der University of California, Berkeley (BSD).

Im Jahre 1984 rief ein Programmierer vom MIT, Richard Stallman, das GNU Projekt ins Leben, mit dem Ziel, ein Unix ähnliches Betriebsystem zu schaffen, das ausschließlich aus freier Software bestand. GNU ist ein rekursives Akronym und bedeutet GNU's Not UNIX. Damit soll einerseits die Distanzierung des GNU Projektes von UNIX unterstrichen und andererseits gleichzeitig eine gewisse Verwandtschaft suggeriert werden. 1985 gründet Stallman die Free Software Foundation (FSF), weil er mit dem damaligen Trend, dass immer weniger Firmen die Quellcodes ihrer Software veröffentlichen, unzufrieden war. Die FSF fordert, dass Software prinzipiell frei sein soll, und entwickelte zu diesem Zwecke die GNU General Public License (GPL), auf die später in diesem Kapitel noch genauer eingegangen wird. Aufgrund der GPL, und anderen, ähnlichen Lizenzen, wurde es für Programmierer möglich, freie Software zu entwickeln, ohne sich Sorgen um ihr geistiges Eigentum machen zu müssen. Zu einem signifikanten Wachstum der Open Source Community kam es jedoch erst Anfang der 1990er Jahre, als es, durch die immer weiter steigende Popularität des Internet, immer einfacher möglich wurde, Code untereinander auszutauschen und zu veröffentlichen.

Mitunter der wichtigste Wendepunkt für die Entwicklung der Open Source Bewegung war zweifellos Linux. Im Jahre 1991 von dem finnischen Studenten Linus Torvalds entwickelt, wurde, was eigentlich als Hobby anfing, schnell zum wichtigsten Open Source Betriebsystem aller Zeiten. Der Grund für die Entwicklung von Linux war ursprünglich Torvalds Unzufriedenheit mit Andrew Tanenbaums *Minix*, eine vereinfachte Unix Version für PCs. Innerhalb kürzester Zeit nach der Veröffentlichung des ersten Kernels entwickelte sich Linux durch die Mitarbeit vieler tausender Programmierer zu einem mächtigen und vollwertigem Betriebssystem. Linux ist ein Musterbeispiel dafür, was durch die Ideologie, die hinter Open Source Software steht, erreicht werden kann.

Erst 1998 wurde der Terminus Open Source schließlich eingeführt, unter anderem um die falsche Assoziation mit dem Wort „*Free*“ zu vermeiden.

Vorteile von Open Source

- **Kosten** - Auch wenn Open Source Software nicht immer zwingend gratis ist, so ist dies doch meist der Fall. Der Einsatz von Open Source Komponenten kann einem Unternehmen unter Umständen eine nicht unbeträchtliche Menge Geld sparen.
- **Erweiterbarkeit** - Open Source ist der Inbegriff von Erweiterbarkeit. Vorausgesetzt man besitzt die nötigen Programmierfähigkeiten, ist es ganz einfach möglich, ein Programm genau an die spezifischen Bedürfnisse eines Unternehmens anzupassen.
- **Sicherheit** - Im Unterschied zu proprietärer Software ist es bei Open Source Programmen nicht möglich, Sicherheit durch die Geheimhaltung des Quelltextes zu erlangen. Dies kann dazu führen, dass von vorn herein sicherere Software geschrieben wird, und Sicherheit durch andere, wirksamere Methoden erreicht wird.
- **Unabhängigkeit** - Sicherheitslücken können viel schneller gefunden und behoben werden, wenn nicht nur eine Firma, sondern viele tausend User Einblick in den Source Code haben.
- **User Support** - Auch wenn es für viele Open Source Produkte keine offiziellen Support Hotlines gibt, so stehen oft riesige Netzwerke von Benutzern und Entwicklern in Form von Mailing Lists, Foren oder IRC Channels frei zur Verfügung, die nicht selten schnelleren und besseren Support liefern können, als die Telefonisten eines Call Centers.

- **Produktlebensdauer** - Bei kommerzieller Software kann es durchaus vorkommen, dass eine Firma entscheidet, ein Produkt nicht mehr weiterzuführen. Ein Unternehmen, dass diese Software nutzt, verliert dann oft jede Möglichkeit, Upgrades oder Support zu bekommen. Open Source Projekte hingegen sterben nie endgültig. Es wird immer Benutzer geben, die eine Software noch verwenden oder diese gar weiterentwickeln.

Heute ist Open Source Software wichtiger und größer denn je. Vor allem im Server Bereich und im Bereich des Internet sind Open Source Produkte wie Linux, das heute von über 50 Millionen Benutzern weltweit eingesetzt wird, und Apache, der auf mehr als 70% aller Webserver verwendet wird [Apache Website], heute nicht mehr wegzudenken.

2.1.2 Open Source Lizenzen

Wenn von Open Source Software gesprochen wird, wird oft angenommen, es handle sich um freie, kostenlos verfügbare Software, die keinerlei Restriktionen unterliegt.
Tatsächlich jedoch muss Software, die Open Source ist, unter einer Lizenz herausgegeben werden, die ganz bestimmten Richtlinien entspricht. Die *Open Source Initiative* (`http://www.opensource.org`) hat eine Liste mit Anforderungen herausgegeben (*Open Source Definition*), welchen eine Lizenz genügen muss, um als Open Source Lizenz anerkannt zu werden:

1. **Free Redistribution** - Die Software darf frei weitergegeben oder verkauft werden
2. **Source Code** - Die Software muss den Quellcode entweder beinhalten oder dieser muss frei erhältlich sein
3. **Derived Works** - Die Verbreitung von modifizierter Software muss erlaubt sein
4. **Integrity of Author's Source Code** - Die Lizenz kann fordern, dass Veränderungen nur als Patches weitergegeben werden
5. **No Discrimination Against Persons or Groups** - Die Lizenz darf niemanden benachteiligen
6. **No Discrimination Against Fields of Endeavor** - Die Lizenz darf den Einsatzbereich der Software nicht einschränken
7. **Distribution of License** - Die Rechte an einem Stück Software müssen auf alle Personen übergehen, die diese erhalten, ohne dafür eine eigene Lizenz erwerben zu müssen
8. **License Must Not Be Specific to a Product** - Die Rechte an einem Programm dürfen nicht davon abhängig gemacht werden, ob es Teil eines bestimmten Paketes ist.
9. **License Must Not Restrict Other Software** - Die Lizenz darf keine Einschränkungen bezüglich anderer Software, die mit der lizenzierten Software weitergeben wird, enthalten.
10. **License must be Technology-Neutral** - Keine Bestimmung der Lizenz darf auf eine bestimmte Technologie oder ein Interface bestehen.

(vgl. Perens 1997)

Zwar sind Open Source Lizenzen meist nicht so restriktiv wie herkömmliche Closed Source Lizenzen, nichts desto trotz gibt es auch bei Open Source Lizenzen Restriktionen, die vor allem dem Schutz der Programmierer und deren Arbeiten dienen.
Natürlich sind diese Einschränkungen nicht immer dieselben, und so gibt es auch viele verschiedene Open Source Lizenzen. Mit den zwei wichtigsten Vertretern (GNU GPL und BSD-Lizenz) werden sich die nächsten beiden Abschnitte auseinandersetzen.

GNU GPL

Die GNU General Public License (GNU GPL) wird von der *Free Software Foundation* (FSF) herausgegeben, und ist die am meisten verbreitete Lizenz für Open Source Software. Sie wurde ursprünglich von Richard Stallman für Programme, die im Zuge des GNU Project veröffentlicht wurden, entwickelt. GPL Version 1.0 wurde 1989 verfasst, Version 2.0 folgte 1991. Seit 2005 wird an der dritten Version der GPL gearbeitet, die 2007 fertig gestellt werden soll.
Die GNU GPL kann ausnahmslos von jedem verwendet werden, solange sie wörtlich übernommen und nicht verändert wird. Im Vergleich zur BSD-Lizenz ist die GNU GPL wesentlich komplexer und umfangreicher, und beinhaltet auch eine größere Anzahl an Restriktionen. Sie räumt jedem Benutzer das Recht ein, die lizenzierte Software für jeden beliebigen Zweck zu verwenden, garantiert freien Zugang zum Source Code und gibt das Recht, die Software zu verändern und frei zu verbreiten, unter folgenden Bedingungen:

- Der gesamte Text der GNU GPL muss der Software beiliegen
- Zugang zum Source Code muss auf einfache Weise möglich sein
- Wird ein Programm verändert und wird diese Veränderung veröffentlicht, so muss auch der modifizierte Source Code veröffentlicht werden. Wird ein Programm nur für den privaten Gebrauch modifiziert, so muss auch der neue Source Code nicht veröffentlicht werden
- Jedes Derivat eines Programms, welches unter der GNU GPL veröffentlicht wurde, muss auch selbst wieder unter der selben Lizenz herausgegeben werden.

Entgegen der verbreiteten Meinung, Open Source Software sei gratis, ist es sehr wohl möglich, mit Software, die unter der GNU GPL Lizenz herausgegeben wurde, Geld zu verdienen. So kann zum Beispiel für Verpackung, Handbuch, Trägermedien oder auch Support- und Garantiepakete ein Entgelt verlangt werden. (vgl. Free Software Foundation 1991)

Die GNU GPL in ihrer Gesamtheit ist in Anhang A nachzulesen.

BSD Lizenz

Die BSD-Lizenz wurde ursprünglich für BSD Unix geschrieben. BSD steht für Berkeley Software Distribution, und wurde an der University of California in Berkeley entwickelte. Durch ihre Einfachheit hat sie eine Vielzahl von anderen Lizenzen inspiriert.
Die BSD hat deutlich weniger Restriktionen als die GNU GPL, und kann eigentlich schon fast als *public domain* angesehen werden. Der größte Unterschied zur GNU GPL ist, dass die BSD Lizenz nicht verlangt, dass zukünftige Modifikationen einer Software unter derselben Lizenz veröffentlicht werden. Dies macht es möglich, ein vorhandenes Programm zu verändern und dieses dann unter einer kommerziellen, proprietären Lizenz herauszugeben.

So haben zum Beispiel eine nicht unbeträchtliche Zahl an Firmen kommerzielle Versionen von Unix auf den Markt gebracht, die auf dem ursprünglichen BSD Code basieren. (vgl. Wikipedia.org 2006a)

Die BSD-Lizenz in ihrer Gesamtheit ist in Anhang A nachzulesen.

2.2 Security

Der englische Begriff *Security* hat in unserem Sprachgebrauch viele Bedeutungen. Wird in dieser Arbeit von *Security* gesprochen, so ist damit, sofern nicht anders angemerkt, stets *Computer Security* oder *IT Security* gemeint. Die IT Security ist Teil der *Information Security*, die weit über *Information Technology* (IT) Systeme, Computer und Netzwerke hinausgeht.

IT Security befasst sich mit der Sicherheit von IT Systemen. Ihr Ziel ist es, eine sichere Umgebung zu schaffen, das bedeutet, ein System, das sich in einem autorisierten Zustand befindet, und nicht in einen nicht autorisierten Zustand übergehen kann. (vgl. Bishop 2003) Dies kann durch Maßnahmen wie Firewalls, Anti-Viren Software, Kryptographie oder Intrusion Detection Systeme erreicht werden.
IT Security ist aus dem heutigen Alltag nicht mehr wegzudenken. Durch den enormen Zuwachs an IT Systemen in den letzten Jahrzehnten in praktisch allen Bereichen unseres täglichen Lebens sowie durch die Verbreitung des Internet, ist es für Unternehmen unvermeidbar geworden, Sicherheitsmaßnahmen zu implementieren, um Informationssysteme zu schützen und das Unternehmen vor möglichen finanziellen Schäden zu bewahren. Hierbei ist allerdings auch zu beachten, dass die Usability (*dt. Benutzbarkeit*) eines Systems nicht unter verwendeten Sicherheitsvorkehrungen leidet. Gute Security muss ein System absichern, ohne dabei die Benutzbarkeit einzuschränken.

Im folgenden Abschnitt soll ein Überblick über die grundlegenden Prinzipien der Informationssicherheit gegeben werden. Anschließend werden mögliche Security Threats analysiert, und Gegenmaßnahmen erläutert.

2.2.1 Prinzipien

Wenngleich auch IT Security nur ein Teilbereich der Information Security ist, so sollen an dieser Stelle dennoch die der Information Security zugrunde liegenden Prinzipien erläutert werden, um ein besseres Verständnis für die Materie zu schaffen.
Information Security basiert im Grunde genommen auf Confidentiality (*dt. Vertraulichkeit*), Integrity (*dt. Integrität*) und Availability (*dt. Verfügbarkeit*)(CIA). Im Folgenden sollen diese Begriffe erläutert werden.

Confidentiality Confidentiality beschreibt die Geheimhaltung von Informationen vor gewissen Personen. Ursprünglich vom Militär motiviert, trifft dieses so genannte *„Need to Know“* Prinzip auch auf private Unternehmen zu, die zum Beispiel Pläne für Entwicklungen geheim halten wollen, um zu vermeiden, dass Unbefugte ihre Ideen stehlen. Zugriffskontrollmechanismen wie zum Beispiel Kryptographie sind Mittel, die dieses Prinzip unterstützen. So kann verschlüsselte Information dazu verwendet werden, um sicher zu gehen, dass nur diejenige Person Zugang zu sensiblen Daten erhält, für die diese auch gedacht waren. Auch können Passwörter oder ähnliche Kontrollmechanismen verwendet werden. Im Unterschied zur Kryptographie jedoch verlieren diese Mechanismen, wenn Sie versagen, jede Wirkung.

So eignen sich Passwörter zwar zum *verstecken* von Informationen, nicht jedoch dazu, um den *Inhalt* dieser Informationen zu verbergen. Fällt z.B. ein Passwort System aus, so liegen die versteckten Informationen plötzlich offen, und können von jedem gelesen werden, wenn sie nicht verschlüsselt sind.

Integrity Integrity bezieht sich auf die Vertrauenswürdigkeit von Daten, und darauf, zu verhindern, dass Daten von unautorisierten Quellen verändert werden. Man unterscheidet zwischen *Data Integrity*, wobei es um den Inhalt von Daten geht, und *Origin Integrity*, was die Quelle von Daten beschreibt (besser bekannt als *Authentication*). Weiters lassen sich Integrity Mechanismen in zwei Klassen unterteilen: Präventions- und Erkennungsmechanismen. Ersteres versucht, das unautorisierte Verändern von Daten zu verhindern, während Letzteres versucht, Veränderungen zu entdecken und zu melden.

Availability Availability bezieht sich auf die Verfügbarkeit von Informationen oder Diensten. Dies ist ein wichtiger Aspekt von Security, wenn es darum geht, dass jemand verhindern möchte, dass gewisse Informationen oder Dienste verfügbar sind. Solche Versuche werden *Denial of Service (DoS)* Attacken genannt, und sind im Security Bereich ein großen Problem. (vgl. Bishop 2003)

Um gute Security zu implementieren, ist es stets notwendig, all diesen Prinzipien Folge zu leisten.

2.2.2 Bedrohungen

Als Bedrohung (engl. *threat*) bezeichnet man eine potentielle Sicherheitsverletzung. Um von einem Threat zu sprechen, muss der tatsächliche Sicherheitsbruch noch nicht passiert sein. Die Möglichkeit, dass sich ein solcher Eingriff ereignen könnte, reicht dafür schon aus.
Es gibt eine Vielzahl von verschiedenen Bedrohungen, die einem Unternehmen, aber auch einer Privatperson, zum Verhängnis werden können. Wird versucht, eine solche Schwachstelle auszunutzen, so wird dies als Attacke (engl. attack), oder auch als Angriff bezeichnet. Die wichtigsten sollen im Folgenden kurz verdeutlicht werden.

Denial of Service (DoS) Ziel einer DoS Attacke ist es, das Opfer außer Gefecht zu setzen, sodass es ihm nicht mehr möglich ist, Anfragen anderer Clients zu beantworten. Erreicht wird dies, indem versucht wird, das angegriffene System so zu überlasten, dass eine oder mehrere Ressourcen verbraucht werden. Dabei kann es sich um Speicher, CPU Zeit, aber auch Bandbreite handeln. Durch die heutige Verbreitung von Breitbandanschlüssen ist letzteres die häufigste Form und wird durch so genannte DDos Attacken (Distributed Denial of Service) erreicht. Ein solcher Angriff wird von mehreren System gleichzeitig ausgeführt und wird meist mit Hilfe von Trojanern realisiert, durch die der Angreifer so genannte *Zombie Computers* (Computer, die unfreiwillig an einer Attacke teilnehmen) steuern kann. Durch DDos Attacken können enorme Bandbreiten erzielt werden, was dazu führt, das das Opfer keine Bandbreite mehr zur Verfügung hat, um reguläre Anfragen zu beantworten. DoS Attacken sind nicht primär dazu gedacht um Zugriff zu einem System zu erlangen, sondern um Systeme handlungsunfähig zu machen, und somit Schaden anzurichten. Es ist sehr schwierig, sich gegen DoS Angriffe zu schützen, da diese nicht immer von regulärem Datenverkehr zu unterscheiden sind.

Information Leakage Information Leakage passiert, wenn ein System Informationen preisgibt, die eigentlich nicht öffentlich sein sollten. Ein Beispiel dafür ist ein Webserver, der, wenn auf eine nicht vorhandene Seite zugegriffen wird, mit einer Fehlermeldung antwortet, die die Versionsnummer der Server Software beinhaltet. Dies könnte einem potentiellen Angreifer Informationen über das System geben, die dieser dann verwenden kann, um entsprechende *Exploits* dieser speziellen Version auszunutzen.

Exploits Unter Exploit versteht man eine Schwachstelle in einem Programm, die dazu genutzt werden kann, sich unbefugt Zugang zum System zu verschaffen, oder dieses zum Absturz zu bringen. Zwar durchläuft Software im Normalfall eine Anzahl von Tests, bevor sie veröffentlicht wird, jedoch finden sich immer wieder außergewöhnlichere Fehler, die erst im Nachhinein entdeckt werden, und somit ausgenutzt werden können.

Sniffing Als *Sniffing* oder *Snooping* bezeichnet man den Vorgang des Belauschens einer Kommunikation. Man unterscheidet zwischen passivem und aktivem Sniffing. Beim passiven Sniffing werden die Daten abgefangen, analysiert und unverändert an die ursprüngliche Ziel Destination weitergeleitet. Dies dient vor allem dazu, um an Informationen zu kommen, die unverschlüsselt übertragen werden. Beim aktiven Sniffing werden Daten nicht nur abgefangen, sondern auch verändert. Dies dient dazu, gezielt falsche Informationen in einen Kommunikationsweg einzuschleusen. Ein Beispiel für aktives Sniffing ist die *Man-in-the-Middle-Attacke*.

Social Engineering Ein Security System kann immer nur so stark sein wie der User, der es verwendet. Unter *Social Engineering* versteht man das Ausnützen von Unachtsamkeit, den Missbrauch von Vertrauen, sowie die gezielte Täuschung oder Irreführung von Individuen, um sich auf diesem Wege Zugang zu IT Systemen zu verschaffen. Ein Beispiel dafür wäre, wenn sich eine Person in einer Nachricht als Systemadministrator ausgibt, und den Benutzer auffordert, ihm sein Passwort zu übermitteln.

Selbstverständlich gibt es noch viele andere Threats (so gilt es beispielsweise auch als Threat, sich gewaltsam physischen Zugang zu einem IT System zu verschaffen), deren Analyse jedoch den Rahmen dieser Arbeit sprengen würden.
Für Unternehmen können diese Bedrohungen zu ernsthaften Problemen führen, die sich von Rufschädigung im besten Fall, im schlechtesten Fall jedoch bis hin zu finanziellen Verlusten bewegen können. Der folgenden Abschnitte sollen nun Mittel und Wege aufzeigen, wie man sich gegen solcherlei Bedrohungen schützen kann.

2.2.3 Policies und Mechanismen

Zum richtigen Verständnis von Security ist es notwendig, zwischen Security Policies und Security Mechanismen unterscheiden zu können.

Eine Security *Policy* ist ein Dokument, in dem der Sicherheitsanspruch eines Unternehmens festgehalten ist. In einer Policy werden sichere und nicht sichere Zustände definiert, sowie festgelegt, was erlaubt, und was nicht erlaubt sein soll.

Im Idealfall sollte eine Security Policy folgende Punkte enthalten:

- **Richtlinien** - Definieren die angestrebte Qualität der Sicherheit im Unternehmen.

- **Aufgaben** - Definiert Rollen und Zuständigkeiten im Unternehmen
- **Klassifizierung/Kontrolle unternehmenskritischer Daten** - Liefert eine Liste unternehmenskritischer Daten und der Maßnahmen zu ihrem Schutz
- **Mitarbeitersicherheit** - Definiert Erwartungen an Mitarbeiter bezüglich Sicherheit und Vertraulichkeit sowie die Rollen der Mitarbeiter
- **Physikalische Sicherheit** - Gerätesicherheit, Zugangsschutz und Kontrollmechanismen
- **Kommunikations- und Operationsmanagement** - Befasst sich mit dem Schutz und der Integrität von Informationen
- **Zugriffskontrolle** - Kontroll- und Überwachungsmaßnahmen für den Zugriff auf Netzwerke und Anwendungen sowie der Schutz vor Eindringlingen
- **Systementwicklung- und Wartung**
- **Kontinuitätsmanagement** - Befasst sich mit Maßnahmen bei schwerwiegenden Ausfällen und der Wiederherstellung nach Notfällen
- **Richtlinieneinhaltung** - Befasst sich mit der Prüfung von Sicherheitsrichtlinien und deren Umsetzung sowie mit der Definition von Audit-Prozessen.

(vgl. ISO 2005)

Security *Mechanismen* sind Methoden, um die Security Policy umzusetzen. Dazu zählen Zugangbeschränkungen zu Serverräumlichkeiten genauso wie Firewalls oder Dateizugriffsberechtigungen.

2.2.4 Ziele

Ziel von Security ist es, die in der Security Policy als *„sicher“* definierten Zustände zu erreichen. Die Methoden, die dazu verwendet werden, unterteilt man im Allgemeinen in drei Gruppen.

Prevention Methoden, die dazu führen sollen, dass ein Angriff verhindert wird, werden und dem Begriff *Prevention* zusammengefasst. Versucht zum Beispiel jemand, auf einen bestimmten Dienst zuzugreifen, jedoch wird dieser Dienst von einer Firewall geschützt, so ist die Firewall ein präventiver Mechanismus.

Detection Ist es nicht möglich, einen Angriff zu verhindern, so sind Detection Mechanismen hilfreich. Ihr Ziel ist es, zu erkennen, dass ein Angriff im Gange ist, und diesen zu melden. So werden Schäden zwar nicht verhindert, zumindest ist aber bekannt, dass ein Schaden eingetreten ist, und meist auch, welcher Schaden angerichtet wurde. Ein typisches Beispiel für einen Detection Mechanismus sind Intrusion Detection Systeme (IDS).

Response Hierbei wird festgelegt, wie das System auf einen Angriff reagieren soll. Dies bewegt sich von einfachen Upgrades oder wiederherstellen gelöschter Dateien bis hin zu Gegenangriffen oder dem Verständigen der entsprechenden Autoritäten. Es ist allerdings recht schwierig, effektive Response Mechanismen einzusetzen, da sich jeder Angriff von vorangegangenen unterscheidet.
(vgl. Bishop 2003)

Fazit

Security Soft- oder Hardware ist heute ein unverzichtbarer Teil eines Firmennetzwerks. Diese zu implementieren und zu optimieren kann jedoch zu einem erheblichen zeitlichen, aber auch finanziellen Aufwand führen. Werden Open Source Produkte eingesetzt, so kann zumindest der finanzielle Aufwand unter Umständen deutlich minimiert werden.
Das folgende Kapitel sollen nun einen Überblick über die Möglichkeiten geben, die mit Open Source Security Software zur Verfügung stehen.

3 Security Tools

Kapitel 2 beschäftigte sich mit den Grundlagen von Open Source und Security, mit dem Ziel, ein grundlegendes Verständnis für dieses Kapitel, das sich mit Open Source Security Tools beschäftigt, zu schaffen.

Security Tools sind Applikationen, die dazu dienen, IT Systeme sicherer gegenüber unbefugten Zugriffen zu machen. Dies kann auf vielen Wegen bewerkstelligt werden. Firewalls zum Beispiel sind so genannte „Frontline Defense“ Tools und zählen zu den präventiven Sicherheitsmethoden (siehe 2.2.4).
Intrusion Detection Systeme wiederum zählen zu den Detection Mechanismen, da es ihre Aufgabe ist, unbefugtes Eindringen zu erkennen und zu melden.
Natürlich zählen auch Anti-Viren Software, Verschlüsselungsmechanismen und diverse Scanner Tools zu den Security Applikationen.

Im folgenden Kapitel soll analysiert werden, welche Produkte der verschiedenen oben erwähnten Sicherheitsapplikationen, die unter Open Source Lizenzen herausgegeben werden, zur Verfügung stehen, und was diese leisten können. Zuvor werden noch grundlegende Möglichkeiten der Betriebsystemsicherheit analysiert, die notwendig sind, um eine sichere Umgebung für spätere Security Tools zu schaffen.

3.1 Betriebssystemsicherheit

Hauptaugenmerk dieser Arbeit ist die Analyse von Open Source Security Applikationen. Um jedoch sicherzustellen, dass diese auch ihren Zweck erfüllen, ist es zuerst notwendig, das Betriebssystem, das den Anwendungen zugrunde liegt, entsprechend abzusichern, da selbst die stärksten Mauern nutzlos sind, wenn sie auf Treibsand gebaut werden. Der folgende Abschnitt beschäftigt sich mit eben diesen grundlegenden Security Aspekten.

Viele Angriffe richten sich heutzutage auf das Betriebssystem. Da viele Betriebssysteme inzwischen sehr komplex geworden sind, gibt es kaum jemanden, der wirklich den Überblick über das gesamte System hat. So werden fast täglich neue Lücken gefunden, die von Angreifern ausgenützt werden können.
Ein weiteres Problem stellt sich dadurch, dass viele Systeme vom Hersteller mit Standardeinstellungen ausgeliefert werden, welche meist so gewählt sind, dass wesentlich mehr Dienste als notwendig geladen werden, um sicherzugehen, dass auch bei Benutzern, die nicht sehr versiert sind, alles *„Out of the box“* funktioniert. Diese Probleme treffen auf Linux genauso zu, wie auf Windows, oder die meisten anderen Betriebssysteme. So werden zum Beispiel bei einer Standardinstallation von RedHat Linux auch wesentlich mehr Dienste und Programme geladen, als der Durchschnittsuser benötigen würde.

Leider beschränken sich diese Probleme nicht nur auf Privatpersonen. Oft sind auch Server in Unternehmen schon auf Betriebssystemebene viel zu schlecht abgesichert, was Angreifern die Möglichkeit gibt, in das System einzudringen und Schaden anzurichten.

Deshalb ist es wichtig, zuerst sicherzustellen, dass das Betriebssystem, auf dem künftige Sicherheits-Software verwendet werden soll, auch selbst sicher ist. Den Vorgang des absichern des Betriebssystems nennt man *Hardening*.

Es gibt eine Vielzahl an Betriebssystemen, und es ist oft nicht leicht zu sagen, welches am sichersten ist. Da das Thema dieser Arbeit jedoch *Open Source* Security ist, wird sich auch das Thema Betriebssystemsicherheit mit einem Open Source Betriebssystem, Linux, befassen.

3.1.1 Hardening

Der Begriff *Hardening* bezeichnet den Vorgang der Absicherung eines Systems. Bei Betriebssystemen fallen darunter zum Beispiel Methoden, wie das Entfernen unnötiger Benutzerkonten, oder das Deaktivieren nicht verwendeter Dienste.
Unter Linux gibt es eine Vielzahl an Methoden, mit denen das System auf einfache Art und Weise sicherer gemacht werden kann, und die Möglichkeit unbefugten Eindringens somit verringert werden kann.

Wenngleich Sicherheitssysteme meist durch Firewalls nach außen hin abgeschottet werden, sollten dennoch Maßnahmen getroffen werden, um das System auch ohne Firewall so sicher wie möglich zu gestalten. Firewalls sind so genannte *Front Line Defense* Security Tools. Sie sind die erste Verteidigunglinie gegen Angriffe von außen. Ist ein System so aufgebaut, dass es sich zu hundert Prozent auf diese Sicherheitsstufe verlässt, so liegt es offen, wenn diese ausfällt oder überwunden wird. Beim Aufbau eines Sicherheitssystems sollte immer darauf geachtet werden, dass es auch ohne *Front Line Security* möglichst sicher ist. Die Firewall sollte nur einen zusätzlichen Schutz darstellen.

Im Folgenden sollen Möglichkeiten aufgezeigt werden, mit denen ein auf Linux basierendes System sicherer gemacht werden kann. Die tatsächliche Durchführung dieser Schritte ist in Kapitel 4 dokumentiert.

Entfernen unnötiger Software Pakete

Ein erster Schritt, der zu mehr Sicherheit führen kann, ist, alle nicht benötigten Software Pakete zu entfernen. Dazu ist es zuerst notwendig, zu definieren, welchen Zweck der Server erfüllen muss. So ist es zum Beispiel nicht notwendig, dass auf einem System, welches als Security Server gedacht ist, ein Webserver oder gar ein FTP Server installiert sind. Jede installierte Software, die eigentlich nicht benötigt wird, birgt ein potentielles Risiko, und könnte einem Angreifer zu Gute kommen.

Netzwerk Ports

Einer der wichtigsten Punkte ist, sicherzustellen, dass keine Netzwerk Ports geöffnet sind (also im `LISTEN` Zustand), die nicht unbedingt benötigt werden. Um herauszufinden, welche Ports auf einem System offen sind, eignet sich das Programm netstat. Sind die offenen Ports bekannt, gilt es festzustellen, welche benötigt werden, und welche nicht. Die Dienste

und Programme, die etwaige unnötige Ports geöffnet haben, sind dann ausfindig zu machen und zu beenden.
Anschließend eignet sich noch der Einsatz eines Port Scanners, wie zum Beispiel Nmap, um festzustellen, ob das Vorgehen erfolgreich war. Es kann vorkommen, dass Nmap Ports, die netstat als geöffnet angezeigt hat, nicht anzeigt. Dies kann daran liegen, dass jene Ports von einer Firewall geblockt werden. Ist dies der Fall, ist es dennoch ratsam, nicht benötigte Ports zu schließen. Fällt nämlich die Firewall aus, so würden diese Ports offen liegen.

Dienste

Dienste, die nicht benötigt werden, und aktiv sind, stellen ein potentielles Sicherheitsrisiko dar und sollten deaktiviert werden. Hierbei gilt es jedoch, sorgsam vorzugehen, da das Deaktivieren eines Dienstes, der benötigt wird, zu Instabilitäten im System führen kann.

Inittab

Die Inittab beinhaltet eine Liste von Operationen, die während eines Startups ausgeführt werden. Es ist ratsam zu überprüfen, ob all diese Einträge gerechtfertigt sind. Weiters sollte auch die Datei `/etc/rc.local` überprüft werden. Sie beinhaltet weitere Startup Befehle oder Scripts. Auch hier gilt festzustellen, ob alle Einträge gerechtfertigt sind.

Passwörter

Um mögliche Attacken auf Passwörter zu erschweren, ist es wichtig, Restriktionen einzuführen. So ist es ratsam, eine gewisse Komplexität für Benutzerkennwörter zwingend vorzuschreiben, da zu einfache Passwörter sehr leicht durch *Brute Force* oder *Dictionary* Attacken geknackt werden können. Wenn die Anforderungen an Passwörter allerdings zu komplex sind, neigen viele Benutzer dazu, sich ihre Passwörter aufzuschreiben, was wiederum ein Sicherheitsrisiko darstellt. Ein sicheres Passwort sollte mindestens 8 Zeichen haben, Groß- und Kleinbuchstaben, sowie Zahlen und Sonderzeichen enthalten.

Benutzerkonten

Wie bei Passwörtern, gibt es auch bei Benutzerkonten Möglichkeiten, Missbrauch zu verhindern oder zumindest zu erschweren. Zuerst sollte das System auf nicht verwendete Konten überprüft werden. Diese sollten umgehend entfernt werden.
Direkter Login sollte für alle Systemkonten unterbunden und Zugang mittels `su` für diese Konten eingeschränkt werden.

Permission und Ownership

Dateizugriffsberechtigungen und Besitzverhältnisse sollten regelmäßig überprüft werden, da sich auf diese Weise Spuren von Eindringlingen entdecken lassen. So ist es wichtig, zu überprüfen, ob das SUID bzw. das SGID Bit bei Files gesetzt ist, bei denen dies nicht der Fall sein sollte. Ein Angreifer könnte sich auf diese Weise Zugang zu sonst gesperrten Systemen verschaffen. Weiters ist es ratsam, das System auf Dateien ohne Besitzer zu prüfen. Diese sind an sich noch kein Sicherheitsrisiko. Sollte jedoch zu einem späteren Zeitpunkt ein User angelegt werden, der dieselbe UID hat, wie die Dateien ohne Besitzer, so würde dieser User automatisch Besitzer der Dateien werden.
Schlussendlich ist es noch wichtig, die *Umask* zu kontrollieren. Die *Umask* legt fest, mit welchen Standardrechten Files erstellt werden. Ein guter Wert ist hier 022, was bedeutet,

dass bei neu erstellten Dateien der Besitzer Lese- und Schreibrechte hat, alle anderen nur Leserechte.

SSH

Secure Shell (SSH) bietet durch eine verschlüsselte und authentifizierte Verbindung eine sichere Alternative zu veralteten Protokollen wie telnet. Da telnet jegliche Information (so auch Passwörter) unverschlüsselt überträgt, ist die Verwendung heute nicht mehr ratsam.
Bei aktuellen Linux Distributionen ist telnet standardmäßig deaktiviert, und SSH wird als Remote Login Protokoll verwendet. Es gibt jedoch Methoden, um das ohnehin schon sehr sichere SSH noch sicherer zu machen. So ist es ratsam, SSH Version 2 zu verwenden, Root Login über SSH prinzipiell zu verbieten, um automatisierten Angriffen das Eindringen zu erschweren.

Systemzugriff einschränken

Üblicherweise werden Server von Firewalls geschützt. Um jedoch zu verhindern, dass im Falle einer Fehlfunktion der Firewall jeder beliebig Zugriff auf den Server hat, ist es notwendig, gewisse Sperren auf Betriebssystemebene einzurichten.
Mit Hilfe der Files `/etc/hosts.allow` und `/etc/hosts.deny` ist es möglich, Zugriff auf Netzwerkdienste nach dem Muster `Dienst:Host(s)` zu gewähren bzw. zu verbieten. Im Unterschied zu vielen anderen Zugriffskontrollen, haben Berechtigungen Vorrang gegenüber Verboten.

NFS

NFS, das Network File System, ermöglicht es vernetzten Systemen, Files untereinander auszutauschen. Wie alle Netzwerkdienste, birgt auch die Verwendung von NFS gewisse Risiken.
Um diese Risiken zu mindern, ist es ratsam, einige Einstellungen anzupassen. Dazu zählt das Einschränken des Zugriffs auf den Dienst von außen, sowie das verwenden von TCP anstelle von UDP.

MTA

Selbst wenn ein Linux System nicht als Mailserver dient, wird der Mail Transport Agent (MTA) benötigt, um lokale Nachrichten zuzustellen. Unter Linux wird dafür meist Sendmail oder Postfix verwendet, wobei Postfix Sendmail vorzuziehen ist. Bei beiden gilt es darauf zu achten, dass keine Anfragen von außerhalb beantwortet werden.

Denial of Service verhindern

Es ist möglich, schon auf Betriebssystemebene DoS Attacken teilweise einzuschränken, indem die gewisse Systemressourcen, die einem User zur Verfügung stehen, limitiert werden. So kann zum Beispiel kontrolliert werden, wie viele File Handles ein Benutzer verwenden darf, wobei zwischen Soft- und Hardlimits unterschieden wird. Ein Softlimit produziert eine Warnung, wenn es erreicht wird. Der User hat dann noch die Möglichkeit, bis zum gesetzten Hardlimit File Handles zu erzeugen.

Kernel Parameter

Neben all den erwähnten Maßnahmen für ein sichereres Betriebssystem, bietet Linux auch die Möglichkeit, mit Hilfe von Kernelparametern sicherheitsrelevante Einstellungen vorzunehmen.
Die SYN-Flood Attacke ist ein Denial of Service Angriff, bei dem ein Client eine große Zahl an SYN Paketen an einen Server schickt, jedoch keinen ACK Pakete. Dies führt dazu, dass für jede dieser unvollständigen neuen Verbindungen Ressourcen reserviert werden, solange, bis dem Server schließlich keine Ressourcen mehr zur Verfügung stehen. Um sich gegen diese Art von Angriff zu wehren, können, mit Hilfe des Parameters `net.ipv4.tcp_syncookies = 1` so genannte SYN-Cookies verwendet werden.

ICMP Redirects werden verwendet, um einem Host mitzuteilen, dass die verwendete Route nicht die optimale ist. Darüber hinaus ist es einem Angreifer jedoch möglich, Redirect Pakete zu verwenden, um einem Host einen bestimmten Pfad aufzuzwingen. Um dies zu verhindern, ist es notwendig, ICMP Redirects zu deaktivieren (`net.ipv4.conf.all.accept_redirects = 0`)

Source Address Verification (`net.ipv4.conf.all.rp_filter = 1`) ist eine Methode Denial of Service Attacken durch IP Spoofing zu verhindern. IP Spoofing bedeutet, dass ein Angreifer ein Packet mit einer falschen Quelladresse an einen Host sendet.

Es ist immer ratsam, im Falle eines Angriffs möglichst vollständige Logfiles zur Verfügung zu haben. Dies gibt die Möglichkeit, den Hergang der Attacke zu analysieren, und entsprechend zu reagieren, um beim nächsten Mal besser vorbereitet zu sein. Um Logging für die oben erwähnten Fälle zu aktivieren, muss der Kernel Parameter `net.ipv4.conf.all.log_martians = 1` gesetzt werden.

Login Banner

Wenngleich ein Login Banner keine Sicherheitsmaßnahme im technischen Sinn darstellt, so ist es aus rechtlichen Gründen dennoch empfohlen, einen Banner zu verwenden. Zusätzlich hat ein Banner mit einem expliziten Zugriffsverbot noch den Effekt, dass potentielle Eindringlinge abgeschreckt werden. Der Nachricht des Banners sollte zumindest ein ausdrückliches Verbot des Zugriffs für Unbefugte beinhalten, sowie die Androhung rechtlicher Schritte bei Zuwiderhandlung.

3.1.2 Bastille Linux

Als Alternative, bzw. auch ergänzend zu den in 3.1.1 genannten Möglichkeiten zur Systemhärtung, kann Bastille Linux eingesetzt werden. Es besteht aus einer Reihe von Scripts, die den Hardening Prozess automatisieren und läuft in zwei verschiedenen Modi. Ein interaktiver Modus und ein nicht interaktiver Modus. Letzterer wird verwendet um viele Systeme in kurzer Zeit mit Hilfe einer vordefinierten Konfigurationsdatei abzusichern. Beim Interaktiven Modus muss der Benutzer dem Programm einige Frage bezüglich der Nutzung des Systems beantworten. Bastille Linux sichert das System entsprechend der Antworten des Benutzers ab. Zusätzlich liefert das Programm eine Reihe von Vorschlägen, welche Maßnahmen, die die Software nicht abdeckt, noch zusätzlich getroffen werden sollten. Eine Implementierung von Bastille findet sich im praktischen Teil der Arbeit (siehe 4.3.2). (vgl. Bastille Linux-Manpage 2005)

Betriebssystemsicherheit Fazit

So wie ein Haus, und ist es auch noch so massiv, früher oder später einstürzen wird, wenn es auf einem schlechten Fundament gebaut wurde, so wichtig ist es, durch die Absicherung des Betriebssystems ein Fundament für ein sicheres System zu schaffen. Selbst die besten Security Tools bieten keinen Schutz, wenn ein schlecht konfiguriertes Betriebssystem darunter liegt, das Lücken und Exploits anbietet, die es ermöglichen die Sicherheitsvorkehrungen zu umgehen.
In Kapitel 4 sind die einzelnen oben beschriebenen Schritte zur Absicherung des Testsystems dokumentiert. Das weitere Kapitel befasst sich mit Open Source Security Tools.

3.2 Firewalls

Eine Firewall ist ein Stück Hard- oder Software, das die Aufgabe hat, Traffic zwischen Netzwerken voneinander zu trennen und zu kontrollieren. So dient eine Firewall zum Beispiel dazu, zu kontrollieren, welche Dienste Benutzer eines Netzwerks nutzen dürfen, oder das interne Netzwerk dem Internet gegenüber abzuschotten. Man unterscheidet im Allgemeinen zwischen Packet Filter und Application Firewalls. Eine Application Firewall kontrolliert den Zugriff von Applikationen auf die Kommunikationsdienste des Betriebsystems, während ein Packet Filter den Datenfluss auf Layer 3 und Layer 4 des OSI Reference Model steuert.
In einem Unternehmen ist die Absicherung des Netzwerks durch eine Firewall heutzutage nicht mehr wegzudenken, da Sicherheitsverletzungen mitunter enorme Kosten verursachen können.
Zwar gibt es eine Vielzahl an kommerziellen Firewall Systemen, wie zum Beispiel Cisco Pix, Checkpoint oder Juniper NetScreen, jedoch sind diese für kleine und mittlere Unternehmen oft nicht finanzierbar.
Als Alternative können Unternehmen auch auf Open Source Firewalls zurückgreifen, welche oft kostenlos angeboten werden. Einige Produkte aus diesem Bereich sollen in diesem Kapitel genauer untersucht werden und später, im praktischen Teil der Arbeit, auf ihre Tauglichkeit überprüft werden.

3.2.1 Grundlagen

Um mit Firewalls richtig umgehen zu können, ist es zuerst notwendig, die Grundlagen der Netzwerktechnik zu verstehen. Auch wenn diese Arbeit nicht als Einführung in die Netzwerktechnik dienen soll, und ein gewisses Grundverständnis vorausgesetzt wird, soll an dieser Stelle aus Gründen der Vollständigkeit dennoch ein kleiner Überblick über die zugrunde liegende Materie gegeben werden.

OSI Layer Nummer	Layer Name
Layer 7	Application Layer
Layer 6	Presentation Layer
Layer 5	Session Layer
Layer 4	Transport Layer
Layer 3	Network Layer
Layer 2	Data Link Layer
Layer 1	Physical Layer

Physical Layer (*dt. Bitübertragungsschicht*) Der Physical Layer stellt die unterste Schicht des OSI Reference Models dar. Sie beschäftigt sich mit Signalen, Übertragungsverfahren und Komponenten. So werden z.B. Antennen, Stecker, Kabel oder Hubs dieser Schicht zugeordnet.

Data Link Layer (*dt. Sicherungsschicht*) Aufgabe des Data Link Layers ist es, den Zugriff auf das Übertragungsmedium zu regeln, sowie eine möglichst fehlerfreie Kommunikation zu gewährleisten. Dazu werden die Daten in dieser Schicht in Blöcke (*sog. Frames*) unterteilt, und mit Prüfsummen zur Fehlerkorrektur versehen. Weiters kommt auf Layer 2 *Flow Control* zum Einsatz, um die Geschwindigkeit zwischen den Kommunikationspartnern dynamisch zu regeln. Ein bekanntes Layer 2 Protokoll ist zum Beispiel Ethernet (IEEE 802.3).

Network Layer (*dt. Vermittlungsschicht*) Layer 3 des OSI Modells ist für die Datenübermittlung zwischen Sender und Empfänger verantwortlich. Dies beinhaltet die Wegsuche (*Routing*) zwischen den Teilnehmen, da nur selten eine direkte Verbindung besteht. Das wohl bekannteste und am meisten verwendete Layer 3 Protokoll ist IP.

Transport Layer *(*dt. Transportschicht) Die Transportschicht beschreibt die *End-to-End* Kommunikation zwischen zwei Systemen, und stellt eine zuverlässige Grundlage für die darüber liegenden Schichten dar. Zu den Aufgaben dieser Schicht zählen zum Beispiel Segmentierung und Fehlerkorrektur. TCP und UDP zählen zu den wichtigsten Layer 4 Protokollen.

Session Layer (*dt. Sitzungsschicht*) Im Session Layer wird der Aufbau, der Verlauf und der Abbau von Verbindungen (*Sessions*), sowie das setzen von so genannten Checkpoints gesteuert.

Presentation Layer (*dt. Darstellungsschicht*) Aufgabe dieser Schicht ist es, einerseits die Daten in ein für den Application Layer verständliches Format zu übersetzen (z.B. ASCII Text, JPG Bild), und andererseits die Daten für die darunter liegenden Schichten in eine systemunabhängige Form zu bringen, um einen syntaktisch einwandfreien Datenverkehr zu gewährleisten.

Application Layer (*dt. Anwendungsschicht*) Der Application Layer ist die oberste Ebene des 7-Schicht Modells. Ihre Aufgabe ist es, Anwendungen ein Kommunikationsinterface zur Verfügung zu stellen. Bekannte Protokolle in dieser Schicht sind HTTP, FTP oder SMTP.

(vgl. Odom 2003)

Die Aufgaben einer Packet Filter Firewall spielen sich hauptsächlich in Layer 3 und 4 des OSI Reference Model ab.

Funktionsweise

Zwar gibt es eine Vielzahl unterschiedlicher Netzwerkprotokolle, wie zum Beispiel IPX/SPX oder AppleTalk, jedoch haben diese heute nur mehr eine untergeordnete Rolle. Aus diesem Grund wird sich diese Arbeit ausschließlich mit dem TCP/IP Protokollstack befassen, da dieser am weitesten verbreitet ist.

Eine Firewall, im ursprünglichen Sinn, ist eine bauliche Trennung zwischen Gebäuden, die im Brandfall das Übergreifen der Flammen verhindern soll. Im Grunde genommen ist eine Firewall im Sinne der IT Security analog dazu. Ihre Aufgabe ist es, Netzwerke voneinander zu trennen, und den Datenaustausch zu kontrollieren. Im Allgemeinen spricht man von einem Internen Netzwerk (oder *Inside*) und einem externen Netzwerk (oder *Outside*), die durch die Firewall voneinander getrennt werden.

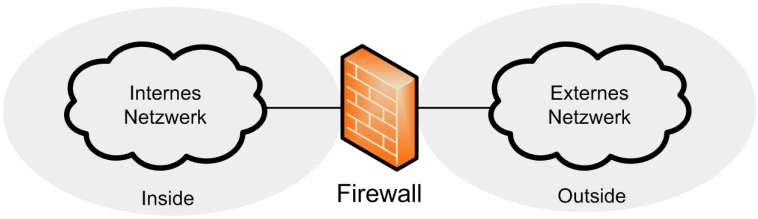

Abbildung 3.1: Schematische Darstellung einer Firewall

Trifft ein Datenpaket auf eine Firewall, so hat diese die Aufgabe, anhand zuvor definierter Richtlinien, zu entscheiden, was mit diesem Paket geschehen soll. Vereinfacht gesprochen hat die Firewall drei Möglichkeiten: Sie lehnt das Paket ab, sie nimmt es an, sofern es an die Firewall selbst gerichtet ist, oder sie leitet es an ein angeschlossenes Netzwerk weiter.
Beim Konfigurieren einer Firewall gibt es prinzipiell zwei Ansätze. Entweder, man erlaubt alles, und definiert dann, was verboten werden soll, oder, alles wird verboten, und man definiert nur, was explizit erlaubt sein soll. Letzterer Ansatz ist meist zu bevorzugen, und es wird nur wenige Umgebungen geben, in denen eine „*Allow All*“ Konfiguration geeignet ist, da diese zu unsicher ist bzw. der Konfigurationsaufwand ins Unermessliche steigen kann. So sind auch die meisten Hardware Firewalls so eingestellt, dass unabhängig von der Konfiguration ein implizites „*Deny All*“ existiert, das alles, was nicht explizit erlaubt ist, verbietet.
Ziel einer guten Firewall Konfiguration ist es, dass nur jene Pakete die Firewall passieren, die von der Security Policy als sicher erachtet werden, alle anderen verworfen werden, und es auch keinen Weg gibt, dies zu umgehen.

Im Folgenden werden nun einige Open Source Firewall Produkte analysiert.

3.2.2 iptables

Die erste Firewall, die analysiert werden soll, ist iptables. Iptables ist der Name des User Space Tools, mit dem, unter Zuhilfenahme des Netfilter Kernel Moduls, eine Packet Filter Firewall realisiert werden kann. Meist wird die gesamte Firewall Infrastruktur als iptables bezeichnet, wenngleich die eigentliche Arbeit vom Netfilter Modul verrichtet wird. Iptables ist heute ein Standardbestandteil jeder Linux Distribution.
Mit iptables ist es möglich, eine vollwertige Packet Filter Firewall zu erstellen, die ermöglicht, Pakete zu filtern und zu verändern, den Status von Verbindungen zu überwachen und NAT zu implementieren.

Der folgende Abschnitt gibt einen kurzen Einblick in die Entstehungsgeschichte von iptables, und schildert anschließend die genaue Funktionsweise der Software.

Geschichte

Das iptables Projekt wurde 1998 von Rusty Russell, als Nachfolger von ipchains, ins Leben gerufen. Das Projekt wuchs, und so wurde 1999 das Netfilter Core Team gegründet. Die von ihnen geschrieben Software, Netfilter, wird unter der GNU GPL herausgegeben und wurde im März 2000 in den Linux 2.3 Kernel eingebunden.
Die vorherrschenden Applikationen zur Erstellung einer Linux Firewall waren, vor iptables, ipchains in Linux 2.2 und ipfwadm in Linux 2.0.
Die grundlegende Idee hinter iptables ist immer noch dieselbe wie bei ipfwadm: Eine Liste mit Regeln, die festlegt, nach welchen Kriterien Pakete als zutreffend erkannt werden, und wie mit solchen Paketen verfahren werden soll.

Das Konzept der *Chains*, das sind Ketten von Regeln (z.B. INPUT oder OUTPUT), wurde in *ipchains* hinzugefügt und später, in *iptables*, noch durch *Tables* erweitert. So gibt es nun beispielsweise eine Table für NAT Entscheidung und eine andere für Filter Entscheidungen. Zusätzlich wurden noch die drei Punkte, an denen gefiltert wird, so verändert, dass ein Paket auf seinem Weg durch die Firewall nur zu einem einzigen Filter Punkt kommen kann.
So wurde es bei iptables (im Unterschied zu ipfwadm oder ipchains) möglich, dass Filtering, Connection Tracking und NAT voneinander getrennt ablaufen. Diese Trennung erlaubt iptables nun, Informationen des Connection Trackings getrennt von NAT zu verwenden, und ermöglichte es, den Zustand einer Verbindung zu erkennen und Entscheidungen aufgrund dieses Zustandes zu treffen. Eine Firewall, die diese Fähigkeit besitzt, wird *stateful* genannt. Im Unterschied dazu ist ipchains eine so genannte *stateless* Firewall, da Entscheidungen nur aufgrund von Quell- und Zielinformationen getroffen werden können. (vgl. Netfilter Website 2005)

Funktionsweise

Netfilter/iptables (im Folgenden *iptables* genannt) ermöglicht es, Regeln zu erstellen, die definieren, wie mit Datenpaketen umgegangen werden soll. Diese Regeln sind in Ketten, so genannten *Chains*, organisiert. Jede Chain besteht aus einer Liste von Regeln. Chains wiederum sind zu Tabellen (*Tables*) zusammengefasst, wobei jede Table einer anderen Funktion zugeordnet ist (siehe Abb. 3.2).

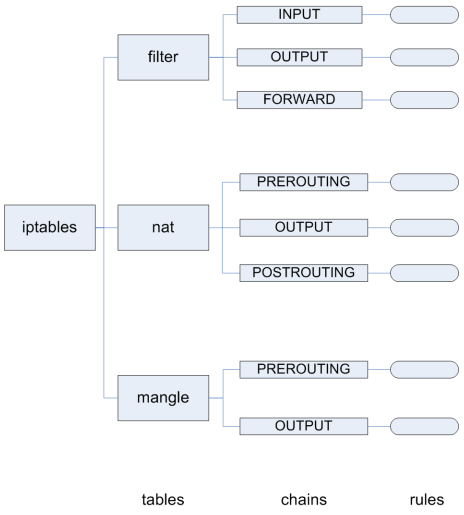

Abbildung 3.2: Tables, Chains und Rules in iptables

Jede Regel enthält ein bestimmtes Muster, dem ein Paket entsprechen muss, um die Regel zu erfüllen, sowie ein Ziel (*Target*), das besagt, was mit einem zutreffenden Paket passieren soll. Jedes Paket, das am System ankommt, oder das System verlässt, durchläuft auf seinem Weg zumindest eine Chain (siehe Abb. 3.3). Diese wird sequentiell abgearbeitet, bis das Paket einer Regel entspricht. Wird eine passende Regel gefunden, so wird der Durchlauf des Pakets gestoppt, und das entsprechende Target angewendet. Findet sich keine passende Regel, bis das Paket das Ende der Chain erreicht hat, so wird eine zuvor definierte Standard Policy angewandt. (vgl. Eychenne 2002)

Tables und Chains

Iptables hat drei vordefinierte Tables: Filter, NAT und Mangle. Es ist auch möglich, über Erweiterungsmodule neue Tables zu erstellen. Jede der drei Tables besteht aus einigen vorgegebenen Chains. Bei der Konfiguration ist es möglich, eine beliebige Anzahl an eigenen Chains hinzuzufügen. Standardmäßig sind alle Chains leer, das bedeutet, sie enthalten keine Regeln, und die Standard Policy ist so definiert, dass alle Pakete passieren dürfen. Im Folgenden sollen nun die Aufgaben der einzelnen Tables, sowie der vordefinierten Chains kurz erläutert werden.

Filter Table Aufgabe der Filter Table ist es, Pakete zu filtern. Dies ist der eigentliche Firewall Teil, der Packet Filter. Die Filter Table enthält folgende vordefinierte Chains:

- **INPUT Chain** - Alle Pakete, die für das lokale System, also für die Firewall selbst, gedacht sind, durchlaufen die INPUT Kette.
- **OUTPUT Chain** - Alle vom System erzeugten Pakete durchlaufen diese Kette.
- **FORWARD Chain** - Die FORWARD Chain ist verantwortlich für all jene Pakete, die die Firewall durchlaufen. Also all jene, die von einem angeschlossenen Netzwerk kommen, und ein anderes angeschlossenes Netzwerk als Ziel haben.

NAT Table Die NAT Table ist zuständig für Network Address Translation. Unter NAT versteht man das Ändern der Source oder Destination IP Adresse innerhalb eines Pakets. Dies wird meist genutzt, um private IP Adressen auf öffentliche abzubilden. Man unterscheidet zwischen Source- und Destination-NAT. Source-NAT wird meist verwendet, um private Adressen hinter öffentlichen zu *„verstecken“*, während Destination-NAT genutzt wird, um Anfragen von außerhalb ins interne Netz weiterzuleiten (*Virtual Server*). Spricht man von NAT, so ist in den meisten Fällen eigentlich PAT (Port Address Translation) gemeint, da auch die Source Ports verändert werden. Die NAT Table enthält folgende Chains:

- **PREROUTING Chain** - Pakete durchlaufen die PREROUTING Chain, bevor eine Routing Entscheidung getroffen wird. Diese Kette wird hauptsächlich für DNAT verwendet.
- **POSTROUTING Chain** - Diese Kette ist für ausgehende Pakete, nachdem die Routing Entscheidung getroffen wurde, und wird hauptsächlich für SNAT verwendet.
- **OUTPUT Chain** - Erlaubt das Verändern von lokal generierten Paketen.

Mangle Table Die Mangle Table ermöglicht es, Pakete zu verändern. Mögliche Änderungen sind zum Beispiel das TOS Feld oder die TTL. Ursprünglich konnten Pakete nur in der PREROUTING und der OUTPUT Chain verändert werden. Seit Kernel 2.4.18 ist dies jedoch in allen Chains möglich.

- **PREROUTING Chain** - Die PREROUTING Chain wird durchlaufen, bevor eine Routing Entscheidung getroffen wird.
- **INPUT Chain** - Alle Pakete, die an das lokale System gerichtet sind durchlaufen diese Kette.
- **FORWARD Chain** - Für alle Pakete, die die Firewall durchlaufen.
- **OUTPUT Chain** - Alle am System erzeugten Pakete durchlaufen diese Kette.
- **POSTROUTING Chain** - Die POSTROUTING Chain wird durchlaufen, nachdem eine Routing Entscheidung getroffen wurde.

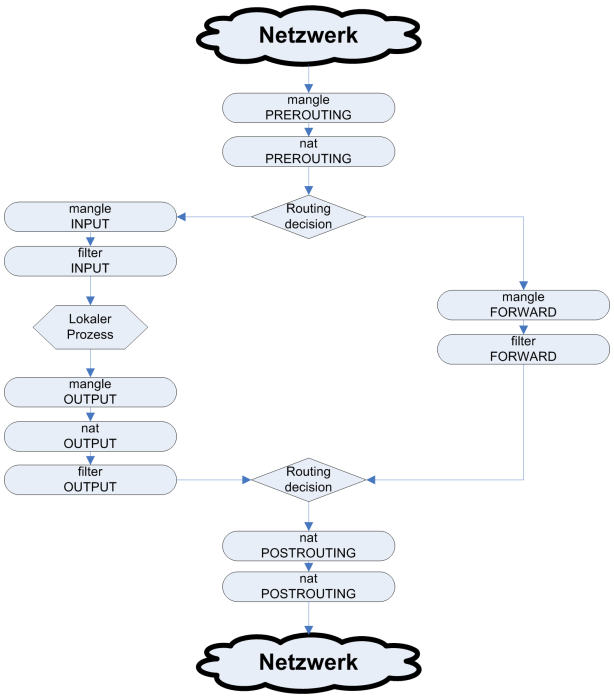

Abbildung 3.3: Funktionsweise von iptables

Beim Durchlaufen der Chains, werden die Datenpakete mit allen Regeln, die sie passieren, verglichen. Es gibt jedoch Einschränkungen, in welcher Kette, welche Eigenschaften überprüft werden können. So ist beispielsweise der Parameter *Outgoing Interface* in der INPUT Kette logischerweise nicht anwendbar. Genauso kann auf gewisse Targets nur in bestimmten Ketten verwiesen werden. Eine ausführliche Beschreibung aller Targets gibt der nächste Abschnitt. (vgl. Eychenne 2002)

Targets

Wie bereits erwähnt, besteht jede Chain aus Regeln. Jede Regel besteht im Grunde genommen aus zwei Teilen. Ein Muster, mit dessen Hilfe zutreffende Pakete erkannt werden, und ein Ziel (*Target*), das festlegt, was mit einem passenden Paket geschehen soll. Als Target kommt entweder eine selbst definierte Chain, oder ein Standardziel in Frage. Wenn keine Regel einer Chain zutrifft, kommt die voreingestellte Policy zu tragen. Die folgenden Standardziele sind bei iptables möglich:

- **ACCEPT** - Dieses Target veranlasst Netfilter, das Paket zu akzeptieren. Abhängig davon, wo das Target definiert wird, werden unterschiedliche Dinge zugelassen. Steht das ACCEPT Target in der INPUT Chain, so kann das Paket vom Host empfangen werden, steht es in der OUTPUT Chain, so kann es versendet werden. Ein ACCEPT Target in der FORWARD Chain veranlasst, dass das Paket weitergeleitet werden darf.
- **DROP** - Trifft ein Paket auf das DROP Target, so wird es ohne weitere Bearbeitung verworfen. Es wird auch keine Benachrichtigung an den Absender verschickt, was zum Ergebnis hat, dass für einen potentiellen Angreifer scheinbar gar kein Host an dieser Adresse existiert.
- **QUEUE** - Das Paket wird in den User Space weitergeleitet, so dass es von einer Applikation gelesen werden kann. Wird die Queue von keinem Programm gelesen, hat dieses Target denselben Effekt wie DROP.
- **RETURN** - Verweist eine Regel auf eine benutzerdefinierte Kette, so wird diese durchlaufen. Trifft keine der Regeln der benutzerdefinierten Kette auf das Paket zu, so springt dieses am Ende der Kette, ähnlich einer Subroutine beim Programmieren, wieder zurück zur aufrufenden Kette, gleich hinter der Regel, die den Sprung verursacht hat. Das RETURN Target erzielt genau den gleichen Effekt. Trifft eine Regel, die RETURN als Target festgelegt hat, in einer benutzerdefinierten Kette zu, so springt das Paket zurück zur aufrufenden Kette. Wird RETURN innerhalb einer Standard Chain definiert, wird diese verlassen, auf die Policy der Chain zurückgegriffen.
- **REJECT** - Dieses Target hat denselben Effekt wie das DROP Target, mit dem Unterschied, dass der Sender über das Verwerfen des Pakets informiert wird. Die Art der Antwort kann vom Benutzer gewählt werden. Es stehen verschiedene ICMP Signale und TCP-RST zur Auswahl.
- **LOG** - Das LOG Target veranlasst, dass Informationen über das Paket im Syslog aufgezeichnet werden, und das Paket anschließend weiter durch die Kette geschickt wird (*non terminating target*).
- **ULOG** - ULOG steht für *Userspace Logging*. Die Logging Informationen werden jedoch nicht, wie bei LOG, in das Syslog geschrieben, sondern über einen Netlink Socket an einen oder mehrere Userspace Applikationen geschickt. Wie bei LOG, wird auch hier das Paket weiter durch die Chain geschickt.
- **DNAT** - Mit diesem Target lässt sich die Zieladresse, und optional auch der Port, eines Pakets verändern. Es ist nur in der OUTPUT und der PREROUTING Kette der NAT Table gültig. Die Änderungen, die mit DNAT vorgenommen werden, gelten auch für alle folgenden Pakete derselben Verbindung. Antworten werden automatisch entsprechend geändert.
- **SNAT** - SNAT veranlasst die Änderung der Quelladresse und des Ports, und ist nur in der POSTROUTING Kette der NAT Table gültig. Wie auch bei DNAT, werden alle Änderungen auch bei allen folgenden Paketen derselben Verbindung automatisch vorgenommen.
- **MASQUERADE** - Dieses Target stellt eine spezielle Form von SNAT dar, und wird für dynamisch vergebene IP Adressen verwendet, wie es z.B. bei einem DSL Anschluss der Fall ist. Statt einer festen Adresse, wie bei SNAT, wird die Adresse, die für MASQUERADE verwendet werden soll, jedes Mal vom entsprechenden ausgehenden Interface übernommen.

- **REDIRECT** - Mit dem REDIRECT Target ist es möglich, eine Verbindung am lokalen Rechner umzuleiten. Auf diese Weise können z.B. transparente Proxyserver realisiert werden. Das Redirect Target ist nur in der PREROUTING und der OUTPUT Kette der NAT Table gültig.
- **MIRROR** - Das MIRROR Target wird als experimentell eingestuft. Es vertauscht die Quell- und Zieladresse eines Pakets, und schickt dieses wieder zurück, ohne dabei von weiterem Filtering oder NAT beeinflusst zu werden. Dieses Target ist nur in der INPUT, FORWARD und PREROUTING Kette gültig. (vgl. Eychenne 2002)

Connection Tracking

Ein wichtiger Bestandteil von iptables ist das *conntrack* Modul, das *Connection Tracking* ermöglicht. Connection Tracking erlaubt es, Überblick über alle am System bestehenden Verbindungen zu behalten, und alle Pakete einer Verbindung zuordnen zu können. NAT verwendet Connection Tracking, um zu erkennen, welche Pakete zu welcher Verbindung gehören, und diese entsprechend zu übersetzen, während iptables diese Information verwendet, um eine *Stateful Firewall* realisieren zu können.

Connection Tracking unterscheidet Pakete nach ihrem Zustand. So ist ein Paket `NEW`, wenn es zu keiner existierenden Verbindung zugeordnet werden kann, oder `ESTABLISHED`, wenn es Teil einer bestehenden Verbindung ist. Des Weiteren kennt Connection Tracking noch die Zustände `RELATED` und `INVALID`. Als `RELATED` wird ein Paket bezeichnet, wenn es nicht Teil einer bestehenden Verbindung ist, jedoch mit einer Verbindung assoziiert ist. Durch die Verwendung von entsprechenden Plug-Ins wird iptables sogar fähig, Layer 7 Protokolle, so z.B. FTP, zu verstehen und auf diese Weise zwei Verbindungen als `RELATED` zu erkennen. Ist ein Paket neu, jedoch nicht fähig, eine neue Verbindung aufzubauen (wenn also das `SYN` Flag nicht gesetzt ist), so wird es als `INVALID` klassifiziert.

Werden diese Zustände verwendet, um zu filtern, so spricht man von einer *Stateful Firewall*. Mit Hilfe einer solchen Firewall, ist es möglich zu kontrollieren, welche Art von Paketen zugelassen werden soll. So könnte man zum Beispiel nur Pakete, die im Zustand `ESTABLISHED` oder `RELATED` sind, von außen zulassen. Dies würde ermöglichen, dass Hosts von außerhalb nicht in der Lage sind, eine neuen Verbindungen aufzubauen, jedoch bestehende Verbindungen problemlos in beide Richtungen laufen zu lassen. (vgl. Eychenne 2002)

Fazit

Mit iptables steht eine mächtige und erweiterbare Firewall zur Verfügung, die obendrein noch kostenlos, und Bestandteil der meisten Linux Distributionen ist. Iptables wird ständig weiterentwickelt, ist durch Module weitgehend unbegrenzt erweiterbar, und ist selbstverständlich IPv6-fähig. All dies sind Merkmale, die iptables zu einer starken Packet Filter Firewall machen, die auch in Zukunft Bestand haben wird.

Eine detaillierte iptables Konfiguration wird im praktischen Teil der Arbeit (Abschnitt 4.4.4) erstellt und analysiert.

3.2.3 Turtle Firewall

Turtle Firewall ist an sich eigentlich keine Firewall. Vielmehr ist es ein graphisches Frontend, das auf iptables und Webmin basiert. Webmin ist ein Mini-Webserver, der in den meisten Linux Distributionen enthalten ist.

Die Idee hinter Turtle ist, die Implementierung und Administration einer Firewall auch Personen zugänglich zu machen, die auf diesem Gebiet nicht sehr versiert sind. Dies wird anhand einer Reihe von Perl Scripts realisiert. Unter dieser Vereinfachung leidet selbstverständlich auch die Komplexität. Dennoch ist es mit Turtle Firewall möglich, alle grundlegenden Filter Optionen von iptables, sowie NAT zu implementieren.

Konfiguration

Die Konfiguration von Turtle basiert auf Zonen ist ausschließlich browserbasiert (siehe Abb. 3.4). Turtle teilt Zonen in „*good*“ und „*bad*“ ein, was soviel bedeutet, wie vertrauenswürdige und nicht vertrauenswürdige Zonen. Außerdem unterstützt Turtle auch eine DMZ (*Demilitarized Zone*).

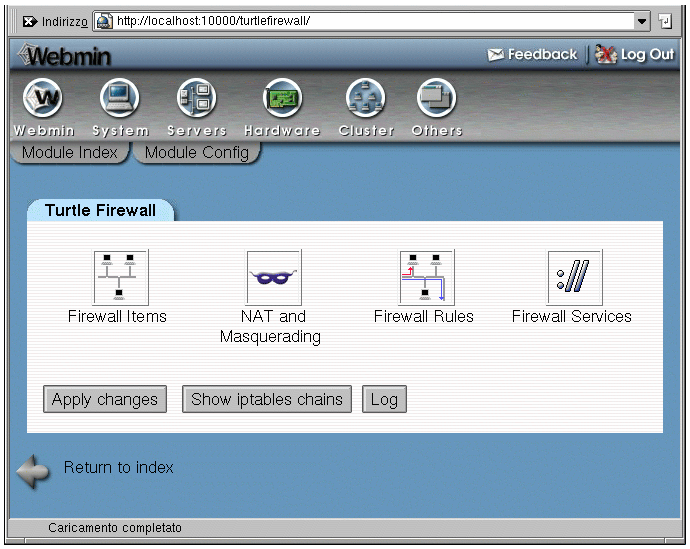

Abbildung 3.4: Turtle Firewall Webmin Konfigurationsbildschirm

Sind die Interfaces den Zonen zugeteilt, können noch einfache Regeln erstellt werden. So kann beispielsweise festgelegt werden, welcher Host aus dem internen Netzwerk auf das externe zugreifen darf, oder welche externen Hosts auf Server in der DMZ zugreifen dürfen. Das Erstellen der Regel läuft graphisch unterstützt ab.
Um NAT zu verwenden ist es lediglich notwendig, zu definieren, für welche Zone NAT verwendet werden soll, was im Allgemeinen die *„gute“* Zone sein wird. (vgl. Frigido 2002)

Fazit

Für kleine Unternehmen, die keine eigene IT Abteilung haben, jedoch auch nicht die finanziellen Mittel, um externe Firmen zu beauftragen, ist Turtle Firewall ist eine gute Möglichkeit, auf einfachem und schnellem Wege eine Firewall zu implementieren, die zumindest

die grundlegenden Funktionen beherrscht. Für komplexere Anwendungen sollte jedoch, zugunsten der Konfigurierbarkeit, auf das grafische Interface verzichtet werden.

3.2.4 SmoothWall Express

SmoothWall Express ist ein Open Source Stand-Alone Komplettpaket auf Basis von GNU/-Linux. Zur Verwendung von SmoothWall, ist es nicht notwendig, zuvor ein Betriebsystem zu installieren. Diese so genannte *Turn-Key* Variante läuft auf einem *Dedicated Server*, dessen einzige Aufgabe es ist, als Firewall zu fungieren. Es wäre selbstverständlich auch möglich, die Funktionalität von SmoothWall zu erreichen, indem manuell verschiedene Tools installiert werden, jedoch ist die *Turn-Key* Variante wesentlich besser geeignet, für Personen, die keine Fachkräfte sind, aber dennoch nicht auf ein gewisses Maß an Sicherheit verzichten wollen.

Im Folgenden sollen kurz die einzelnen Bestandteile von SmoothWall Express beschrieben werden.

Komponenten

- **Firewall** - SmoothWall basiert auf Netfilter/iptables, verfügt allerdings nur über eingeschränkte Konfigurationsmöglichkeiten.
- **VPN** - Diese Funktion bietet die Möglichkeit, einen IPSec Tunnel zu einem anderen System aufzubauen.
- **DHCP** - SmoothWall Express enthält sowohl einen DHCP Client als auch einen Server. Der Client ist meist notwendig, um vom ISP eine IP Adresse zu beziehen, während der Server dazu dient, IP Adressen im LAN zu vergeben.
- **SSH und Web Zugang** - Die Konfiguration kann über SSH Kommandozeile oder über ein Web-Interface durchgeführt werden. Beide Möglichkeiten werden vom SmoothWall unterstützt.
- **Proxy Server** - SmoothWall bietet einen WebProxy Server, über den Clients ins Internet verbinden können.
- **Caching Server** - Zusätzlich bietet das Programm auch einen Caching Server, der Inhalte speichern und Zugriffe damit beschleunigen kann.
- **IDS** - SmoothWall Express bietet einige grundlegende Network Intrusion Detection Fähigkeiten, die über Snort realisiert werden, welches später in diesem Kapitel behandelt wird (siehe 3.3.2).
- **Analyse Tools** - Eine Reihe von Graphen dienen der Analyse von Bandbreitenbedarf oder Netzwerkproblemen.

Es existiert auch eine kommerzielle Version von SmoothWall, SmoothWall Corporate, die über weitere Features verfügt. Diese soll hier jedoch nicht behandelt werden.
Die Installation verläuft über eine bootfähige CD und setzt keinerlei Linux Kenntnisse voraus. Die grundlegenden Einstellungen wie Hostname oder IP Adresse, aber auch komplexere, wie Web Proxy Einstellungen oder DHCP Server Konfiguration, können bereits während der Installation vorgenommen werden. Nach der Installation ist SmoothWall auch ohne weitere Konfiguration einsatzbereit.

Abbildung 3.5: SmoothWall Express Web Interface

Administration

Wie bereits erwähnt, bringt SmoothWall den großen Vorteil mit sich, dass die Administration keinerlei Fachwissen voraussetzt. Die Konfiguration kann vollständig über das integrierte Web Interface (siehe Abb. 3.5) vorgenommen werden, das sowohl HTTP, als auch sichere HTTPS Verbindungen zulässt. Wie auch schon bei Turtle Firewall, werden die verschiedenen Netzwerke bei SmoothWall in Zonen eingeteilt. So stellt beispielsweise *„Green“* das Intranet dar, während *„Red“* für das Internet steht. Entsprechend der Zoneneinstellungen nimmt SmoothWall die Firewallbasiskonfiguration vor.

Sämtliche Funktionen, wie VPN, Web Proxy oder DHCP Server können vollständig über das Web Interface konfiguriert werden. An dieser Stelle soll jedoch nur der Firewall Teil behandelt werden.
Die Konfigurationsmöglichkeiten der Firewall beschränken sich auf Port Forwarding, IP Blocking, PPP, sowie einige wenige erweiterte Einstellungen. (vgl. SmoothWall Limited 2003)

Fazit

Alles in allem ist SmoothWall Express eine vollständige und dank geringer Hardware Anforderungen günstige Sicherheitslösung, die den Anforderungen der meisten Unternehmen gerecht werden sollte.
Durch die einfache Administration mittels Web Interface, ist SmoothWall ideal geeignet für Firmen, die keine eigene IT Abteilung haben, jedoch nicht auf ein breites Spektrum an Security Komponenten verzichten wollen. Wie die meisten Lösungen mit graphischem Interface, leidet leider auch bei SmoothWall der Grad der Konfigurierbarkeit. So sind Firewallkonfigurationen auf einem Level, wie es mit iptables möglich wäre, weit entfernt von den Fähigkeiten von SmoothWall. Es ist allerdings in Betracht zu ziehen, dass diese Einschränkung für einen großen Teil der potentiellen Benutzer nicht relevant ist, da derlei komplexe und spezielle Konfigurationsmöglichkeiten in den meisten Unternehmen nicht vonnöten sein werden.

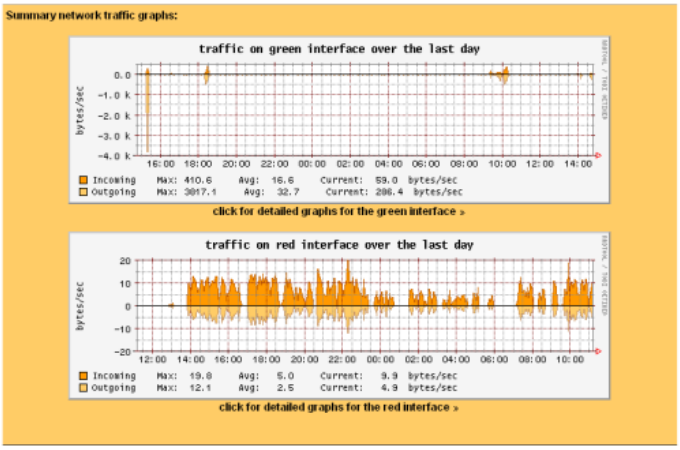

Abbildung 3.6: SmoothWall Express Traffic Report

3.2.5 PF - OpenBSD Packet Filter

PF ist die Packet Filter Firewall von OpenBSD, die sowohl TCP/IP Filtering, als auch NAT und QoS unterstützt. Die Konfiguration von PF erfolgt über eine Datei, die sich in sieben Abschnitte teilt.

- **Macros** - In diesem Abschnitt werden benutzerdefinierte Variablen festgelegt. Diese können unter anderem IP Adressen oder Interface Namen beinhalten.
- **Tables** - Eine Table bei PF, ist eine Struktur, in der Listen von IP Adressen gespeichert werden.
- **Options** - An dieser Stelle können verschiedene Optionen definiert werden, die das Verhalten von PF steuern.
- **Scrub** - Zuständig für Normalisierung und Defragmentierung.
- **Queueing** - Bandbreitenkontrolle und Paketpriorisierung ist an dieser Stelle möglich.
- **Translation** - Dieser Abschnitt ist für NAT zuständig
- **Filter Rules** - Zum Schluss werden Filter Regeln festgelegt.

(vgl. OpenBSD Website 2006)

Mit Ausnahme von Macros und Tables, müssen die Abschnitte in oben stehender Reihenfolge in der Konfigurationsdatei stehen. Im Folgenden sollen die einzelnen Abschnitte genauer analysiert werden.

Macros

Macros sind bei PF benutzerdefinierte Variable, die verschiedene Daten wie z.B. IP Adressen beinhalten können. Macros sind dazu gedacht, die Wartung der Konfigurationsdatei zu vereinfachen. Ein Beispiel für die Anwendung eines Macros wäre:

```
ext_if = "fxp0"
block in on $ext_if from any to any
```

Im obigen Beispiel wird der Name der ausgehenden Schnittstelle durch eine Variable ersetzt. Sollte sich die ausgehende Schnittstelle ändern, würde diese Form den Konfigurationsaufwand deutlich verringern. (OpenBSD Website 2006)

Tables

Tables sind dafür gedacht, große Mengen an IP Adressen zu enthalten, da das Nachschlagen in Tables sehr schnell funktioniert. Dies macht es sehr effizient, einzelne Regeln mit Tables zu verwenden, anstatt mehrerer Regeln mit einzelnen Adressen. Tables können für Quell- und Zieladressen von Filter-, Scrub-, NAT- oder Redirection-Regeln verwendet werden. Außerdem können sie noch für Übersetzungsadressen für NAT verwendet werden, sowie für einige routing Optionen.

Options

Options steuern das Verhalten der Firewall. So kann Beispielsweise mit `set block-policy [drop|return]` festgelegt werden, wie die Firewall bei einem *block* Statement reagieren soll. Es gibt eine relativ große Anzahl an möglichen Optionen, mit denen sich das Verhalten der Firewall unterschiedlichen Bedürfnissen und Situationen anpassen lässt. Eine Liste aller Optionen befindet sich im Handbuch zu PF.

Scrub

Die `scrub` Direktive bewirkt, dass Datenpakete normalisiert und defragmentiert werden. Außerdem verwirft `scrub` Pakete mit nicht konsistenten Flags (z.B. fragmentierte Pakete bei denen das *do not fragment* Bit gesetzt ist). Scrub Regeln ähneln den Filter Regeln, was es relativ einfach macht, zu spezifizieren, welche Pakete *„gescrubbed“* werden sollen, und welche nicht. Scrub hat eine Reihe von Optionen, die hilfreich sind, um sich an gewisse Situationen optimal anpassen zu können. Im Allgemeinen ist es jedoch ratsam, alle Pakete mit `scrub in all` zu bearbeiten. (vgl. OpenBSD Website 2006)

Queueing

Mit PF ist es möglich, mittels Queueing, QoS zu realisieren. Beim Queueing werden die einzelnen Datenpakete in eine Warteschlange (*Queue*) gereicht. Vom System wird dann entschieden, welches Paket zu welchem Zeitpunkt gesendet wird. Diese Entscheidung lässt sich mit PF beeinflussen. Standardmäßig verwendet OpenBSD FIFO (First-In-First-Out). Das bedeutet, dass jedes Paket, abhängig von der Eingangsreihenfolge abgearbeitet wird, unabhängig von der Wichtigkeit des Pakets. Ist die Queue voll, werden neu ankommende Pakete verworfen. Alternativ unterstützt OpenBSD noch zwei andere Scheduler:

- Class Based Queueing
- Priority Queueing

Beim Class Based Queueing (CBQ) wird die Bandbreite zwischen mehreren Warteschlangen oder Klassen aufgeteilt. Jeder Klasse wird Traffic nach bestimmten Kriterien zugeteilt. Die kann zum Beispiel die Zieladresse oder das verwendete Protokoll sein. Weiters wird jeder Queue eine gewisse Bandbreite und Priorität zugeteilt, so dass zum Beispiel wichtiger Datenverkehr immer Vorrang vor unwichtigerem hat. Warteschlangen beim CBQ sind

immer hierarchisch organisiert. So gibt es eine Parent Queue, die die gesamte verfügbare Bandbreite definiert. Dieser untergeordnet sind so genannte Child Queues, die sich die Bandbreite zu beliebigen Teilen untereinander aufteilen. Child Queues haben die Möglichkeit, sich Bandbreite von der Parent Queue zu *„borgen“*, wenn dies benötigt wird bzw. sofern welche zur Verfügung steht.

Beim Priority Queueing werden den unterschiedlichen Warteschlangen verschiedene Prioritäten zugeteilt. Eine Warteschlange mit höherer Priorität hat *immer* Vorrang. Erst wenn keine Pakete mehr in dieser Queue sind, kann die nächste an die Reihe kommen. Innerhalb einer Queue wird nach dem FIFO Prinzip gearbeitet.

Translation

Wie auch andere Firewall Lösungen, erlaubt PF mehreren privaten IP Adressen, mittels NAT den Zugang zum Internet über eine einzige öffentliche Adresse. NAT findet vor dem Filtern statt. Alle Pakete die von NAT verändert wurden, müssen trotzdem noch durch die Filter Engine, die im Folgenden beschrieben wird.

Filter Rules

Wie bei allen anderen Packet Filter Firewalls, werden auch bei PF eine Reihe von Regeln spezifiziert, mit denen Datenpakete verglichen werden. Trifft eine Regel zu, so kann das Paket abgelehnt oder zugelassen werden. Im Unterschied zu vielen anderen Firewalls, wie zum Beispiel iptables, wird ein Datenpaket bei PF mit *allen* Regeln verglichen, bevor eine Entscheidung getroffen wird. PF hat ein implizites *pass all* am Beginn. Das bedeutet, dass Pakete zugelassen werden, wenn keine passende Regel gefunden werden kann.
Die Syntax für PF Regeln lautet, vereinfacht:

```
action [direction] [log] [quick] [on interface] [af] [proto protocol] [from src_addr
[port src_port]] [to dst_addr [port dst_port]] [flags tcp_flags] [state]
```

Wobei `action` bezeichnet, was passieren soll, wenn eine Regel auf ein Paket zutrifft. `Direction` bezeichnet, in welche Richtung sich ein Paket bewegen muss, `in` oder `out`. `Log` gibt an, dass Informationen über ein Paket aufgezeichnet werden sollen, während `quick` besagt, dass nach dieser Regel keine weiteren Regeln mehr verglichen werden. `Interface` gibt an, auf welche Schnittstelle diese Regel angewandt werden soll, `af` definiert ob es sich um ein IPv4 oder ein IPv6 Paket handelt. Über den Parameter `protocol` wird angegeben, um welches Layer 4 Protokoll es sich handelt. Quell- und Zieladresse, sowie Ports, werden über die nächsten vier Parameter definiert. Schlussendlich kann noch auf gesetzte Flags überprüft werden, und definiert werden, welche Zustandsinformationen über ein Paket gespeichert werden sollen. Der letzte Punkt ermöglicht eine *Stateful Inspection* von Datenpaketen. (vgl. PF Manual 2003)

Fazit

OpenBSD's PF präsentiert sich als starke und sehr anpassungsfähige Firewall, die durch ihre flexible und gut verständliche Konfiguration beeindruckt. Im Unterschied zu iptables fehlt jedoch die Erweiterbarkeit durch Module, was PF in punkto Flexibilität etwas hinter iptables rückt. Die Möglichkeit, Tables zu verwenden, um große Mengen an IP Adressen zu speichern, ist jedoch sehr positiv zu bewerten, und bringt vor allem bei Firewalls großer Netze Geschwindigkeitsvorteile.

Firewall Fazit

Es gibt eine Vielzahl an Open Source Firewall Lösungen, und die oben angeführten Beispiele sind selbstverständlich nur ein Teil dessen, was am Markt zur Verfügung steht. Grob genommen lassen sich die Produkte in zwei Gruppen unterteilen. Jene, die ein graphisches Interface zur Konfiguration benutzen, und jene, die über CLI oder Konfigurationsdatei zu konfigurieren sind. Erstere eignen sich vor allem für Benutzer, die nicht sehr versiert mit Linux oder Firewalling sind, dennoch aber Sicherheit benötigen. So zum Beispiel kleinere Unternehmen, denen die finanziellen Mittel für eine eigene IT Abteilung oder für die Beauftragung externen Firmen fehlen. Ist allerdings höchstmögliche Sicherheit und Flexibilität ein Kriterium, so können sich die graphischen Varianten mit denen, die über die Shell zu konfigurieren sind, nicht messen. Die getesteten Produkte dieser Sparte, PF und iptables, sind beide hochgradig flexibel und vielseitig. Im Bereich der Paketmanipulation und -filterung sind diesen Produkten eigentlich keine Grenzen gesetzt. Allerdings bedarf es einer gewissen Expertise, um sie richtig konfigurieren zu können. Für Unternehmen mit höheren Sicherheitsansprüchen, deren Netzwerkstruktur veränderlich ist, sind diese flexibleren Lösungen die bessere Wahl. Kapitel 4 dokumentiert die praktische Implementierung von iptables, als repräsentatives Beispiel für Open Source Firewall Systeme.

3.3 Intrusion Detection Systeme

Der letzte Abschnitt hat sich mit dem Thema Firewalls befasst. Diese sind so genannte *Frontline Defense* Systeme. Es kommt jedoch vor, das diese Sicherheitsmaßnahme versagt. Firewalls sind oft so konfiguriert, dass das interne Netzwerk als vertrauenswürdige Zone angesehen wird. Kommt also ein Angriff von innerhalb des lokalen Netzwerkes, oder verschafft sich ein Angreifer Zugriff auf einen internen Rechner, so ist die Firewall als Schutzmaßnahme nutzlos. An dieser Stelle greifen Intrusion Detection Systeme (IDS) ein.
IDS sind Systeme, deren Aufgabe es ist, Netzwerke und Hosts zu überwachen und, im Falle einer Anomalie, Alarm zu schlagen. Man unterscheidet im Allgemeinen zwischen zwei Gruppen von Intrusion Detection Systemen, Host-Based IDS (HIDS) und Network-Based IDS (NIDS).

NIDS überwachen den Netzwerk Traffic, und versuchen, anhand von gewissen Merkmalen, Angriffe zu erkennen, während HIDS die Systemintegrität eines einzelnen Hosts überwachen.
Im Folgenden werden die zwei verschiedenen Arten von Intrusion Detection Systemen anhand einiger stellvertretender Produkte genauer untersucht.

3.3.1 NIDS

Aufgabe eines NIDS ist es, den Datenverkehr eines Netzwerks zu überwachen, und etwaiges unbefugtes Eindringen zu melden. Es gibt zwei Ansätze, anhand deren dies bewerkstelligt werden kann: Signature-Based NIDS und Anomaly-Based NIDS.
Signature-Based Systeme versuchen Angriffe mittels vorgegebener Signature zu identifizieren. Sie arbeiten ähnlich wie Anti-Viren Programme. Diese Form des NIDS ist die am weitesten verbreitete, da sich die meisten Angriffe als Signaturen darstellen lassen, und die Fehlerquote sehr gering ist.
Die zweite Gruppe, Anomaly-Based NIDS, versucht Angriffe aufgrund von Abweichungen im Datenverkehr zu erahnen. Dies hat den Vorteil, dass auch Angriffe, für die es noch keine Signaturen gibt, möglicherweise erkannt werden können. Großer Nachteil dieser Systeme

ist allerdings, dass es zu einer großen Anzahl von *false positives*, das sind Fehlalarme, oder auch *false negatives*, das sind Angriffe, die nicht erkannt werden, kommen kann.
Ein großer Vorteil von NIDS Systemen ist, dass ein einziger Sensor die Überwachung eines ganzen Netzwerksegments ermöglicht. Wird also ein System außer Gefecht gesetzt, existieren immer noch Aufzeichnungen über den Angriff am IDS Sensor. Dies kann allerdings auch dazu führen, dass das Netzwerk mehr Traffic produziert, als das NIDS verarbeiten kann. In diesem Fall werden Pakete verworfen und eine lückenlose Überwachung ist nicht mehr gewährleistet. In geswitchten Netzwerken muss der Switch so konfiguriert werden, dass der NIDS Sensor den gesamten Traffic dieses Netzwerksegments erhält.
Wenngleich auch Signature-Based NIDS weniger dazu tendieren, Fehlalarme auszulösen, so gibt es einige Fälle, die dazu neigen, zu so genannten *false positives* zu führen.

- **Netzwerk Monitor Aktivität** - Viele Netzwerk Monitoring Systeme (NMS) produzieren Verhaltensmuster, die von Intrusion Detection Systemen oft als verdächtiges Verhalten eingestuft werden,
- **Vulnerability Scanner** - Werden Port Scans oder Vulnerability Scans durchgeführt, um *Sicherheitsüberprüfungen* durchzuführen, wird ein NIDS dies als Attacke einstufen, da die verwendeten Tools dieselben sind, die auch von Eindringlingen verwendet werden.
- **Virusähnliches Verhalten** - Es kann vorkommen, das normale Programme ein Verhalten zeigen, das von einem NIDS mit dem Verhalten eines Virus verwechselt werden kann.

Um solche falschen Alarme zu minimieren, ist es notwendig, ein IDS an das entsprechende System anzupassen. Ein IDS, das mit Standardeinstellungen installiert wird, wird viele unnötige Warnungen liefern. Selbst gut konfigurierte Systeme, sollten jedoch regelmäßig überprüft und angepasst werden, um die Eindringlingserkennung so effizient wie möglich zu gestalten.
Der folgende Abschnitt beschäftigt sich mit Snort, einem der wichtigsten Vertreter von Network-Based Intrusion Detection Systemen. (vgl. Howlett 2004)

3.3.2 Snort

Snort ist mit über zwei Millionen Downloads und etwa 150.000 aktiven Benutzern weltweit das am häufigsten eingesetzte IDS. Snort kann Datenverkehr in Echtzeit untersuchen, und ist in der Lage, eine Anzahl von Attacken zu erkennen und zu melden.
Das Programm wurde 1998 von Martin Roesch als „leichtgewichtiges IDS (vgl. Sourcefire 2006) geschrieben. Im Laufe der Zeit hat sich Snort zu einem vollwertigen IDS entwickelt, das zu einem de facto Standard für Intrusion Detection Systeme geworden ist. Die Software wird unter der GPL Lizenz herausgegeben, es gibt allerdings auch eine kommerzielle Version, die, im Vergleich zur offenen Version, eine größere Anzahl an Funktionen besitzt. Anfang 2006 wollte Checkpoint Sourcefire, so der Name der Firma, die Snort herausgibt, kaufen, was jedoch scheiterte.
Im Folgenden sollen die einzelnen Funktionen von Snort genauer analysiert werden.

Funktionsumfang

Snort kann in drei verschiedenen Modi betrieben werden: Packet Sniffer Mode, Packet Logger Mode oder Intrusion Detection Mode

Packet Sniffer Mode Wird Snort im Packet Sniffer Mode ausgeführt, funktioniert es wie jeder andere Packet Sniffer. Alle Pakete, die am System ankommen, werden angezeigt. Wird nur ein Packet Sniffer benötigt, ist es natürlich nicht notwendig, Snort zu verwenden. In diesem Fall könnte auch ein einfacheres Programm, wie z.B. Ethereal oder tcpdump benutzt werden. Dieser Modus kann allerdings recht hilfreich sein, um zu überprüfen, ob Snort ordnungsgemäß funktioniert, also zum Beispiel, ob alle Pakete von Snort gesehen werden.

Packet Logger Mode Dieser Modus funktioniert im Prinzip analog zum Packet Sniffer Mode, mit dem Unterschied, dass die empfangenen Pakete zur späteren Analyse aufgezeichnet werden können. Um die Analyse einfacher zu gestalten, bietet Snort eine Reihe von Filtern an, mit denen bestimmt werden kann, welcher Traffic geloggt werden soll, und was verworfen werden kann. Zur besseren Übersicht, können gespeicherte Informationen auch nach IP Adresse oder nach lokalem und nicht lokalem Datenverkehr aufgezeichnet werden.

Intrusion Detection Mode Der Intrusion Detection Mode ist der eigentlich wichtige Modus von Snort. In diesem werden Pakete gespeichert und entsprechend den spezifizierten Parametern analysiert. Die Konfiguration dieser Parameter ist der wichtigste Schritt, um sicher zu stellen, das Snort sinnvoll arbeitet. Zwar können die Standardeinstellungen verwendet werden, allerdings wird dieser Ansatz eine große Anzahl an Fehlalarmen zum Resultat haben.

Wird ein Alarm ausgelöst, so gibt es verschiedene Möglichkeiten, wie Snort den Benutzer von diesem Vorfall informieren kann. Eine dieser Möglichkeiten, ist eine Windows Pop-Up Nachricht mittels SMB. Hierbei kann festgelegt werden, welche Hosts im Falle eines Alarms eine Nachricht erhalten sollen. Wird diese Art der Benachrichtigung verwendet, ist es allerdings notwendig, eine sehr gut eingestellte Konfigurationsdatei zu verwenden, da der Benutzer sonst mit einer Flut von Pop-Up Nachrichten überschwemmt wird.
Eine weitere Möglichkeit ist die Verwendung von Syslog. In diesem Fall sendet Snort eine Nachricht an einen Syslog Server. Die Verwendung von Syslog Servern erhöht die Sicherheit, da es einem Eindringling erschwert wird, seine Spuren zu verwischen. Die dritte Option, Alarme zu melden, sind Datenbanken. Snort bietet die Möglichkeit, Meldungen direkt an MySQL, PostgreSQL, Oracle oder unixODBC zu schicken.

Um Snort so zu konfigurieren, dass es verlässliche Alarme auslöst, und möglichst wenige *false positives* produziert, ist es notwendig, die Konfigurationsdatei genau auf die individuellen Anforderungen eines Netzwerks einzustellen. So ist es zum Beispiel notwendig, den IP Adressraum des lokalen Netzes zu definieren, oder die Adressen von lokalen Servern festzulegen. Weiters ist es notwendig, sämtliche Regeln zu deaktivieren, die nicht auf die individuelle Situation zutreffen. Snort bietet eine Reihe an vordefinierten Regeln für verschiedenste Angriffsszenarien. (vgl. Howlett 2004)
Um die Konfiguration von Snort zu erleichtern, bzw. es auch weniger versierten Benutzern zu erlauben, Snort zu verwenden, gibt es die Möglichkeit zur graphischen Konfiguration. Zwar bietet Snort selbst keine graphische Konfigurationsumgebung, jedoch ist die Konfiguration mittels eines Webmin Moduls möglich. Dieses Modul ermöglicht die Aktivierung bzw. Deaktivierung von Regeln durch einfaches Klicken.

Eine Dokumentation der praktischen Implementierung von Snort findet sich im praktischen Teil der Arbeit.

3.3.3 HIDS

Neben Network-Based Intrusion Detection Systemen gibt es auch noch Host-Based Intrusion Detection Systeme. Diese laufen direkt auf dem Host, der überwacht werden soll, und haben meist die Funktion eines *File Integrity Checker*. Zusätzlich haben HIDS Systeme den Vorteil, dass auch verschlüsselter Netzwerkverkehr analysiert werden kann, da sie sich ja bereits hinter dem Terminierungspunkt der Verschlüsselung befinden.
Kommt es zu einem Angriff, bei dem es dem Angreifer gelingt, in ein System einzudringen, so werden meist wichtige Systemdateien verändert. Dies geschieht meist, um dem Eindringling auch zukünftig den Zugang zu sichern, indem entweder Passwort Dateien verändert werden, oder neue Benutzer angelegt werden. Aufgabe eines HIDS ist es, diese wichtigen Systemdateien regelmäßig zu überprüfen, und zu warnen, wenn diese verändert wurden. Wie auch bei NIDS, muss ein HIDS sehr genau konfiguriert werden, um unnötige Alarme zu vermeiden.
Ein großer Nachteil von HIDS allerdings ist, dass der Eindringling, wenn er es geschafft hat, Zugang zum System zu bekommen, auch ohne weitere Probleme die Aufzeichnungen des HIDS verändern kann, und somit seinen Angriff verschleiern kann. Um dies zu vermeiden, ist es sinnvoll, die vom HIDS gesammelten Daten auf einem externen System abzulegen, oder, bei sehr sensiblen Systemen, die Daten direkt als Hardcopy auf einen Drucker auszugeben. Im nächsten Abschnitt wird die Funktionsweise von HIDS Systemen anhand einer stellvertretenden Software, Tripwire, analysiert.

3.3.4 Tripwire

Der wohl bekannteste Vertreter von HIDS ist Tripwire, welches als Open Source Version wie auch als kommerzielle Version zur Verfügung steht. Die kommerzielle Version ist für eine Anzahl an Plattformen erhältlich, während es die Open Source Version nur für Linux gibt. Wenngleich Erstere vom Leistungsumfang der Open Source Variante überlegen ist, so wird diese Arbeit dieser dennoch kein Augenmerk schenken, da die kommerzielle Version nicht unter einer Open Source Lizenz veröffentlicht wird.

Die Aufgabe von Tripwire ist es, eine Datenbank zu erstellen, die Informationen über vom User spezifizierte Dateien enthält und aufgrund dieser Datenbank regelmäßig zu überprüfen, ob die Dateien geändert wurden. Tripwire ist kein Echtzeit-Warnsystem. Es wurde nicht entworfen, um vor gerade ablaufenden Angriffen zu warnen, sondern vielmehr, um im Nachhinein herausfinden zu können, welche Schäden durch den Angriff entstanden sind. Dies erleichtert das Wiederherstellen des Ausgangszustandes des Systems und hilft, auf den nächsten Angriff besser vorbereitet zu sein.

Bevor Tripwire jedoch verwendet werden kann, ist es notwendig, die Konfiguration genau auf das zu überwachende System anzupassen. Mit Hilfe des Policy Files ist es möglich, Tripwire mitzuteilen, welche Dateien in die Datenbank aufgenommen werden sollen. Die Dateien können dabei auf eine Reihe von Attributen, auf die im praktischen Teil der Arbeit genauer eingegangen wird (siehe 4.6.1), überprüft werden.
Um Tripwire die Möglichkeit zu geben, Veränderungen zu erkennen, muss zuerst eine Referenzdatenbank erstellt werden. Dies *muss* auf einem sauberen System passieren, da Tripwire eventuelle Verletzungen nicht erkennen könnte, wenn diese bereits passiert wären. Sobald Tripwire läuft, ist es notwendig, die Referenzdatenbank regelmäßig zu aktualisieren, damit diese den aktuellen Zustand des Systems widerspiegelt und somit falsche Alarme eliminiert werden können. (vgl. Tripwire Inc. 2003)

3.3.5 Intrusion Prevention Systeme

Zusätzlich zu den bereits angesprochenen Varianten von Intrusion Detection Systemen kursiert heutzutage auch immer wieder der Name IPS. Diese so genannten Intrusion Prevention Systeme sind prinzipiell IDS, die zusätzlich noch die Möglichkeit bieten, in Echtzeit auf einen Angriff zu reagieren, und entsprechende Maßnahmen zu setzen. Realisiert wird dies, indem so genannte IDS Gateways mit Firewall Programmen zusammenarbeiten, und so dynamisch Regeln erstellen können. Allerdings befinden diese Funktionalitäten noch in einem sehr frühen Entwicklungsstadium und sind oft eher als Marketingstrategie zu verstehen, als als ernstzunehmende Funktion. (vgl. Cox, Gerg 2004)

Dennoch gibt es eine Hand voll Anwendungen, die Intrusion Prevention Mechanismen beinhalten. Eine dieser Anwendungen ist Snort Inline, die im Folgenden kurz beleuchtet werden soll.

Snort Inline

Snort Inline ist ein von Jed Haile geschriebenes Plug-In zu Snort, welches es dem Benutzer ermöglicht, IPS Funktionalitäten mit Snort zu realisieren. Um diese Funktion zu ermöglichen, muss Snort als so genanntes Inline IDS platziert werden. Das bedeutet, dass es auf einem System mit der Firewall betrieben werden muss, um zwischen dem Angreifer und dem angegriffenen Netzwerk platziert zu sein. Im Falle von Snort Inline, muss als Firewall iptables eingesetzt werden. Realisiert wird das IPS dann über das `QUEUE` target (siehe 3.2.2) von iptables, indem das Programm sämtlichen Datenverkehr an Snort weiterleitet. Snort Inline hat dann die Möglichkeit dynamisch, basierend auf den in Snort konfigurierten Regeln, Pakete zu verwerfen. (vgl. Sourcefire 2005)

Prinzipiell ist die Idee eines IPS sehr viel versprechend, und wird in Zukunft auch immer stärker präsent sein. Jedoch ist eine solche Funktionalität in der Praxis nicht ungefährlich, da durch *false positives* unter Umständen mehr Probleme geschaffen werden könnten, als verhindert werden.

IDS Fazit

Intrusion Detection Systeme sind ein wichtiger Bestandteil eines jeden Sicherheitssystem. Die beiden im letzten Abschnitt behandelten unterschiedlichen Typen von Intrusion Detection Systemen haben beide sowohl vor als auch Nachteile. So ist der Installationsaufwand eine NIDS in einem großen Netzwerk wesentlich geringer, als der eines HIDS. Andererseits wiederum können mit HIDS Systemen lokale Einbruchsversuche detektiert werden, was einem NIDS nicht möglich ist.
Grundsätzlich gilt es zu erkennen, dass die zwei analysierten Typen keine Gegensätze und keine Alternativen darstellen, sondern, ganz im Gegenteil, am effizientesten funktionieren, wenn sie parallel betrieben werden.

Aufgrund dieser Erkenntnis sollen im praktischen Teil auch beide Varianten implementiert werden.

3.4 Port Scanner

Port Scanner sind nützliche Analysetools, die verwendet werden, um festzustellen, auf welchen TCP oder UDP Ports ein System auf ankommende Verbindungen wartet. Dies kann

sinnvoll sein, um zu überprüfen, ob ein bereits abgesichertes System noch Lücken aufweist. Allerdings sind Port Scanner auch von Angreifern gern verwendete Werkzeuge, um festzustellen, auf welche Weise ein System angreifbar ist.
Port Scanner treten in verschiedensten Ausführungen auf. Einige von ihnen, wie zum Beispiel Nmap, auf das später in diesem Abschnitt eingegangen wird, sind sehr komplex und ermöglichen eine Vielzahl von verschiedenen Möglichkeiten, ein System zu scannen. Es gibt allerdings auch sehr einfache Varianten. So kann beispielsweise `telnet` als primitiver Port Scanner angesehen werden, da es ermöglicht, eine Verbindung zu einem beliebigen TCP Port aufzubauen, und feststellbar macht, ob ein System auf diesem Port Verbindungen akzeptiert.

Ein Port Scanner kann für verschiedenste Zwecke zum Einsatz kommen. So kann er zum Beispiel eingesetzt werden, um herauszufinden, wie viele Rechner sich in einem Netzwerk befinden, und auf welchen Dienste zur Verfügung stehen. Ein weiterer Einsatzbereich ist das Überprüfen und Optimieren von bestehenden Systemen. Ein Port Scanner kann zum Beispiel eingesetzt werden, um festzustellen, ob das betreffende System noch Lücken aufweist, und ob die Firewall richtig konfiguriert wurde. Außerdem sind Port Scanner auch nützlich, um Trojaner aufzuspüren, die auf einem System laufen und auf ankommende Verbindungen des Angreifers warten.
Neben all diesen sicherheitsverbessernden Maßnahmen, sind Port Scanner leider auch beliebte Werkzeuge für Angreifer, da sie ermöglichen, ein fremdes System auf Lücken zu überprüfen, die später für einen Angriff genutzt werden können

Im Folgenden wird einer der wichtigsten Open Source Port Scanner, Nmap, analysiert.

3.4.1 Nmap

Der wohl bedeutendste Open Source Port Scanner ist Nmap. Es wurde von Fyodor, nach eigener Definition ein Hacker (vgl. Fyodor), geschrieben, erstmals im September 1997 unter der GPL veröffentlicht, und ist inzwischen in der Version 4 erhältlich. Der Name *Nmap* steht für *Network Mapper*. Nmap wurde entwickelt um schnell große Netzwerke scannen zu können, jedoch ist es auch für den Scan eines einzelnen Hosts geeignet (vgl. Fyodor 2005). Das Programm hat sich in den letzten Jahren zu einem quasi Standard entwickelt, ist für eine Vielzahl von Plattformen erhältlich und wurde schon in viele verschiedene Programmiersprachen portiert. Nmap ist ein wichtiger Bestandteil vieler Netzwerkanalysetools. So wird es zum Beispiel vom Vulnerability Scanner *Nessus*, der später in diesem Kapitel analysiert wird, eingesetzt, um offene Ports ausfindig zu machen. (vgl. Howlett 2004)

Ein wichtiger Grund für die Popularität von Nmap ist die Flexibilität des Programms. Nmap bietet eine Vielzahl an Optionen, die es ermöglichen, sich jeder Situation anzupassen. Die Definition der einzelnen Zieladressen, sowie die Ports, die auf diesen Hosts überprüft werden sollen, ist sehr flexibel gestaltbar, und somit auch sehr effizient. Weiters erlaubt Nmap die Frequenz, in der ein System gescannt wird, also den zeitlichen Abstand, der zwischen den einzelnen Verbindungsversuchen liegt, festzulegen. Dies hilft, Bandbreitenengpässe zu vermeiden, kann aber auch verwendet werden, um den Port Scan möglichst unauffällig ablaufen zu lassen. Nmap unterstützt eine Reihe von so genannten *Stealth* Modi, die erlauben, einen Scan so ablaufen zu lassen, das dieser nur schwer als solcher zu erkennen ist. Wenngleich auch oft angenommen wird, dass diese Funktion nur von Hackern angewendet wird, so kann sie dennoch nützlich sein, wenn zum Beispiel festgestellt werden soll, wie sensibel ein IDS konfiguriert ist (vgl.). Eine weitere Funktion ist die so genannte *OS Detection*. So ermöglicht Nmap anhand gewisser Merkmale die Erkennung des Betriebsystems des

gescannten Hosts. Dies wird mittels einer Reihe von TCP und UDP Paketen bewerkstelligt, indem Nmap die Antworten des Hosts mit einer Datenbank vergleicht, und auf diese Weise einen relativ zuverlässigen Tipp über das verwendete Betriebsystem abgeben kann.

Wird ein Port Scan durchgeführt, so liefert Nmap eine Liste der gescannten Hosts, sowie aller Ports und deren Status. Nmap unterscheidet dabei zwischen vier möglichen Zuständen:

- **open** - Der Host akzeptiert aktiv Verbindungen auf diesem Port.
- **closed** - Der Port ist erreichbar, der Host akzeptiert jedoch keine Verbindungen.
- **filtered** - Es ist nicht möglich festzustellen, in welchem Zustand sich der Port befindet. Dies tritt auf, wenn der Scan durch eine Firewall gefiltert wird.
- **unfiltered** - Der Port ist erreichbar, jedoch kann Nmap nicht feststellen ob er sich im Zustand `open` oder `closed` befindet. Diese Klassifizierung tritt nur bei einem ACK Scan auf.

Neben den vier beschriebenen Zuständen, kennt Nmap noch die zwei *Zwischenzustände* `open|filtered` sowie `closed|filtered`. Diese treten auf, wenn Nmap nicht in der Lage ist, zu bestimmen, in welchem Zustand sich ein Port befindet.

Neben der Bestimmung des Zustandes eines Ports, ist es mit Nmap auch möglich zu bestimmen, welcher Dienst auf einem Port bereitgestellt wird. Oft ist es sogar möglich, die genaue Version des Dienstes in Erfahrung zu bringen. Dies ist sehr hilfreich, da die so genannten *Well Known Ports* nicht immer für die ihnen zugeteilten Dienste verwendet werden. Ist beispielsweise der TCP Port 80 auf einem Host im Zustand *open*, so ist dies noch keine Garantie dafür, dass es sich dabei um einen Webserver handelt. Nmap besitzt eine Datenbank mit etwa 1500 Diensten, auf Grund derer es bestimmen kann, welcher Dienst auf einem Port erreichbar ist.
Nmap unterstützt eine Reihe von unterschiedlichen Scan Methoden, die verschiedene Wege nutzen, um den Zustand eines Ports zu erkennen. Im Folgenden werden die wichtigsten kurz analysiert:

- **TCP SYN Scan** - Der TCP SYN Scan ist die Standardeinstellung von Nmap. Wird keine Scan Methode angegeben, so wird der SYN Scan verwendet. Im Unterschied zu TCP Connect Scan wird beim SYN Scan keine Verbindung aufgebaut, sondern nur ein SYN Paket gesendet. Antwortet die Gegenseite mit SYN/ACK, so steht fest, dass der angesprochene Port `open` ist. Ist die Antwort ein RST Paket, so ist der Port `closed`. Kommt nach mehreren Versuchen keine Antwort zurück, so wird der Port als `filtered` klassifiziert. Die SYN Scan Methode ist bevorzugt, da sie sehr schnell abläuft und relativ unauffällig ist, da nie eine vollständige Verbindung aufgebaut wird.
- **TCP Connect Scan** - Beim TCP Connect Scan wird versucht, mittels des `connect()` System Call, eine Verbindung zum Zielhost aufzubauen. Diese Methode ist langsamer und auffälliger als der SYN Scan. Sie wird standardmäßig angewandt, wenn ein SYN Scan aus benutzerrechtlichen Gründen nicht möglich ist.
- **UDP Scan** - Das scannen nach offenen UDP Ports ist sachgemäß eine schwierige Angelegenheit. Durch die Natur von UDP, das ein *verbindungsloses* Protokoll ist, ist es viel schwieriger herauszufinden, ob ein Port offen ist, oder nicht. Wird ein UDP Port gescannt so gib es drei Möglichkeiten. Schickt der Zielhost ein *ICMP Port unreachable* Paket zurück, so wird der Port als `closed` klassifiziert. Bei allen anderen *ICMP unreachable* Paketen wird der Port als `filtered` eingestuft. Kommt gar keine Antwort vom

Host zurück, so wird der Port als `open|filtered` angenommen. Bei UDP Scans ist es somit nie eindeutig feststellbar, ob ein gescannter Port wirklich offen ist. Die größte Herausforderung beim UDP Scan ist es, ihn möglichst schnell durchzuführen. Da vermeintlich offene Ports meist keine Antwort senden, muss stets ein gewisses Timeout abgewartet werden, was dazu führen kann, dass ein UDP Scan eines Systems bis zu mehreren Stunden in Anspruch nehmen kann.

- **TCP Null Scan** - Der TCP Null Scan, sowie auch die folgenden zwei Scan Methoden, nützen eine Schwachstelle in der TCP RFC aus, die besagt, dass die Antwort auf ein Paket, welches weder das SYN noch ACK oder RST Flag gesetzt hat, ein RST Paket sein muss, wenn der Port geschlossen ist, und gar keine Antwort erfolgen darf, wenn der Port offen ist. Beim TCP Null Scan wird keines der Flags gesetzt.
- **TCP FIN Scan** - Beim TCP FIN Scan wird, wie der Name besagt, nur das FIN Bit des Pakets gesetzt.
- **TCP Xmas Scan** - Der Name des Xmas Scan fußt darin, dass das Paket leuchtet wie ein Weihnachtsbaum, da sowohl das PSH, das URG und FIN Bit gesetzt sind. Der Vorteil der letzten drei genannten Scan Varianten ist, dass sie sogar noch unauffälliger sind, als die SYN Scan Variante, und in einem System ohne IDS im Allgemeinen nicht aufgezeichnet werden.
- **TCP ACK Scan** - Im Unterschied zu den bisher genannten Varianten, ist die Aufgabe des ACK Scan nicht, den Zustand eines Ports, also `open` oder `closed`, zu bestimmen, sondern nur herauszufinden, ob dieser erreichbar ist. Wird ein Port mit dieser Methode gescannt, so liefert ein nicht gefiltertes System in jedem Fall ein RST Paket zurück, egal ob der Port offen oder geschlossen ist. Der Port wird in diesem Fall von Nmap als `unfiltered` registriert. Kommt keine Antwort, oder eine ICMP Antwort zurück, so wird der Port als `filtered` erkannt.
- **TCP Window Scan** - Der TCP Window Scan entspricht dem TCP ACK Scan, mit dem Unterschied, das versucht wird, zwischen offenen und geschlossenen Ports zu unterscheiden. Die wird bewerkstelligt, indem die Window Size analysiert wird, die bei manchen Systemen positiv ist, wenn der Port offen ist, und Null, wenn er geschlossen ist.

Neben den genannten Methoden bietet Nmap noch die Möglichkeit, sämtliche TCP Flags selbst zu setzen und somit eigene Scan Varianten zu erschaffen. (vgl. Fyodor 2005)

Trotz dieser Vielfalt an Möglichkeiten, ist Nmap extrem leichtgewichtig. Das RPM Paket hat knapp unter einem Megabyte. Das Programm selbst läuft problemlos auf jedem Rechner, auch wenn dieser schon sehr alt ist.

Ein genauer Überblick über die Optionen von Nmap, sowie ein praktisches Anwendungsbeispiel findet sich in Kapitel 4.

3.4.2 Unicornscan

Unicornscan ist ein eher kleines Projekt, das im Unterschied zu Nmap nicht allzu bekannt ist. Nichtsdestotrotz ist Unicornscan ein vielseitiger Scanner, der eine Reihe interessanter Ansätze bietet. Das Programm wird von DYAD Security entwickelt, und unter der GNU GPL herausgegeben. Die ursprüngliche Idee für die Entwicklung von Unicornscan war, Probleme, die die Entwickler mit anderen Port Scannern erlebt hatten, zu beseitigen.

Eines dieser Probleme, unter dem viele andere Port Scanner leiden, sind UDP Scans. Wie bereits weiter oben bei Nmap erwähnt, besteht bei UDP Scans immer die Schwierigkeit, festzustellen, in welchem Zustand sich ein Port befindet. Da UDP von den Protokoll Spezifikationen her ein verbindungsloses Protokoll ist, gibt es auch keine Antwort, wenn auf einen Port verbunden wird. Darum ist es zum Beispiel bei Nmap auch nie möglich, einen UDP Port als `open` zu klassifizieren, sondern stets nur als `open|filtered`. Mit etwas Glück wird bei einem geschlossenen Port eine entsprechende ICMP Nachricht zurückgeschickt, so dass sich zumindest diese Ports zeitweise als `closed` einordnen lassen.
Um diesem Problem entgegenzuwirken, hat Unicornscan den Ansatz, nicht nur auf einen Port zu verbinden, sondern auch mit dem Dienst dahinter direkt zu kommunizieren. Da die Antwort auf eine ankommende Verbindung bei UDP eben nicht Sache des Protokolls ist, sondern dem Dienst vorbehalten bleibt, der auf dem entsprechenden Port läuft, versucht Unicornscan, einen Client zu simulieren, der mit diesem Dienst kommunizieren kann, und somit eine Antwort auszulösen. Dazu hat Unicornscan eine Datei, die eine Reihe von *Payloads*, also Nutzdaten verschiedenster Dienste, enthält. Auf diese Weise läst sich wirklich bestimmen, ob ein UDP Port offen oder geschlossen ist.

Ein weiterer interessanter Punkt von Unicornscan ist dessen Architektur. Das Programm verwendet eine so genannte *Scatter Connect* Technologie. Dabei wird der Scan nicht von *einem* Prozess durchgeführt, sondern auf drei Prozesse aufgeteilt:

- **Master Control Prozess** - Aufgabe des Master Control Prozesses ist es, zu koordinieren, welche Pakete gesendet und welche empfangen werden, die beiden Kindprozesse zu steuern, sowie den Zustand der Verbindungen zu überwachen.
- **Sender Prozess** - Der Sender Prozess ist dafür zuständig, Pakete an einen Host zu Senden
- **Listener Prozess** - Der Listener Prozess ist ein Sniffer, der auf ankommende Pakete wartet.

Wird ein Scan durchgeführt, so werden vom Master Control Prozess die beiden anderen Prozesse erzeugt und entsprechend koordiniert. Die Kommunikation zwischen den Prozessen läuft über IPC. Ziel dieser Technik soll es sein, das so genannte *Connection State Tracking* vom *Kernel Space* in den *User Space* zu verlagern, und somit die Geschwindigkeit des Scans zu erhöhen.

Ein weiterer Kritikpunkt der Entwickler von Unicornscan war, dass kein anderer Portscanner genau regelbare Timing Optionen besitzt. Bei Unicornscan ist es möglich, die Frequenz genau in *pps (packets per second*) zu regeln, und somit genau zu bestimmen, wie viel Bandbreite ein Scan verwenden darf.
Die sonstigen Optionen des Programms entsprechen in etwa denen, eines durchschnittlichen Port Scanners. So lässt sich zum Beispiel die Scan Methode, also TCP SYN Scan, TCP Connect Scan, Xmas Scan, etc. wählen, oder die gewünschte Source Adresse einstellen.

Im Unterschied zu Nmap besitzt Unicornscan eine Konfigurationsdatei. Ihr Zweck ist es, Standardoptionen festzulegen, die bei jedem Scan verwendet werden. So zum Beispiel die Scan Methode oder die Scan Frequenz. Wird über die Kommandozeile ein Scan initiiert, so werden alle Optionen, die nicht angegeben sind, aus der Konfigurationsdatei übernommen. Optionen, die über die Kommandozeile eingegeben werden, haben jedoch stets Vorrang.

Zusätzlich bietet Unicornscan ein auf PHP und SQL basiertes Web Frontend, das ermöglicht, die Scan Ergebnisse in einer Datenbank abzulegen. (vgl. Unicornscan)

Alles in allem ist Unicornscan ein recht umfangreiches Tool, mit einigen interessanten Ansätzen. Allerdings ist die Dokumentation praktisch nicht existent, und es ist nicht einfach, verwertbare Information über das Programm zu erhalten

Port Scanner Fazit

Port Scanner gehören zu den wichtigsten Tools, wenn es darum geht, die Sicherheit eines Systems zu überprüfen, und mögliche Lücken aufzuspüren. Dennoch ist die Zahl der Open Source Produkte sehr klein. Zwar gibt es insgesamt eine große Zahl an Port Scannern, und sogar einige kostenlose Produkte, jedoch nur eine handvoll werden unter einer Open Source Lizenz publiziert. Ein Grund dafür könnte sein, dass Nmap ohnehin einer der wichtigsten und am meisten verwendeten Produkte ist, und deshalb kein Bedarf an weiteren Open Source Port Scannern besteht.

Der nächste Abschnitt beschäftigt sich mit Vulnerability Scannern, die eine Weiterentwicklung von Port Scannern darstellen.

3.5 Vulnerability Scanner

Wie in 2.2.2 bereits beschrieben, gibt es eine Vielzahl an möglichen Bedrohungen, die einem Angreifer das Eindringen in ein System erleichtern können. All diese Sicherheitslücken manuell zu finden ist allerdings nur schwer möglich und würde viel Zeit und Ressourcen benötigen. Um diesem Problem Abhilfe zu schaffen, gibt es Vulnerability Scanner.
Vulnerability Scanner sind Tools, deren Zweck es ist, Schwachstellen in einem System aufzuzeigen. Im Unterschied zu Port Scannern, deren Funktion im vorhergehenden Abschnitt analysiert wurde, arbeiten Vulnerability Scanner nicht nur bis Layer 4 des OSI Modells, sondern bis Layer 7. Das bedeutet, dass auch Applikationsprotokolle, wie zum Beispiel HTTP oder SSH, auf mögliche Schwächen überprüft werden. Dies gibt die Möglichkeit, nicht nur offene Stellen am System zu erkennen, sondern auch den Dienst, der an dieser Stelle arbeitet, auf Lücken zu überprüfen und anhand der gesammelten Daten eine entsprechende Lösung zu erarbeiten.

Im Folgenden wird einer der umfangreichsten und am meisten verbreiteten Open Source Vulnerability Scanner auf seinen Funktionsumfang analysiert.

3.5.1 Nessus

Das Nessus Projekt wurde 1998 von Renaud Deraison als Open Source Projekt ins Leben gerufen. Seit Oktober 2005, seit der Version 3, jedoch, wird die Software unter einer proprietären Lizenz herausgegeben. Zwar ist das Programm immer noch kostenlos erhältlich, jedoch ist der Quellcode nicht mehr allgemein zugänglich. Nachdem die letzte offene Version, 2.2.8, immer noch von Tenable Network Security vertrieben wird, und Nessus einer der wichtigsten Vulnerability Scanner ist, soll das Programm in dieser Arbeit dennoch analysiert werden.

Nessus basiert auf Nmap, das weiter oben schon behandelt wurde. Im Unterschied zu Nmap ist Nessus jedoch nicht nur ein Port Scanner, sondern ein vielseitiger Vulnerability Scanner.

Zum Leistungsumfang von Nessus gehören über 2000 Tests aus verschiedenen Kategorien, die von Port Scans über DoS Vulnerabilities bis hin zu betriebsystemspezifischen Tests für Windows oder Cisco IOS hin reichen. Wird ein Scan durchgeführt, so hat der Anwender die Möglichkeit, nur die für das System relevanten Kategorien zu aktivieren. Wird zum Beispiel ein Linux System getestet, so wäre es nur wenig sinnvoll die Tests für Cisco IOS oder Microsoft Windows zu aktivieren. Zusätzlich können auch potentiell gefährliche Tests, die zu Denial of Service führen können, deaktiviert werden, um ein System nicht zum Absturz zu bringen. Dies ist wichtig, wenn an aktiven Produktionssystemen getestet wird. Prinzipiell hat der Anwender die Möglichkeit, jeden Test manuell zu aktivieren oder zu deaktivieren.
Zusätzlich zu den bereits vorhandenen Verwundbarkeitstests bietet Nessus ein Plug-In System, mit dessen Hilfe immer wieder neue Tests eingebunden werden können, um auf neu entdeckte Bedrohungen zu reagieren. So kann garantiert werden, dass die Software immer am aktuellen Stand ist, und auch mit neuen Verwundbarkeiten umgehen kann. Die Tatsache, dass Nessus unter einer Open Source Lizenz veröffentlicht wird, hat einen positiven Einfluss auf das Plug-In System, da jedem ermöglicht wird Plug-Ins zu erstellen, was dazu führt, dass auf neue Bedrohungen viel schneller reagiert werden kann, als eine einzelne Firma das könnte. Bei einem proprietären System liegt die Entscheidungsgewalt, ob ein Plug-In veröffentlicht werden soll, allein beim Herausgeber. Bei Open Source Produkten ist dies nicht der Fall, da nicht kontrolliert werden kann, ob jemand ein entsprechendes Plug-In veröffentlicht.
Eine zusätzliche Vereinfachung zum Plug-In System bietet NASL. NASL steht für Nessus Attack Scripting Language und ist eine relativ simple Skriptsprache, die der Erstellung neuer Plug-Ins dient, ohne auf dem Gebiet des Software Engineering besonders versiert sein zu müssen.

Architektur

Nessus basiert auf einer Client-Server Architektur, wobei der Server die ausführende Rolle und der Client die kontrollierende Rolle einnimmt. Soll ein Scan durchgeführt werden, so verbindet sich ein Client zu einem verfügbaren Nessus Server, und teilt diesem mit, was gescannt werden soll, und welche Tests durchgeführt werden sollen. Diese Architektur bringt einige Vorteile. So gibt es dem Anwender die Möglichkeit, ein Netzwerk, in dem er sich selbst befindet, von einer externen Perspektive aus zu scannen, sofern außerhalb ein Server zur Verfügung steht. Des Weiteren bringt eine Client-Server Architektur eine weitgehende Plattformunabhängigkeit, da immer nur die relativ einfache Client Software neu geschrieben werden muss. Momentan existieren Clients zwar nur für UNIX und Windows, da Nessus jedoch Open Source ist, steht es jedem Anwender frei, selbst einen Client für jedes beliebige Betriebsystem zu schreiben. (vgl. Howlett 2004)

Vulnerability Scanner Fazit

Vulnerability Scanner sind wichtige Werkzeuge beim Aufbau eines sicheren Systems. Ohne sie wäre es meist nicht möglich, sich gegen alle bekannten Bedrohungen abzusichern. Zwar stimmt es, dass auch mit Vulnerability Scannern nicht alle Probleme erkannt werden können, jedoch ist der durchschnittliche Angreifer meist eher unerfahren, und wird sich selbst auf ähnliche Werkzeuge stützen. So wird es eher selten vorkommen, dass ein System von einem wirklich erfahrenen *Hacker* attackiert wird, der Nachforschung anstellt, Social Engineering einsetzt, und sich wirklich mit dem anzugreifenden System auseinandersetzt. Das Gros der Angreifer sind so genannte *Script Kiddies*, die fertige Tools verwenden, und somit meist auf bekannte Schwachstellen abzielen. Somit ist die Wahrscheinlichkeit, dass eine

mögliche Lücke bereits erkannt und behoben wurde, relativ hoch, wenn Vulnerability Scanner eingesetzt werden.

3.6 Anti-Viren Software

Computerviren und -würmer stellen heutzutage ein immer größer werdendes Problem dar. In den letzten Jahren haben Viren und Würmer wie ILoveYou, CodeRed oder Sasser nicht nur Privatanwender terrorisiert, sondern auch wirtschaftlich Schäden in Milliardenhöhe, durch den Ausfall ganzer Firmennetzwerke, erzeugt, und es notwendig gemacht, sowohl einzelne Systeme, als auch ganze Netzwerke gegen Befall durch Viren abzusichern.

Um der Virenproblematik entgegenzuwirken, gibt es Anti-Viren Software. Darunter versteht man Programme, deren Aufgabe es ist, bekannte, aber auch unbekannte Viren, Würmer, oder Trojaner ausfindig zu machen, und wenn möglich zu entfernen.

3.6.1 Viren, Würmer und Trojaner

Wird von Computerviren gesprochen, so sind fälschlicherweise meist auch Computerwürmer und trojanische Pferde damit gemeint. Zwar gehören sie alle zur Gruppe der *Malware* und sind alle im Bezug auf Anti-Viren Programme relevant, dennoch gibt es deutliche Unterschiede, die im Folgenden aufgezeigt werden:

Virus Computerviren sind sich selbst reproduzierende Programme, die sich in andere Programme einschleusen, und sich dadurch vermehren. Der Begriff Virus entwickelte sich durch die Analogie zu biologischen Viren, die in fremde Zellen eindringen, und sich durch diese reproduzieren lassen. Ähnlich seinem biologischen Vorbild nützt auch der Computervirus die Ressourcen des befallenen Systems, und kann dadurch Schaden verursachen. Zusätzlich haben Viren meist vom Autor eingebaute Routinen, deren Zweck es ist, Schaden anzurichten. Diese Schäden können dabei von relativ harmlosen Darstellungsfehlern, bis hin zu Datenverlust oder gar Hardwareschäden führen. Im Alltag haben Computerviren heute jedoch kaum mehr Bedeutung. Ihr Platz wurde von Computerwürmern eingenommen, die im nächsten Absatz behandelt werden.

Wurm Ein Computerwurm ist ein Programm, das die Fähigkeit hat, sich selbst über Netzwerke zu verbreiten. Im Unterschied zum Virus benötigt der Wurm kein anderes Programm, sondern kann sich selbst, zum Beispiel durch den Versand infizierter Emails, verbreiten. Wurmprogramme haben oft gar keine eigenen Schadensroutinen, jedoch kann der durch die Verbreitung verursachte Ressourcenverlust schwere wirtschaftliche Schäden nach sich ziehen, indem zum Beispiel ganze Netzwerke vollkommen überlastet werden. Meist verbreiten sich Würmer über Emails, die infizierte Anhänge enthalten. Der Inhalt der Email ist meist so gewählt, dass ein Großteil der Empfänger dem Öffnen des Anhangs nicht widerstehen kann, und somit eine Infektion auslöst. Oft werden auch Lücken in Programmen genutzt, um die Verbreitung von Würmern zu vereinfachen.

Trojanisches Pferd Ein so genanntes *Trojanisches Pferd*, oder kurz *Trojaner*, ist ein Programm, das, als nützliche Anwendung getarnt, im Hintergrund andere, meist für den Nutzer negative Funktionen erfüllt. Den Namen hat diese Programmart aus der griechischen Mythologie, in der die Trojaner ein hölzernes Pferd zum Geschenk erhielten, in dessen Bauch sich der Feind versteckt hatte. Meist werden Trojaner verwendet, um Zugang zu einem fremden

System zu erlangen, mit der Absicht, dieses für meist kriminelle Machenschaften zu nützen, wie zum Beispiel für DDoS Attacken (siehe 2.2.2).

Wird im Folgenden von *Viren* gesprochen, so sind, sofern nicht anders vermerkt, alle oben stehenden *Malware* Typen gemeint.

3.6.2 Arbeitsweise von Anti-Viren Software

Um etwas gegen Viren unternehmen zu können, müssen diese zuerst aufgespürt werden. Anti-Viren Programme verwenden Listen mit Signaturen, das sind eindeutige Merkmale eines Virus, um bösartigen Code zu finden. Wird eine solche Signatur, oder Muster, in einer Datei gefunden, so wird diese von der Software als *infiziert* erkannt. Diese Listen müssen jedoch in regelmäßigen Abständen aktualisiert werden. Da fast täglich neue Viren, Würmer und Trojaner erschaffen werden, sind Anti-Viren Programme, deren Signaturlisten veraltet sind, praktisch nutzlos, und bieten bestenfalls noch einen Grundschutz. Das Herzstück einer Anti-Viren Software, ist die so genannte Scan Engine. Darunter versteht man jenen Teil des Programms, der für die Untersuchung des Systems zuständig ist. Die Leistungsfähigkeit eines Virenscanners wird maßgeblich von der Geschwindigkeit der Scan Engine beeinflusst.

Bei Host basierten Virenscannern ist zwischen verschiedenen Scan Techniken zu unterscheiden. So wird bei den meisten Anti-Viren Programmen ein Daemon ausgeführt, der so genannte Hintergrundscans durchführt. Dabei werden langsam aber stetig Dateien, die sich auf dem Host befinden nach *Malware* untersucht, und zusätzlich noch zugriffsbasierte Scans eingesetzt. Das bedeutet, dass jede Datei, die gelesen oder geschrieben wird, von der Scan Engine untersucht wird. Zusätzlich besteht auch meist die Möglichkeit, manuelle Scans durchzuführen, um das gesamte System, oder gezielt eine verdächtige Datei scannen zu können.
Neben Host basierten Anti-Viren Programmen, existieren auch Netzwerk basierte Varianten. Diese treten in gestalt von Content Scannern auf, die entweder auf der Firewall, oder in Verbindung mit einem Proxy laufen, und dort den gesamten Datenstrom nach bösartigem Code durchleuchten und gegebenenfalls blockieren. Eine weitere Möglichkeit besteht darin, ein Anti-Viren Programm mit einem Mailserver zu koppeln. Wie bereits erwähnt, breiten sich die meisten Viren und Würmer heutzutage über Emailverkehr aus. Wird ein Virenscanner direkt am Mailserver eingesetzt, so kann verhindert werden, dass sich infizierte Mails überhaupt ausbreiten können, indem verdächtige Anhänge bereits am Server entfernt werden.

Unabhängig vom Typ, also Host basierend oder Netzwerk basierend, arbeiten die meisten Anti-Viren Programme heutzutage *reaktiv*. Das bedeutet, dass zuerst eine Signatur eines Virus vorhanden sein muss, bevor dieser erkannt werden kann. Durch die immer größer werdende Bedrohung durch *Malware* ist allerdings zu erwarten, dass zukünftige Viren Scanner vermehrt auch *proaktiv* arbeiten werden. Proaktiv bedeutet, dass auch unbekannte Viren erkannt werden können. Dies kann zum Beispiel durch die Beobachtung des Verhaltens eines Programms, oder durch Erkennen allgemeiner Virusmerkmale erreicht werden. Diese Technologien befinden sich allerdings momentan in einem sehr frühen Entwicklungsstadium und sind meist noch sehr unzuverlässig.

Wird von einem Anti-Virus System, egal ob proaktiv oder reaktiv, netzwerk- oder hostbasierend, eine infizierte Datei gefunden, so gibt es prinzipiell drei Möglichkeiten, wie mit dieser Datei verfahren werden kann.

- **Bereinigen** - Der Viruscode wird von ursprünglichen Programmcode entfernt. Das Programm ist wieder in seinem Ausgangszustand und der Virus ist beseitigt. Diese Methode ist allerdings nicht bei allen Viren anwendbar.
- **Quarantäne** - Die meisten Anti-Viren Programme bieten einen so genannten Quarantäne Ordner. Wird eine infizierte Datei gefunden, so kann sie in diesen Ordner verschoben werden und der Zugriff wird verweigert.
- **Löschen** - Die letzte Möglichkeit ist, die gesamte infizierte Datei zu entfernen. Zwar geht dabei das Programm verloren, allerdings wird auch der Virus effektiv beseitigt.

Im Folgenden werden zwei Open Source Anti-Viren Programme, ClamAV und OpenAntiVirus, genauer betrachtet.

3.6.3 ClamAV

ClamAV ist ein Open Source Virenscanner, und wird unter der GNU GPL herausgegeben. Das Programm wird, aufgrund seiner Unterstützung für mbox und Maildir, häufig als Email Scanner auf Mailservern eingesetzt. Außerdem unterstützt ClamAV das Durchsuchen aller gängigen Archive, wie zum Beispiel Zip, RAR, Tar, oder GZip. Wie auch die Software selbst, sind Updates der *Virus Definition Database*, also der Liste mit Virensignaturen, gänzlich kostenfrei.

Neben einem *Command Line Scanner* bietet ClamAV auch einen *Daemon*, also ein Programm, das im Hintergrund ausgeführt wird, der nach Viren sucht, sowie einen *Daemon*, der für automatische Updates der Signaturenliste zuständig ist. Außerdem enthält das Paket auch eine Library für C, die es Entwicklern ermöglicht, Virenschutz in eigene Software Produkte einzubauen. Entsprechende Software muss, nach den Regeln der GPL, natürlich auch wieder unter dieser Lizenz veröffentlicht werden.

Das Tool für automatische Updates, `freshclam`, lässt sich auf mehrere Arten bedienen. Entweder interaktiv über ein CLI, als Daemon, der automatisch im Hintergrund in regelmäßigen Abständen nach Updates sucht, oder mittels *cronjob* mit der Option `-quiet`, die sämtliche Ausgaben des Programms unterdrückt.
Der Daemon für Hintergrundscans, clamd, lässt sich wahlweise über TCP, UNIX Socket oder über einen entsprechenden Client ansprechen. Durch die Installation eines Drittanbietermoduls lassen sich auch *On-Access* Scans realisieren. Vor dem ersten Ausführen, muss jedoch eine entsprechende Konfigurationsdatei erstellt werden. Wird über TCP, also zum Beispiel mittels `telnet` zugegriffen, so bietet `clamd` eine Reihe von Befehlen:

- **PING** - Status des Daemons überprüfen.
- **VERSION** - Gibt die Version des Programms und der Datenbank aus.
- **RELOAD** - Lädt die Datenbank neu.
- **SHUTDOWN** - Beendet den Daemon.
- **SCAN Datei/Verzeichnis** - Überprüft eine angegebene Datei oder ein Verzeichnis rekursiv.
- **RAWSCAN Datei/Verzeichnis** - Wie SCAN, jedoch mit deaktivierter Archiv Unterstützung.

- **CONTSCAN Datei/Verzeichnis** - Wie SCAN, jedoch wird der Scanvorgang nicht unterbrochen, wenn ein Virus gefunden wird.
- **STREAMScan**- `clamd` gibt eine Portnummer zurück, an die zu überprüfende Dateien geschickt werden sollen.
- **SESSION, END** - Beendet eine `clamd` Session.

Zusätzlich gibt es noch eine Vielzahl an Drittanbietermodulen, die die Zusammenarbeit mit den verschiedensten Programmen, wie zum Beispiel MTAs oder Proxys, erlauben.
(vgl. Kojm 2006)

3.6.4 OpenAntiVirus

Das OpenAntiVirus Projekt wurde im August 2000 ins Leben gerufen, mit dem Ziel, eine Plattform zur Kommunikation für Entwickler zu gründen, die als Ergebnis Anti-Viren Software für private Nutzer und Unternehmen haben soll. Bedauerlicherweise hat das Projekt in den letzten Jahren keine großen Fortschritte gemacht, und nach eigenen Angaben wäre es nicht ratsam, OpenAntiVirus zu verwenden, um ein System abzusichern (*„No one serious about the security of their systems will honestly use OAV as the only means to protect them“*, (OAV Crew 2006)).

ScannerDaemon ScannerDaemon ist der erste Virenscanner, der im Zuge des OpenAntiVirus Projekts erstellt wurde. Die Software wurde gänzlich in Java programmiert und befindet sich momentan im Beta Stadium. ScannerDaemon ist, wie der Name verrät, ein Daemon, der grundlegende Anti-Viren Fertigkeiten bietet, und Archive untersuchen kann, allerdings teilweise noch sehr fragmenthaft ist. So werden zum Beispiel Viren mit polymorphem Code von der Engine noch nicht erkannt. Nachdem das OpenAntiVirus Projekt im Moment jedoch anscheinend nicht weiterverfolgt wird, gibt es auch nur sehr wenig Information oder Dokumentation über ScannerDaemon. Das Programm wurde zum letzten Mal im Mai 2004 aktualisiert.

VirusHammer VirusHammer ist ein ebenfalls in Java geschriebenes, stand-alone Anti-Viren Programm, das, wie ScannerDaemon, ebenfalls aus dem OpenAntiVirus Projekt hervorgegangen ist. Es unterstützt einfaches Scannen von Dateien, nicht aber das Untersuchen von Archiven. Auf der OpenAntiVirus Website (`www.openantivirus.org`) befindet sich eine Onlineversion des Tools. Die letzte verfügbare Version des Programms ist allerdings bereits über vier Jahre alt, die aktuellsten Virus Definitions stammen aus 2004. Trotz intensiver Recherche war es nicht möglich, in Erfahrung zu bringen, ob das Projekt überhaupt noch weiterentwickelt wird. Selbst wenn dies der Fall ist, wird es keinen brauchbaren Schutz liefern, da es keinen Virus, der nach Mai 2004 erschaffen wurde, entdecken könnte.

Alles in allem scheint sich das OpenAntiVirus Projekt bis dato eher als ein Misserfolg herauszustellen. Es darf allerdings nicht außer Acht gelassen werden, dass sich das Projekt selbst als noch sehr weit entfernt vom Ziel beschreibt und nach eigenen Angaben bis jetzt nur ein *„set of toys“* (OAV Crew 2006) hervorgebracht hat, und noch lange keine nutzbaren Werkzeuge. Dennoch ist fraglich, ob ein Projekt, in dem seit zumindest zwei Jahren keine Bewegung mehr steckt, noch Potential zur Fertigstellung hat. (vgl. Link 2006)

Anti-Viren Software Fazit

Im Unterschied zu allen anderen bisher analysierten Sicherheitstechnologien, sind im Bereich der Anti-Viren Software Open Source Produkte nicht sehr gut vertreten. Auch nach längeren Recherchen konnten keine anderen als die zwei untersuchten Programme gefunden werden. Warum dies so ist, konnte nicht geklärt werden. Wie ClamAV eindrucksvoll zeigt, ist es offensichtlich nicht unmöglich, ein vollwertiges Anti-Viren Programm unter einer Open Source Lizenz zu publizieren. Allerdings steht dieses Programm ziemlich alleine da. OpenAntiVirus befindet sich momentan bestenfalls im Alpha Stadium, ohne Aussicht auf Veränderung.

3.7 Verschlüsselung

Alle bisher analysierten Sicherheitstechnologien bezogen sich darauf, Eindringlinge von einer gesicherten Umgebung fern zu halten. Der folgende Abschnitt beschäftigt sich damit, was mit Informationen passiert, die sich außerhalb dieser geschützten Umgebung bewegen.

Werden Informationen über ein öffentliches Netz, wie das Internet, versendet, so ist es nahezu unmöglich, zu kontrollieren, wer Zugang zu diesen Informationen erlangt. Da viele der im Internet üblichen Protokolle (HTTP, FTP, SMTP, etc.) Informationen in Klartext (*engl. plain text*) übertragen, wäre es für jeden, der Zugang zu den Informationen erhält, möglich, diese auszulesen. Um dies zu verhindern, werden in vielen Unternehmen Verschlüsselungstechnologien eingesetzt.

3.7.1 Kryptographie

Die Kryptographie ist die Wissenschaft der Verschlüsselung von Informationen. Sie ist ein Teilgebiet der Kryptologie und das Gegenstück zur Kryptoanalyse. Das Wort stammt aus dem griechischen und bedeutet soviel wie *verborgen schreiben*. Aufgabe der Kryptographie ist es, den Inhalt einer Nachricht für Dritte unverständlich zu machen, und nicht, die Nachricht an sich zu verbergen. Wenngleich die Kryptographie schon seit langer Zeit eine mathematische Disziplin war, entwickelte sie sich erst im 20.Jahrhundert, vor allem unterstützt durch elektronische Rechenhilfen, zu einer allgegenwärtigen Wissenschaft.

Die Kryptographie unterscheidet zwischen so genannten *Codes* und *Ciphers*. Bei einem Code wird ein Wort mit einem anderen oder einem Symbol vertauscht. Auch wenn Codes in der Menschheitsgeschichte lange, und selbst bis vor kurzer Zeit, eine große Bedeutung hatten, werden sie heute kaum mehr verwendet. Cipher dagegen, sind Bit-für-Bit Transformationen, ohne Rücksicht auf linguistische Strukturen.

Die Ziele modernen Kryptographie lassen sich in den folgenden drei Begriffen zusammenfassen:

- **Confidentiality** - Der Grundsatz der Vertraulichkeit bezieht sich darauf, dass nur der Empfänger in der Lage sein sollte, den Inhalt einer verschlüsselten Nachricht zu lesen. Dies muss auch zutreffen, wenn die Nachricht über ein nicht sicheres Medium verschickt wird. Potentielle Angreifer sind dann zwar in der Lage, die Nachricht zu sehen, jedoch nicht, den Inhalt der Nachricht zu entschlüsseln.

- **Authentication** - Dem Empfänger muss es möglich sein, festzustellen, ob die Nachricht tatsächlich vom vorgegebenen Absender stammt.
- **Integrity** - Die Datenintegrität muss gewahrt sein. Das bedeutet, dass der Empfänger in der Lage sein muss, festzustellen, ob die Nachricht seit dem Versenden verändert wurde.
- **Non-repudiation** - Dem Absender darf es nicht möglich sein, zu bestreiten, dass er eine bestimmte Nachricht gesendet hat.

(vgl. Chandra et al. 2002)

3.7.2 Kryptographische Methoden

Wird eine Klartextnachricht (*engl. plain text*) durch eine von einem Schlüssel (*engl. key*) abhängige Funktion verschlüsselt, so ist das Ergebnis ein Geheim- oder Schlüsseltext (*engl. ciphertext*). Wird diese Nachricht nun übertragen, und von einem Dritten abgefangen, so kann dieser die Nachricht nicht entschlüsseln, wenn er den entsprechenden Schlüssel nicht besitzt. Wird der Ciphertext durch $C = E_K(P)$ dargestellt, und der Plaintext durch $P = D_K(C)$, wobei E_K und D_K die Ver- und Entschlüsselungsfunktionen repräsentieren, so folgt daraus, dass $D_K(E_K(P)) = P$.

E_K und D_K sind mathematische Funktionen, die beide vom Parameter K, dem Schlüssel, abhängig sind. Eine grundlegende Regel moderner Kryptographie besagt, dass davon auszugehen ist, dass der Algorithmus öffentlich bekannt ist, und somit der Schlüssel geheim sein muss. Diese Aussage wird als Kerckhoffs-Prinzip, nach dem holländischen Kryptologen Auguste Kerckhoffs, bezeichnet. Ausgehend von diesem Prinzip und von der Tatsache, dass es äußerst aufwändig wäre, jedes mal einen neuen Algorithmus zu entwickeln, wenn dieser bekannt würde, ist es heute üblich, dass der Mechanismus öffentlich, der Schlüssel jedoch geheim ist. Die gegenteilige Vorgehensweise, also den Algorithmus geheim zu halten, wird als *Security by Obscurity* bezeichnet, und hat sich in der Vergangenheit als nicht effektiv erwiesen. Zusätzlich hat ein öffentlicher Algorithmus den Vorteil, dass er, ähnlich Open Source Software, von vielen Wissenschaftlern untersucht, und somit leichter und schneller verbessert werden kann.

Prinzipiell gibt es zwei verschiedene Arten von Verschlüsselungsverfahren, symmetrische und asymmetrische. Die folgenden Abschnitte widmen sich diesen unterschiedlichen Verfahren.

Symmetrische Schlüssel

Symmetrische Verschlüsselungsverfahren sind klassische Verschlüsselungsmethoden, wie sie schon seit Jahrtausenden verwendet werden. Sie basieren auf einem gemeinsamen geheimen Schlüssel (*engl. shared secret*), der sowohl der Person, die die Nachricht verschlüsselt, als auch der Person, die die Nachricht wieder entschlüsseln will, bekannt sein muss. Bei symmetrischen Verfahren wird, ähnlich einem Schloss, der gleiche Schlüssel zum Ver- und Entschlüsseln verwendet.
Man unterscheidet bei symmetrischen Verfahren zwischen so genannten *Blockcipher* und *Streamcipher*. Streamcipher verschlüsseln ein Zeichen nach dem anderen, während Blockcipher Gruppen von Zeichen als eine Einheit verschlüsseln. Symmetrische Verschlüsselungsverfahren reichen schon bis in die Zeit von Julius Cäsar zurück, nach dem heute noch

der *Cesar cipher* benannt ist. Im Folgenden werden drei modernere Verfahren kurz analysiert.

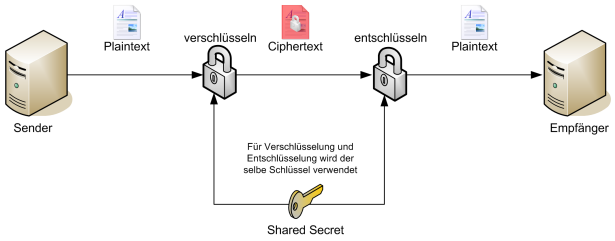

Abbildung 3.7: Symmetrische Verschlüsselung

Data Encryption Standard (DES) Das in den 1970er Jahren von IBM entwickelte und 1976 von der US Regierung als Standard bestätigte Verschlüsselungsverfahren galt mit seinem 56bit Schlüssel damals als unknackbar. Wegen seiner geringen Schlüssellänge und durch den ständigen Leistungsanstieg moderner Computer, gilt DES heute als nicht mehr sicher, wird aber dennoch in System mit geringfügiger Sicherheit eingesetzt. Der DES Algorithmus basiert auf 64bit großen Klartextblöcken, und besteht aus neunzehn Stufen. Zu Beginn und am Ende werden die Zeichen jedes Blocks nach einem festen Muster transponiert. Zusätzlich werden zum Schluss noch die ersten 32bit mit den letzten 32bit vertauscht. Die verbleibenden sechzehn Schritte sind Funktionen, bei denen die 64bit Blöcke mit dem geheimen Schlüssel durch XOR vermischt werden, wobei bei jedem Durchlauf eine andere Permutation des Schlüssels verwendet wird. Die Entschlüsselung basiert auf denselben Prinzipien, nur wird das System in entgegengesetzter Richtung durchlaufen.

Triple DES (3DES) Triple DES wurde bereits 1979 entwickelt und basiert auf dem DES Algorithmus. Wie in Abbildung 3.8 dargestellt werden beim 3DES drei Stufen und zwei 56bit Schlüssel verwendet. Die erste Stufe verschlüsselt mit K1, die zweite entschlüsselt mit K2 und die dritte Stufe verschlüsselt wieder mit K1. Der Grund für dieses EDE (*Encrypt Decrypt Encrypt*) Verfahren, war die Abwärtskompatibilität zu alten DES Systemen. Um diese zu erreichen muss nur K1=K2 gesetzt werden.

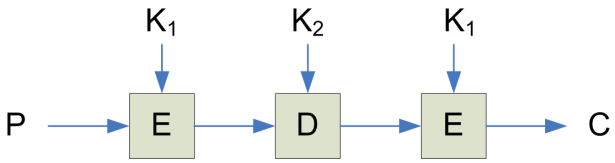

Abbildung 3.8: Triple DES

Advanced Encryption Standard (AES) Aufgrund der wachsenden Unzulänglichkeiten von DES, beschloss die US Regierung im Jänner 1997 eine Ausschreibung für einen neuen Verschlüsselungsstandard. Der Zuschlag ging letztlich an zwei belgische Kryptologen, die, auf Basis ihres Algorithmus Rijndael, AES entwickelten. AES bietet eine Schlüssellänge mit bis zu 256bit, was auch für die nähere Zukunft ausreichend sein dürfte.

Ein großer Vorteil von symmetrischen Algorithmen ist der relativ geringe Rechenaufwand, und folglich, die hohe Geschwindigkeit, mit der ver- bzw. entschlüsselt werden kann. Das grundlegende Problem mit dieser Methode, ist jedoch die sichere Übertragung des Schlüssels. Fällt der geheime Schlüssel in die Hände eines Dritten, so ist dieser in der Lage, verschlüsselte Nachrichten zu lesen. Abhilfe zu diesem Problem schufen Whitfield Diffie, Martin Hellman und Ralph Merkle, durch die Erfindung von so genannten *Public Key* Verfahren, mit denen sich der nächste Abschnitt befasst.

Asymmetrische Schlüssel

Bei asymmetrischen, oder *Public Key* Verfahren, wird der Schlüssel in zwei kleinere Schlüssel unterteilt, von denen einer öffentlich ist, und einer geheim bleibt. Soll eine Nachricht verschlüsselt werden, so wird dies mit dem öffentlichen Schlüssel (*Public Key*) des Empfängers gemacht. Dieser wiederum kann die verschlüsselte Nachricht mit seinem Geheimschlüssel (*Private Key*) entschlüsseln. Der große Unterschied zu symmetrischen Verfahren ist hierbei, dass der private Schlüssel nie übertragen werden muss und somit keine Möglichkeit besteht, dass er in falsche Hände fällt.
Die beiden Schlüssel werden dabei mit so genannten *One Way* Verfahren erzeugt. Dabei werden Funktionen verwendet, die, zumindest aus heutiger Sichtweise, unumkehrbar sind. Zwar ist es meist theoretisch möglich, den *Private Key* aus dem *Public Key* zu errechnen, allerdings bedürfte es dazu mit heutigen Mitteln mehrerer Millionen Jahre.

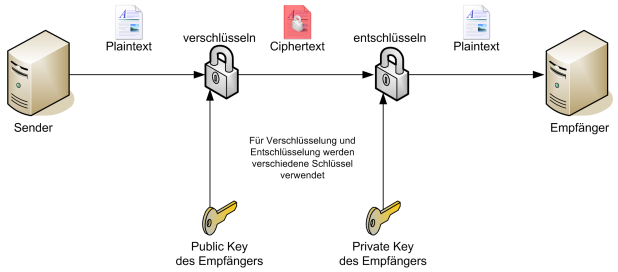

Abbildung 3.9: Asymmetrische Verschlüsselung

Trotz aller Vorteile, die asymmetrische Verschlüsselungsverfahren mit sich bringen, bergen sie auch einige Nachteile in sich. So ist der Rechenaufwand, der zur Ver- bzw. Entschlüsselung nötig ist, für die meisten Anwendungen nicht praktikabel, was es unmöglich macht, größere Nachrichten mit asymmetrischen Verfahren zu verschlüsseln. Diesem Problem kann jedoch mit der Verwendung hybrider Verfahren, die im nächsten Abschnitt behandelt werden, entgegengewirkt werden. Ein weiterer Nachteil von *Public Key* Verfahren ist das Problem mehrerer Empfänger. Wird eine Nachricht an mehrere Personen gesendet, so müsste sie auch mehrmals verschlüsselt werden, da ja immer der öffentliche Schlüssel des jeweiligen Empfängers zur Verschlüsselung verwendet werden muss. Auch dieses Problem lässt sich durch Anwendung hybrider Verfahren umgehen. Ein weitaus größeres Problem ist die Verwendung von *One Way* Funktionen. Sollte in Zukunft ein Verfahren entwickelt werden, die die Umkehrung einer solchen Funktion auf leichte Weise ermöglicht, so wären alle Algorithmen, die auf diese Funktionen vertrauen plötzlich nutzlos. Schlussendlich bleibt auch bei asymmetrischen Verfahren das Risiko einer *Man In The Middle* Attacke. Wird dem Sender von einem Mittelsmann ein falscher öffentlicher Schlüssel zugespielt, so hat der Angreifer die Möglichkeit, die gesendete Nachricht zu entschlüsseln, sie zu lesen oder gar zu verändern, und anschließend mit dem tatsächlichen öffentlichen Schlüssel verschlüsselt an den

ursprünglichen Empfänger weiterzuleiten. Dieses Problem lässt sich jedoch durch die Verwendung von Zertifikaten beseitigen.

RSA Bereits kurze Zeit nachdem Diffie, Hellman und Merkle ihre Arbeit über asymmetrische kryptographische Verfahren veröffentlicht hatten, entwickelten Ronald Rivest, Adi Shamir und Leonard Adleman eine praktische Anwendung der Theorie, den Verschlüsselungsalgorithmus RSA.
Das Verfahren wurde 1977 entwickelt und basiert bei der Schlüsselerzeugung darauf, das die Faktorisierung großer Zahlen mit heutigen mitteln nicht in einer vernünftigen Zeitspanne bewältigbar ist, das Erzeugen dieser Zahl durch Multiplikation jedoch sehr einfach ist. Der Algorithmus besteht grundlegend aus vier einfachen Schritten:

- Es werden zufällig zwei unterschiedliche Primzahlen p und q gewählt und das Produkt $N = p \cdot q$ berechnet
- Anschließend wird $\varphi(N) = (p-1) \cdot (q-1)$ berechnet
- Danach wird eine Zahl e gewählt, für die gilt $1 < e < \varphi(N)$ und die teilerfremd zu $\varphi(N)$ ist
- Zum Schluss wird eine Zahl d gewählt, für die $e \cdot d \equiv 1 \pmod{\varphi(N)})$

Der öffentliche Schlüssel besteht folglich aus N, dem Primzahlenprodukt, sowie aus e, dem öffentlichen Exponenten. Der private Schlüssel besteht auch aus N, und aus dem privaten Exponenten d. p, q und $\varphi(N)$ werden nach der Schlüsselerstellung nicht mehr benötigt und können gelöscht werden.

Soll nun eine Nachricht mit RSA verschlüsselt werden so wird auf Seite des Senders, mit Hilfe des öffentlichen Schlüssels e und N, $C = P^e \pmod{N}$ zur Erstellung des Ciphertextes angewandt. Der Empfänger kann nun unter Zuhilfenahme seines privaten Schlüssels d und N, $P = C^d \pmod{N}$ anwenden, und den Klartext aus der verschlüsselten Nachricht wiederherzustellen.

RSA ermöglicht auch das digitale Signieren von Nachrichten. Um eine Signatur zu erstellen, muss die Nachricht einfach mit dem privaten Schlüssel des Senders verschlüsselt, und der Nachricht beigefügt werden. Der Empfänger kann diese dann mit Hilfe des öffentlichen Schlüssels des Senders entschlüsseln und mit der eigentlichen Klartext Nachricht vergleichen. Stimmen die beiden Nachrichten überein, so ist gesichert, dass die Nachricht von demjenigen signiert wurde, der den privaten Schlüssel besitzt, und dass diese seitdem auch nicht verändert wurde. So werden die Prinzipien der Authentizität sowie der Integrität gewahrt, sofern davon ausgegangen werden kann das der private Schlüssel geheim geblieben ist und nur dem Sender bekannt war.
In der Praxis wird jedoch meist nicht die gesamte Nachricht verschlüsselt, um eine digitale Signatur zu erzeugen, sondern, um Rechenzeit zu sparen, nur ein *Hash Wert* (siehe 3.7.2) der Nachricht. Dieser Hash wird der Nachricht beigefügt, und kann vom Empfänger mit dem von ihm erzeugten Hash verglichen werden.

RSA hat, wie alle asymmetrischen Verfahren, den großen Nachteil, dass es für die Anwendung auf große Datenmengen viel zu langsam ist. Der Algorithmus wird allerdings bei hybriden Verfahren, denen sich der nächste Abschnitt widmet, zum Schlüsselaustausch eingesetzt.

Hybride Verschlüsselung

Hybride Verschlüsselungsverfahren sind solche, bei denen eine Kombination aus symmetrischen und asymmetrischen Verfahren zum Einsatz kommt. Dabei wird in der Regel ein asymmetrisches Verfahren verwendet, um einen sicheren Kanal zur Übertragung des Schlüssel zu schaffen, der dann für das symmetrische Verfahren genutzt werden kann, welches zur Verschlüsselung der Nutzdaten dient. Auf diese Weise werden die Vorteile beider Verfahren ausgenutzt: Der sichere Schlüsselaustausch der asymmetrischen Verschlüsselung, und die hohen Geschwindigkeiten von symmetrischen Verfahren. Ein bekanntes Beispiel für ein hybrides Verfahren ist PGP, welches weiter unten im Kapitel behandelt wird.

Hash Funktionen

Eine Hashfunktion ist eine Funktion, die meist eine große Datenmenge als Eingabe hat, und eine deutlich kleinere, den Hashwert, als Ausgabe. Ein guter Hash Algorithmus zeichnet sich durch eine möglichst geringe Zahl an Kollisionen aus. Das bedeutet, dass möglichst wenige nicht identische Eingaben den gleichen Hashwert zum Ergebnis haben. Hashfunktionen sind Einwegfunktionen, das bedeutet, das sich aus dem Hashwert der ursprüngliche Eingabewert nicht, oder nur unter sehr großem Aufwand errechnen lässt. Somit ist es möglich, einen mehr oder weniger eindeutigen *Fingerabdruck* einer Datei zu erzeugen. In der Informationssicherheit werden Hashfunktionen oft eingesetzt, um Passwörter zu speichern, was den Vorteil hat, dass nur die Hashwerte von Passwörtern, und nie die Passwörter selbst gespeichert werden müssen. Zwar sind Hashfunktionen per Definition keine Verschlüsselungsfunktionen, da sie nicht umkehrbar sind ($D_K(E_K(P)) = P$, siehe 3.7.2), dennoch haben sie in der Kryptographie einen sehr hohen Stellenwert. So werden Hashfunktionen beispielsweise eingesetzt, um Nachrichten digital zu signieren, und somit deren Integrität zu sichern.

MD5 MD5 ist eine sehr weit verbreitete Einweg-Hashfunktion, die bei vielen Security Anwendungen zum Einsatz kommt, zum Beispiel auch, um Nachrichten digital zu signieren. Sie wurde 1991 von Ronald Rivest entwickelt. MD5 produziert einen 128bit Hash Wert, indem zuerst die vorliegende Nachricht so mit Bits aufgefüllt wird, dass die sie kongruent 448 modulo 512 ist. Anschließend werden noch 64bit an die Nachricht angehängt die deren ursprüngliche Länge repräsentieren. Die Nachricht hat an diesem Punkt eine Länge, die ein Vielfaches von 512 darstellt. Dann wird die Nachricht über eine Funktion in vier *Runden* mit 32bit Wörtern vermischt, und hat schlussendlich einen 128bit Wert als Ausgabe.
Wie jede andere gute Hashfunktion auch, ist es Ziel von MD5, folgenden Parametern zu genügen:

- Bei gegebenem P, muss es sehr einfach sein $MD(P)$ zu berechnen
- Bei gegebenem $MD(P)$, muss es unmöglich bzw. nur mit unrealistisch großem Aufwand möglich sein, P zu berechnen
- Es darf kein P' zu einem gegebenem P auftreten, für das $MD(P') = MD(P)$ gilt
- Eine Änderung der Eingabe, selbst nur um 1 bit, muss eine vollkommen andere Ausgabe erzeugen.

Sind diese Bedingungen erfüllt, so kann die Funktion als sicher angesehen werden. Für MD5 wurden jedoch 2004 von einem chinesischen Wissenschaftlerteam Kollisionen nachgewiesen. Dies funktioniert jedoch nur, wenn P und $MD(P)$ bekannt sind, und das Ziel ist

ein P' zu finden, für das $MD(P') = MD(P)$ gilt. Ist nur der Hashwert $MD(P)$ bekannt, besteht keine Gefahr. Aus diesem Grund wird MD5 auch heute noch sehr häufig eingesetzt. (vgl. Tanenbaum 2003)

3.7.3 OpenSSL

OpenSSL ist eine Open Source Implementierung des SSL-Protokolls, dessen Aufgabe es ist, gesicherte Verbindungen zwischen zwei Hosts aufzubauen. SSL wird hauptsächlich für sicheres HTTP eingesetzt, um Daten auf einer Website verschlüsselt übertragen zu können. OpenSSL ist für die meisten UNIX basierten Systeme, sowie für Microsoft Windows erhältlich.
Der folgende Abschnitt soll zuerst einen Überblick über das SSL Protokoll geben, und anschließend das OpenSSL Paket genauer analysieren.

SSL

Das Secure Sockets Layer Protokoll ist ein transparentes Verschlüsselungsprotokoll, das im OSI Modell zwischen der Transportschicht und der Anwendungsschicht angesiedelt ist. SSL ist ein so genanntes hybrides Verschlüsselungsprotokoll (siehe 3.7.2), das bedeutet, dass sowohl asymmetrische als auch symmetrische Verschlüsselungsverfahren zum Einsatz kommen. Es bietet sowohl Verschlüsselung als auch *Public Key* basierte sowie zertifikatsbasierte Authentifizierung. SSL befindet sich inzwischen in der dritten Version und heißt seit 1999 eigentlich TLS (*Transport Layer Security*, RFC2246). TLS 1.0 und SSL 3.0 sind bis auf sehr geringe Unterschiede identisch. Wird im Folgenden von SSL gesprochen, so ist damit auch TLS gemeint.

Die Verbindung läuft bei SSL in zwei Schritten ab. Zuerst findet ein Handshake statt, bei dem die verwendeten Protokolle verhandelt werden, und der geheime Schlüssel für die spätere Datenübertragung mittels eines asymmetrischen Verfahrens ausgetauscht wird. Zusätzlich besteht noch die Möglichkeit der gegenseitigen Authentifizierung. Anschließend werden die Nutzdaten mit einem symmetrischen Verfahren verschlüsselt übertragen. SSL unterstützt eine Reihe verschiedener Protokolle für symmetrische, asymmetrische und Hash Verfahren:

- **Symmetrische Algorithmen** - RC2, RC4, IDEA, DES, Triple DES, AES
- **Asymmetrische Algorithmen** - RSA, Diffie-Hellman, DSA, Fortezza
- **Hashfunktionen** MD5, SHA

OpenSSL basiert auf SSLeay, das 1995 von Eric A. Young und Tim J. Hudson geschrieben wurde. Im Dezember 1998 wurde de Entwicklung von SSLeay eingestellt und die erste Version von OpenSSL veröffentlicht. Die OpenSSL Bibliothek unterstützt alle oben erwähnten kryptographischen Algorithmen und Hashfunktionen. Neben der Funktion als sichere Transportschicht für Applikationen, bietet das OpenSSL Paket auch ein CLI, über das sich die verschiedenen Hashfunktionen, sowie Verschlüsselungsverfahren und digitale Signaturen bedienen lassen. So ist es möglich, über das CLI jede beliebige Datei mit einem gewünschten Verschlüssellungsverfahren zu verschlüsseln, einen Hashwert zu erstellen, oder die Datei digital zu signieren. Zusätzlich ist es möglich, mit dem OpenSSL Paket eine so genannten *Certificate Authority Server* (CA) zu erstellen. (vgl. OpenSSL Website 2006)

OpenSSL und GPL

Zwar wird OpenSSL unter einer Open Source Lizenz vertrieben, jedoch beinhaltet diese Lizenz einen Absatz, der nicht mit Sektion 6 der GPL (*„You may not impose any further restrictions on the recipients' exercise of the rights granted herein“* (Free Software Foundation 1991)) vereinbar ist. Dies kann zu Problemen führen, wenn Software, die auf OpenSSL aufbaut, oder die OpenSSL Bibliothek verwendet, unter der GPL herausgegeben werden soll.
So wird auf `www.openssl.org` geraten, eine andere Lizenz als die GPL zu verwenden, oder explizit den Zusatz *„This program is released under the GPL with the additional exemption that compiling, linking, and/or using OpenSSL is allowed.“* (OpenSSL Website 2006) zu vermerken.

Ein Überblick über die Implementierung, sowie die einzelnen Funktionen des OpenSSL Pakets findet sich im praktischen Teil der Arbeit.

3.7.4 OpenPGP

OpenPGP (RFC 2440) ist ein offener Standard für Verschlüsselungssoftware, der auf PGP basiert. Der Standard beschreibt alle Fähigkeiten, die auch PGP besitzt. Es dient der Verschlüsselung, Signierung und Kompression von Daten, und wird hauptsächlich für den Versand elektronischer Nachrichten eingesetzt.

PGP PGP (*Pretty Good Privacy*) ist ein hybrides Datenverschlüsselungsverfahren, das 1991 von Phil Zimmermann entwickelt wurde. Anfänglich wurde RSA für die asymmetrische, und IDEA für die symmetrische Verschlüsselung eingesetzt. Wie bei anderen hybriden Verfahren, wird auch bei PGP die eigentliche Nachricht mit einem symmetrischen Verfahren verschlüsselt, und nur der symmetrische Schlüssel durch ein *Public Key* Verfahren geschützt, da das Verschlüsseln der gesamten Nachricht viel zu rechenaufwändig wäre. Dies ermöglicht dem Absender auf einfache Weise, eine verschlüsselte Nachricht an mehrere Empfänger zu verschicken, da immer nur ein kleiner Teil neu verschlüsselt werden muss, und nicht die gesamte Nachricht (siehe Abb. 3.10).

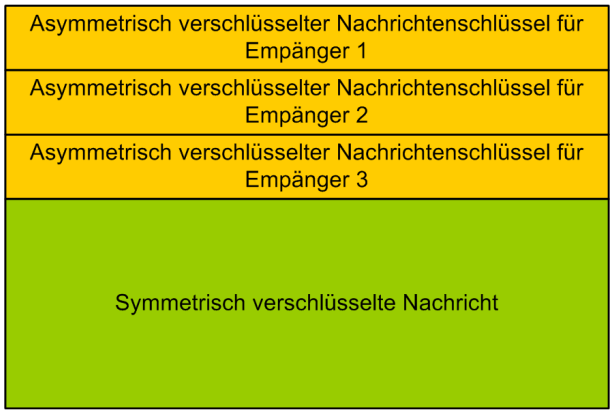

Abbildung 3.10: PGP Nachricht an mehrere Empfänger

Obwohl PGP zum Verschlüsseln jeder Art von Daten verwendet werden kann, ist das Haupteinsatzgebiet die Verschlüsselung von Emails. Soll eine elektronische Nachricht verschlüsselt

versendet werden, so müssen Sender und Empfänger zuerst jeweils ein asymmetrisches Schlüsselpaar besitzen. Anschließend muss der Absender einen symmetrischen Schlüssel erzeugen, mit dem die Nachricht verschlüsselt wird. Dann wird der symmetrische Schlüssel durch ein asymmetrisches Verfahren geschützt, indem er mit dem öffentlichen Schlüssel des Empfängers verschlüsselt wird, und der Nachricht beigefügt. Wird die Nachricht nun empfangen, so muss zuerst unter Zuhilfenahme des privaten Schlüssels des Empfängers der symmetrische Schlüssel wiederhergestellt werden, mit dessen Hilfe dann die Nachricht entschlüsselt werden kann.
Zusätzlich ist es mit PGP natürlich auch möglich, Nachrichten digital zu signieren, um die Grundsätze der Authentizität und Integrität zu wahren. Dazu wird ein *Hash* der unverschlüsselten Nachricht erzeugt, der wiederum mit dem privaten Schlüssel des Absenders verschlüsselt wird, und mit der Nachricht mit versandt wird. Der Empfänger kann nun den Hashwert mit Hilfe des öffentlichen Schlüssels des Absenders wiederherstellen, und ihn mit dem selber gewonnenen Wert vergleichen.

In den Anfängen von PGP gab es Schwierigkeiten mit dem Export aus den USA, da spezielle Gesetze verboten, Verschlüsselungstechnologien, die einen Schlüssel von über 40bit besitzen, zu exportieren. Als Reaktion darauf, verfasste Zimmermann ein Buch, das den gesamten PGP Quellcode enthielt. Dieser wurde dann im Ausland abgeschrieben und kompiliert.
1997 wurde PGP der IETF als offener Standard (RFC 2440) überlassen, die daraufhin die OpenPGP Working Group gründeten. Nachdem sich PGP jedoch nach der Entstehung des OpenPGP Standards weiterentwickelte, sind heutige PGP Versionen mit OpenPGP nicht mehr vollständig kompatibel.

GnuPG Der GnuPG (GNU Privacy Guard) oder GPG ist eine freie Implementierung des OpenPGP Standards, und ist vollständig konform mit RFC 2440. Das Programm ist Teil des GNU Projektes, wird unter der GPL herausgegeben, und ist meist Teil freier Betriebsysteme, wie Linux. Das Programm ermöglicht sowohl Verschlüsselung von Daten mit eine Reihe an symmetrischen und asymmetrischen Verschlüsselungsverfahren, als auch das digitale Signieren von Dokumenten, sowie Kompression. Dass Programm basiert auf einem Kommandozeilen Tool, allerdings gibt es eine Vielzahl an graphischen Frontends, sowie Plug-Ins für die meisten Email Clients. (vgl. GNU GPG Website 2006)

Security Tools Fazit

Open Source Produkte existieren für sämtliche Bereiche der IT Security. Es bestehen jedoch große Unterschiede in der Zahl an Produkten, die in den jeweiligen Kategorien zur Verfügung stehen. So gibt es beispielsweise eine große Zahl an Open Source Firewall Programmen, und Open Source Varianten von Intrusion Detection Systemen sind heutzutage fast die Regel. Die Streuungsdichte von Port Scanner hingegen ist bereits wesentlich geringer, und bei Anti-Viren Programmen konnte selbst nach längerer Recherche eigentlich nur ein nennenswerter Vertreter dieser Gruppe ausgemacht werden.
Unabhängig vom Einsatzbereich der verschiedene Produkte, ist jedoch die Qualität der meisten Programme sehr zufrieden stellend. Funktionsumfang, Stabilität und Dokumentation können sich problemlos mit kommerzieller Software messen bzw. übertreffen diese sogar teilweise.
Im folgenden Kapitel wird jeweils ein Vertreter jeder Kategorie praktisch eingesetzt, und auf seine Funktionsweise hin analysiert.

4 Praxistest

Ziel des praktischen Teils, ist die Implementierung eines Open Source Security Systems auf Basis von Linux. Dazu soll von jeder der in Kapitel 3 beschriebenen Gruppen von Security Tools ein repräsentatives Programm zur Installation herangezogen und bewertet werden. Dieses Kapitel dient der Dokumentation des praktischen Teils und ist nicht als Installationsanleitung oder Handbuch zu den gewählten Produkten zu verstehen. Jeder Punkt wird eine kurze Beschreibung der Installation enthalten, um etwaige Komplikationen zu dokumentieren, sowie eine ausführliche Erläuterung des Konfigurationsprozesses. Im Anschluss daran, wird am Ende des Kapitels eine Sicherheitsüberprüfung durchgeführt, um zu erörtern, ob sich mit den implementierten Tools eine sichere Umgebung herbeiführen lässt.

4.1 Teststellung

Die Umsetzung des praktischen Teils wird auf einem technisch nicht mehr aktuellen PC durchgeführt. Dies soll zusätzlich zeigen, ob auch sehr preisgünstige Hardware verwendet werden kann, um Open Source Security zu implementieren.

Testsystem

Das Testgerät ist ein handelsüblicher PC mit einem Intel Pentium III Prozessor (733MHz), 256MB Arbeitsspeicher, 8GB Festplatte, sowie zwei 100Mbps Netzwerkkarten. Die Netzwerkkarten werden jeweils an ein Netz angeschlossen, welche ein externes Netz (Internet) bzw. ein internes Netz (Firmen LAN) simulieren sollen (siehe Abb. 4.1).

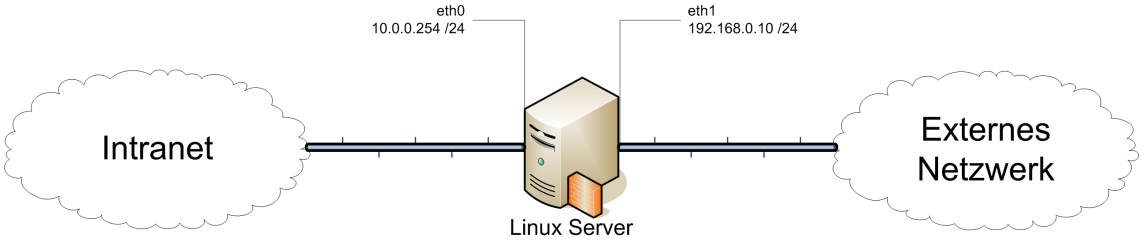

Abbildung 4.1: Konfiguration des Testsystems

4.2 Policy

Bevor in einem Unternehmen Sicherheitslösungen implementiert werden können, müssen zuerst Befugnisse und Verbote definiert werden, um festzulegen, was erlaubt ist, und was

verboten. Zu diesem Zweck wird eine Security Policy erstellt (siehe 2.2.3). Sie definiert den gesamten Ablauf, von der Organisation bis zur Durchführung, der notwendig ist, um einen definierten sicheren Zustand zu erreichen.

In Kapitel 2.2.3 wurden die nach ISO-17799 spezifizierten Richtlinien für Security Policies behandelt. Nachdem es sich hier jedoch lediglich um eine Teststellung eines Sicherheitssystems eines imaginären Unternehmens handelt, würde eine Policy von diesem Ausmaß den Rahmen dieser Arbeit sprengen.
Nichtsdestotrotz ist es notwendig, gewisse Richtlinien festzulegen, *bevor* mit der Konfiguration begonnen wird. Um diesem Anspruch gerecht zu werden, wird in jedem Abschnitt bei Bedarf der entsprechende Auszug einer Security Policy formuliert. Zusätzlich soll an dieser Stelle eine Liste mit Anforderungen festgelegt werden, die den Sicherheitsanspruch des gesamten Systems definiert:

- Der Zugang zu allen angeschlossenen Netzwerken muss durch die Firewall kontrolliert werden.
- Filialen muss es möglich sein, auf Dienste im internen Netz zuzugreifen.
- Öffentlich erreichbare Server dürfen sich nur in der DMZ befinden.
- Sämtlicher Web Traffic aus dem internen Netzwerk muss über einen Proxy Server laufen. Es darf nicht möglich sein, diesen zu umgehen.
- Alle Benutzer, denen Zugriff zur Firewall gewährt wird, müssen strengsten Passwortrichtlinien unterliegen.
- Der gesamte interne Netzwerkdatenverkehr muss auf Anzeichen von möglichen Sicherheitsverletzungen überwacht werden.
- Wichtige Dateien auf sensiblen Systemen müssen durch integritätsüberwachende Maßnahmen geschützt werden.
- Maßnahmen zum Schutz gegen Viren und andere bösartige Software müssen vorhanden sein.
- Zur Kommunikation mit außerhalb des geschützten Netzwerks liegenden Hosts, müssen Verschlüsselungsmechanismen zur Verfügung stehen.

Im Anschluss an die vollständige Implementierung des Sicherheitssystems, wird eine *Sicherheitsüberprüfung* durchgeführt, um zu erörtern, ob die in der obigen Liste geforderten Sicherheitsansprüche erfüllt wurden.

4.3 Betriebsystem

Bei der Wahl des Betriebsystem fiel die Entscheidung auf die Linux Distribution Fedora Core 4 (Kernel 2.6.11).
Das heutige Fedora Projekt wurde 2003 durch die Zusammenarbeit von Red Hat und dem damaligen Fedora Projekt gegründet. Das ehemalige Red Hat Linux wurde zugunsten von Fedora Core aufgegeben. Fedora Core versteht sich als ein vollständiges, vielseitiges Betriebsystem, das einfach zu installieren, zu konfigurieren und zu bedienen ist.

4.3.1 Installation

Die Installation von Fedora verlief reibungslos, steht an Einfachheit einer Installation von Windows XP in nichts nach, und wäre von jedem Anfänger problemlos durchführbar. Die Paketauswahl kann, dank Presets, vollkommen übersprungen werden, wenn dies gewünscht wird. Die Hardware wurde fehlerfrei erkannt, und die Konfiguration der Netzwerkkarten kann mittels eines *Wizards* durchgeführt werden.
Kurzum, die Zeiten, in denen eine Linux Installation nur von einem ausgebildeten Profi durchgeführt werden konnte, sind eindeutig vorbei.

4.3.2 Hardening

Nach erfolgreicher Installation des Betriebsystems, gilt es, dieses, wie in Kapitel 3.1.1 beschrieben, zu „härten“. Die einzelnen Schritte dieses *Hardendings* werden im Folgenden dokumentiert.

Entfernen unnötiger Software Pakete

Der erste Schritt beim Absichern eines Systems ist immer, zu definieren, welchen Zweck dieses erfüllen soll. Im Falle dieser Teststellung soll ein Sicherheitssystem realisiert werden. Sofern dies nicht bei der Installation schon vorgenommen wurde, gilt es zu bestimmen, welche der installierten Pakete, nicht benötigt werden und entfernt werden sollten.

Fedora Core basiert auf dem Paket Management System von RedHat, RPM. Der Befehl `rpm -qa` liefert eine Liste aller installierten Pakete.

```
[root@firewall ~]# rpm -qa
...
gcc-4.0.0-8
httpd-2.0.54-10
mod_ssl-2.0.54-10
...
```

Dieser Ausschnitt der Ausgabe von `rpm -qa` zeigt, dass `httpd-2.0.54-10` installiert ist. Dies ist der Apache Webserver. Für ein Security System ist es nicht ratsam, einen Webserver zu betreiben, sofern dies nicht unbedingt notwendig ist. Um zusätzliche Sicherheitsrisiken zu vermeiden, ist es also ratsam, den Webserver zu entfernen. Um Probleme mit Abhängigkeiten zu anderen Paketen zu vermeiden, empfiehlt sich die Verwendung von `yum`, dem Paketmanager von Fedora Core.

```
[root@firewall ~]# yum remove httpd
Setting up Remove Process
Resolving Dependencies
--> Populating transaction set with selected packages. Please wait.
---> Package httpd.i386 0:2.0.54-10 set to be erased
--> Running transaction check
...
Dependencies Resolved
...
Removed: httpd.i386 0:2.0.54-10
Complete!
```

Ein anschließender Aufruf von `rpm -qa httpd` bringt kein Ergebnis mehr. Neben dem Webserver gibt es noch eine Reihe anderer Pakete, die nicht benötigt werden. Es ist ratsam, die

Liste der installierten Pakete sorgfältig zu untersuchen, um wirklich alle unnötigen Pakete ausfindig zu machen.

Netzwerk Ports und Dienste

Der nächste Schritt sollte sein, sämtliche Dienste und Ports zu überprüfen, um sicherzugehen, dass keine unerwünschten Sicherheitslöcher entstehen. Zwar wird später die Firewall Ports nach außen hin schützen, fällt diese jedoch aus, lägen alle unachtsam geöffnet gelassenen Ports nach außen offen. Wie in 3.1.1 bereits erwähnt, sind Firewalls so genannte *Front Line Defense* Security Tools. Ziel ist es, das System auch ohne Firewalls möglichst gut abzusichern.

Der Befehl `netstat` liefert eine Liste aller verwendeten Ports, sowie die Information, welcher Prozess diesen Port verwendet.

```
[root@firewall ~]# netstat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address     Foreign Address     State       PID/Program name
tcp        0      0 0.0.0.0:32769     0.0.0.0:*           LISTEN      1396/rpc.statd
tcp        0      0 0.0.0.0:111       0.0.0.0:*           LISTEN      1378/portmap
tcp        0      0 127.0.0.1:631     0.0.0.0:*           LISTEN      1709/cupsd
tcp        0      0 127.0.0.1:5335    0.0.0.0:*           LISTEN      1691/mDNSResponder
tcp        0      0 127.0.0.1:25      0.0.0.0:*           LISTEN      1783/sendmail: acce
tcp        0      0 0.0.0.0:22        0.0.0.0:*           LISTEN      1757/sshd
udp        0      0 0.0.0.0:32768     0.0.0.0:*                       1396/rpc.statd
udp        0      0 0.0.0.0:68        0.0.0.0:*                       2908/dhclient
udp        0      0 0.0.0.0:724       0.0.0.0:*                       1396/rpc.statd
udp        0      0 0.0.0.0:5353      0.0.0.0:*                       1691/mDNSResponder
udp        0      0 0.0.0.0:5353      0.0.0.0:*                       1691/mDNSResponder
udp        0      0 0.0.0.0:111       0.0.0.0:*                       1378/portmap
udp        0      0 0.0.0.0:631       0.0.0.0:*                       1709/cupsd
```

Die Ausgabe von `netstat` zeigt, das zum Beispiel `cupsd`, der Druckerservice von Linux, auf UDP Port 631 auf allen angeschlossenen Interface aktiviert ist.
Der Befehl `chkconfig -list | grep on` liefert eine Liste aller Dienste, die nicht deaktiviert sind. Hier zeigt sich das `cups` in den Runlevels 2, 3, 4 und 5 aktiviert ist.

```
[root@firewall ~]# chkconfig --list | grep on
...
crond           0:off   1:off   2:on    3:on    4:on    5:on    6:off
cups            0:off   1:off   2:on    3:on    4:on    5:on    6:off
dhcpd           0:off   1:off   2:on    3:on    4:on    5:on    6:off
...
```

Dieser lässt sich mit `chkconfig cups off` in allen Runlevels deaktivieren und mit `service cups stop` sofort beenden. Sollte der Dienst gar nicht benötigt werden, kann das Paket, wie im vorhergehenden Punkt demonstriert, entfernt werden.

Ein anschließender Port Scan von außen mittels `nmap`, zeigt, dass der besagte UDP Port nicht mehr geöffnet ist.

```
Starting nmap V. 3.00 ( www.insecure.org/nmap/ )

Interesting ports on  (192.168.0.10):
(The 3063 ports scanned but not shown below are in state: closed)
Port       State       Service
```

```
22/tcp      open         ssh
67/udp      open         dhcpserver
68/udp      open         dhcpclient
111/tcp     open         sunrpc
111/udp     open         sunrpc
925/udp     open         unknown

Nmap run completed -- 1 IP address (1 host up) scanned in 1476 seconds
```

Inittab

Die Inittab Datei `/etc/inittab` beschreibt den Prozess des Hochfahrens des Systems. Hier sollte überprüft werden, ob alle Einträge dem System entsprechen. So ist es bei einem Sicherheitssystem meist nicht notwendig, das *X Windows System*, die graphische Oberfläche von Linux, zu verwenden. In diesem Fall ist es ratsam, das *Default Runlevel* auf 3 (Multi User Mode mit Netzwerk, ohne graphische Oberfläche) zu setzen.

```
id:3:initdefault:
```

Weiters ist es ratsam, die *CTRL-ALT-DEL-Trap*, die dem Erkennen und Abfangen der Tastenkombination Strg, Alt und Entf dient, zu deaktivieren, um versehentliche Reboots zu vermeiden. Dazu muss die folgende Zeile auskommentiert werden:

```
#ca::ctrlaltdel:/sbin/shutdown -t3 -r now
```

Ein weiterer Teil des Startup Prozesses ist die Datei `/etc/rc.local`. Mit Hilfe dieser Datei können benutzerdefinierte Scripts geladen werden. Sie sollte im Moment nur eine einzige Zeile enthalten:

```
touch /var/lock/subsys/local
```

Benutzerkonten

Existieren auf einem System Konten, die von keinem Dienst und keinem Benutzer verwendet werden, so sollten diese entfernt werden, um Missbrauch zu vermeiden. Um herauszufinden, welche Konten seit längerer Zeit nicht zum Login verwendet wurden, eignet sich zum Beispiel der Befehl `last`.
Bevor nicht genutzte Benutzerkonten gelöscht werden, sollte überprüft werden, ob der Benutzer Besitzer irgendwelcher Dateien ist, um nicht ein neues Sicherheitsrisiko durch besitzerlose Dateien entstehen zu lassen.

```
find / -path /proc -prune -o -user <account> -ls
```

Anschließend kann das Konto mit `userdel -r <account>` gelöscht werden.

Weiters sollte überprüft werden, ob alle nicht zum Login verwendeten Konten gesperrt sind. Wird ein Konto gesperrt, so steht anstelle des verschlüsselten Passwortes ein Rufzeichen oder ein Stern in der Datei `/etc/shadow`.

```
egrep -v '.*:\*|:\!' /etc/shadow | awk -F: '{print $1}'
```

Zusätzlich ist es noch wichtig zu überprüfen, ob alle Konten ein verschlüsseltes Passwort besitzen. Ist dies der Fall, so steht anstelle des Passwortes in der Datei `/etc/passwd` ein `x`. Folgender Befehl sollte im Idealfall keine Ergebnisse liefern.

```
grep -v ':x:' /etc/passwd
```

Fehlt das `x` bei einem Konto, so wird beim Login vom System keine Passworteingabe gefordert.

Passwörter

Wie in Kapitel 3.1.1 bereits beschrieben, gibt es einige Wege um die Verwendung von Passwörtern noch effektiver zu gestalten. Grundsätzlich ist es ratsam, dass Passwörter eine gewisse Komplexität besitzen. Um zu forcieren, dass ein Passwort aus mindestens 8 Zeichen besteht, Groß- und Kleinbuchstaben, sowie Zahlen und Sonderzeichen beinhaltet, müssen in der Datei `/etc/pam.d/system-auth` einige Parameter gesetzt werden.

```
...
password    requisite     /lib/security/$ISA/pam_cracklib.so retry=3 minlen=8 \
                          lcredit=-1 ucredit=-1 dcredit=-1 ocredit=-1
...
```

Standardmäßig ist in der oben angegeben Zeile nur der Parameter `retry=3` definiert, der die Anzahl der möglichen Eingabeversuche festlegt. Der Parameter `minlen=8` gibt an, dass ein Passwort aus mindestens 8 Zeichen bestehen muss. `lcredit=-1` bewirkt, dass ein Passwort nur als gültig erkannt wird, wenn es zumindest einen Kleinbuchstaben enthält. `ucredit=-1`, `dcredit=-1` und `ocredit=-1` haben denselben Effekt bezüglich Großbuchstaben, Zahlen und Sonderzeichen. All diese Maßnahmen bewirken, dass es unmöglich wird, ein Passwort mittels eines Dictionary Angriffs zu knacken, und es ungleich schwerer wird, mit Hilfe von Brute Force ans Ziel zu kommen.

Um zu verhindern, dass ein Benutzer ein und dasselbe Passwort für unbegrenzte Zeit verwendet, ist es sinnvoll, *Password Aging* zu verwenden. Hierbei werden in der Datei `/etc/login.defs` einige Parameter spezifiziert.

```
...
PASS_MAX_DAYS   60
PASS_MIN_DAYS   7
PASS_WARN_AGE   7
...
```

Zusätzlich können in der Datei `/etc/default/useradd` die Parameter `INACTIVE` und `EXPIRE` angegeben werden, die definieren, nach welcher Zeit ein Konto mit einem abgelaufenen Passwort deaktiviert werden soll, bzw. ein absolutes Verfallsdatum für ein Konto angeben.

Alle Änderungen sollten jedoch vorgenommen werden, *bevor* Benutzer angelegt werden, da sie erst durch den Befehl `useradd` zur Anwendung kommen. Um Passwort Aging Parameter für bereits bestehende Benutzerkonten zu verändern, muss der Befehl `chage` verwendet werden. Die festgelegten Parameter finden sich wieder in der Datei `/etc/shadow`, nach dem Muster

```
<username>:<password>:<date>:PASS_MIN_DAYS:PASS_MAX_DAYS:PASS_WARN_AGE:INACTIVE:EXPIRE:
```

Als letzte Maßnahme sollte noch sichergestellt werden, das bereits verwendete Passwörter nicht wieder verwendet werden können, und das sich das neue Passwort vom alten in mindestens 3 Zeichen unterscheiden muss. Hierzu ist der Parameter `difok=3` für das Modul `pam_cracklib` sowie remember=13 für das Modul `pam_unix` in der Datei `/etc/pam.d/system-auth` zu setzen. Der Wert `13` für den Parameter `remember` wurde gewählt, da zuvor `PASS_MIN_DAYS` auf `7` gesetzt wurde. Werden also die letzten dreizehn Passworte gemerkt, und ist die Mindestdauer eines Passwortes sieben Tage, so ergibt das eine Zeit von 3 Monaten, in denen das gleiche Passwort nicht zweimal verwendet werden darf.

SSH

SSH ist, wie in Kapitel 3.1.1 bereits beschrieben, ein sicherer Weg, um auf ein System von einem anderen Standort aus zuzugreifen. Es gibt allerdings einige Parameter in der Datei `/etc/ssh/sshd_config`, die konfiguriert werden müssen, um zu verhindern, dass Sicherheitsrisiken entstehen
Zum einen soll in diesem Szenario nur die IT Abteilung Zugang zum System haben. Aus diesem Grund ist es also nicht notwendig, dass der SSH Daemon auf allen Interfaces erreichbar ist, wie es aus der unten stehenden Ausgabe von `netstat` ersichtlich ist.

```
[root@firewall ~]# netstat
...
tcp        0      0 0.0.0.0:22      0.0.0.0:*           LISTEN      1757/sshd
...
```

Um dies zu ändern, muss folgende Zeile muss in der Konfigurationsdatei des SSH Daemons eingetragen werden:

```
ListenAddress 10.0.0.254
```

Ein erneutes Ausführen von `netstat` zeigt, dass SSH nun nur mehr über das interne Netzwerk erreichbar ist.

```
[root@firewall ~]# netstat
...
tcp        0      0 10.0.0.254:22      0.0.0.0:*           LISTEN      1757/sshd
...
```

Ein weiterer wichtiger Punkt ist, den SSH Login für den Superuser `root` zu verbieten, indem folgende Zeile in der Konfigurationsdatei eingefügt wird.

```
PermitRootLogin no
```

Um sicherzustellen, dass das System sehr gut abgesichert ist, sollten auch folgende Parameter konfiguriert werden.

```
...
UsePrivilegeSeparation yes
Protocol 2
AllowTcpForwarding no
X11Forwarding no
StrictModes yes
IgnoreRhosts yes
HostbasedAuthentication no
RhostsRSAAuthentication no
...
```

Diese bewirken, dass der Daemon in einem *chroot jail* ausgeführt wird, dass nur SSH Version 2 verwendet werden darf, dass Forwarding verboten wird, und dass *Host-Based Authentication* deaktiviert ist.

MTA

Unabhängig davon, welcher MTA (Mail Transport Agent) verwendet wird, ist es wichtig darauf zu achten, über welche Interfaces dieser erreichbar ist. Ist ein System, so wie diese Teststellung, nicht als Mailserver konzipiert, so muss der MTA nur über `127.0.0.1` erreichbar sein. Die Ausgabe von `netstat` sollte also diese Zeile enthalten.

```
tcp        0      0 127.0.0.1:25      0.0.0.0:*           LISTEN      1783/sendmail: acce
```

Ist dies nicht der Fall, so muss der entsprechende Eintrag in der Konfigurationsdatei, in diesem Fall `/etc/mail/sendmail.cf`, geändert werden.

Kernel

Einige Kernelparameter lassen sich so konfigurieren, dass, wie in Kapitel 3.1.1 beschrieben, gewisse Angriffsszenarien verhindert werden können. In der Datei `/etc/sysctl.conf` lassen sich einige securitybezogene Kernelparameter einstellen. Dazu gehören SYN-Cookies, ICMP Redirect, Source Address Verification und Logging. Um diese in genannter Reihenfolge entsprechend zu setzen, müssen folgende Einträge vorgenommen werden

```
net.ipv4.tcp_syncookies = 1
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.all.log_martians = 1
```

Details zu den einzelnen Parametern sind Kapitel 3.1.1 zu entnehmen.

Banner

Wie bereits erwähnt, haben Login Banner vor allem den Zweck, potentielle Eindringlinge abzuschrecken. Um Login Banner zu realisieren, gibt es verschiedene Möglichkeiten. Soll der Banner *nach* dem Login erscheinen, so ist die Botschaft in die Datei `/etc/motd` einzutragen. Soll der Banner jedoch bereits *vor* dem Login Prompt sichtbar sein, so muss die Datei

/etc/issue geändert werden. Um einen SSH Login Banner zu implementieren, muss in der SSH Konfigurationsdatei der Parameter Banner bearbeitet werden.
Unabhängig davon, wo der Banner eingesetzt wird, könnte der Inhalt etwa so aussehen:

```
                    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                    !!!!!!!!  W A R N I N G  !!!!!!!!!
                    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

        This system is a restricted access area. All activity on this system
        is subject to careful MONITORING. Unauthorized subjects are to LEAVE
        AT ONCE! If any information collected reveals possible criminal
        activity or activity that exceeds privileges, all evidence of such
        activity will be provided to the responsible authorities for further
        action. By continuing past this point you expressly consent to this
        monitoring.
```

Bastille Linux

Ein weiterer Weg bzw. eine Ergänzung zu den bisher durchgeführten Maßnahmen ist Bastille Linux. Im folgenden Abschnitt soll kurz gezeigt werden, wie das automatisierte Tool funktioniert.

Nachdem Bastille herunter geladen und installiert wurde, kann es mittels bastille -c im Textmodus gestartet werden. Der Ablauf von Bastille ist sehr einfach gestaltet und kann auch von weniger versierten Benutzern problemlos durchgeführt werden. In einer Reihe von Fragen wird erhoben, welche Sicherheitsmaßnahmen am System durchgeführt werden sollen. Jede Frage beinhaltet eine ausführliche Erklärung sowie voreingestellte Antworten, die auf den Großteil der Systeme zutreffen dürften. Die angebotenen Sicherheitsmaßnahmen decken sich im Großen und Ganzen mit den bisher bereits manuell implementierten Maßnahmen. Allerdings gibt es noch ein paar Erweiterungen, die sich sogar hin bis zu FTP Sicherheit und Firewalling ausdehnen. Das gesamte Paket beinhaltet die folgenden Änderungen:

- **Zugriffsrechte für Administrationstools verschärfen** - Der Zugriff auf eine Reihe von Administrationstools, wie zum Beispiel ifconfig, wird auf root beschränkt
- **SUID-Bit deaktivieren** - Für mount, umount, ping, at, usernetctl und traceroute kann das SUID-Bit, dass es Benutzern ermöglicht ein Programm als root auszuführen, deaktiviert werden
- rtools **deaktivieren** - Die nicht mehr zeitgemäßen Remote Access Tools rlogin, rsh, rcp und rdist werden deaktiviert
- **Password Aging aktivieren** - Die oben dokumentierten Maßnahmen zum Password Aging werden automatisch implementiert
- **Default umask konfigurieren** - Die umask bestimmt die Standardberechtigungen für neu erstellte Dateien
- root **Login auf ttys verbieten** - Direkten Root Login verhindern, um su zu erzwingen
- **Passwort geschützter Bootloader** - Passwortabfrage zur Eingabe von Boot Argumenten

- **Passwort geschützter Single-User Mode** - Um zu vermeiden, dass sich ein Angreifer über den Bootloader `root` Zugang im Single-User Mode verschafft, kann dieser durch ein Passwort geschützt werden
- **Warnmeldung erstellen** - Die Datei `/etc/issue` wird mit einem standardisierten Banner befüllt
- **Systemressourcen limitieren** - Um DoS Attacken zu vermeiden, können die Ressourcen, die einem Prozess zur Verfügung stehen, limitiert werden
- **Konsolenzugriff beschränken** - Ermöglicht die Einschränkung des Konsolenzugriffs auf eine ausgewählte Benutzergruppe
- **Process Accounting** - Es wird festgehalten, welcher User welche Befehle ausführt. Dies kann hilfreich sein, um herauszufinden, was genau ein Angreifer gemacht hat
- **Nicht genutzte Daemons deaktivieren**
  - APM, NFS/Samba, PCMCIA, DHCP, mDNSResponder, Bluetooth, HP Officejet, ISDN, kudzu, sendmail deamon mode
- **FTP** - Bastille beinhaltet auch einige Absicherungsmöglichkeiten für *Anonymous FTP Access*
- **Firewall** - Zum Schluss wird noch die Möglichkeit geboten, ein einfaches iptables Skript zu erstellen, um zumindest einen grundlegenden Schutz zu bieten

Bastille ist ein sehr hilfreiches Tool, dass auch weniger geschulten Benutzern in kürzester Zeit ermöglicht, wichtige Schlüsselpunkte im System abzusichern. Wie jedes automatisierte Tool bringt es jedoch den Nachteil mit sich, dass nicht genau kontrollierbar ist, was verändert werden sollte, und somit eine manuelle Absicherung immer genauer sein wird.

4.4 Firewall

Nachdem das Betriebsystem entsprechend abgesichert wurde, ist der erste Schritt zu einer robusten Security Lösung die Konfiguration einer Firewall. Als so genanntes *Frontline Security Tool* ist sie dafür verantwortlich, zu kontrollieren, auf welchen Wegen der Zugriff auf das System erfolgen darf. Eine Firewall ist sozusagen die grobe Kontrollinstanz, die die meisten unerwünschten Zugriffe verhindern wird.

4.4.1 Netzplan

Um die Konfiguration der Firewall umfangreicher und vollständiger gestalten zu können, werden zusätzlich zum weiter oben beschriebenen Laboraufbau eine DMZ (*DeMilitarized Zone*), sowie drei Server, die verschiedene Dienste anbieten, angenommen. Weiters befinden sich drei Workstations im internen Netz sowie eine Filiale auf der Außenseite (siehe Abb. 4.2).

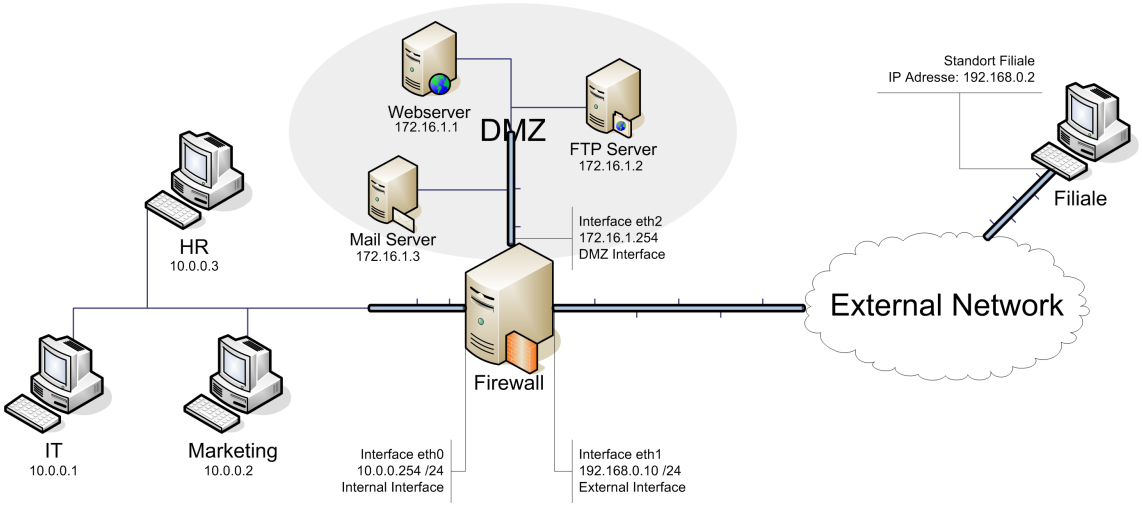

Abbildung 4.2: Firewall Netzplan

4.4.2 Policy

Um eine Firewall zu konfigurieren, muss zuerst festgelegt werden, was erlaubt, und was verboten sein soll. Die in 4.2 definierten Parameter zum Erreichen eines sicheren Zustandes, sollen an dieser Stelle genauer spezifiziert werden, um festzulegen, welche Kriterien die Firewall erfüllen muss:

- Jedem Host aus dem Intranet (internes Netz) wird uneingeschränkter Zugang zum Internet (externes Netz) gewährt.
- Sämtlicher HTTP Traffic (TCP Port 80) aus dem Intranet wird über einen Transparenten Proxy, der auf der Firewall implementiert ist, umgeleitet.
- Keinem in der DMZ befindlicher Host darf der Zugriff auf das Intranet erlaubt sein.
- Sämtlicher HTTP (TCP Port 80), FTP (TCP Port 21), SMTP (TCP Port 25) und POP (TCP Port 110) Traffic aus dem Internet wird in die DMZ zu den entsprechenden Servern umgeleitet.
- Der Zugang zur Firewall ist der IT Abteilung (IP Adresse 10.0.0.1) sowie der Filiale (IP Adresse 192.168.0.2)vorbehalten.
- Traffic der Filiale (IP Adresse 192.168.0.2) für den Zugriff auf die Personaldatenbank (TCP Port 5950) aus dem Internet wird zur HR Abteilung (IP Adresse 10.0.0.3) weitergeleitet.
- Alle oben nicht erwähnten Verbindungen werden blockiert.

4.4.3 Shell Scripts

Um die Firewallkonfiguration nicht bei jedem Start des Systems neu eingeben zu müssen, ist es hilfreich, diese in ein *Shell Script* zu verpacken. Unter Linux werden sehr häufig Shell Scripts zur Abarbeitung automatisierter Vorgänge verwendet. Ein Shell Script besteht aus einer Reihe von Befehlen, die normalerweise per Hand in die Shell eingegeben würden. Für Aufgaben, die sich oft wiederholen und aus mehreren Befehlen bestehen, ist es sinnvoll, ein

Script zu schreiben, da dann immer nur ein Befehl eingegeben werden muss. Ein gute Beispiel für Shell Scripts, sind automatisierte Aufgaben, die von `cron` ausgeführt werden. Soll beispielsweise jeden Tag um zwanzig Uhr ein Backup von wichtigen Files gemacht werden, die in einen Ordner mit dem entsprechenden Datum kopiert werden sollen, so könnte ein entsprechendes Shell Script etwa so aussehen:

```
mkdir /backup/‘date +%d%m%Y‘
cp /important_files/* /backup/‘date +%d%m%Y‘
```

Liegt das Script unter dem Namen `backup.sh` im Verzeichnis /scripts, sähe die entsprechende Zeile in der `crontab` so aus:

```
...
00 20 * * * root /scripts/backup.sh
...
```

Da bei iptables die Regeln als eine Reihe von Shell Befehlen eingegeben wird, kann es sehr leicht passieren, dass man den Überblick verliert. Außerdem ist es oft notwendig, die Konfiguration zu ändern bzw. nach einem Neustart oder einer falschen Konfiguration den ursprünglichen Zustand schnell wieder herzustellen.
Aus diesem Grund ist es ratsam, alle iptables Befehle in ein gemeinsames Shell Script zu schreiben. Dies verbessert den Überblick und erleichtert die Konfiguration.

4.4.4 Konfiguration

Wie in Kapitel 3.2.2 bereits beschrieben, baut iptables auf einer Reihe von Regeln auf, die in Chains zusammengefasst sind. Diese wiederum werden in Tables aufgeteilt, wobei jede Table einen Teilbereich der Funktionen von iptables darstellt (filter, nat und mangle).
Konfiguriert wird iptables mit eine Reihe von Shell Befehlen in der Form `iptables -[AD] chain rule-specification [options]`. Um die Konfiguration zu erleichtern, werden die einzelnen Befehle, wie bereits erwähnt, meist in einem Shell Script zusammengefasst. Im folgenden Abschnitt soll die Konfigurationsdatei anhand von wichtigen Schlüsselstellen analysiert werden. Das Script in seiner Gesamtheit findet sich in Anhang B.

Variable und Systemkonfiguration

Um den Umgang mit der Konfigurationsdatei zu vereinfachen, ist es ratsam, zu Beginn wichtige Variable zu definieren. Dazu gehören zum Beispiel die wichtigsten Daten der angeschlossenen Netzwerke, sowie Netzwerkadapter, oder die IP Adressen wichtiger Hosts. Der folgende Ausschnitt zeigt die Variablendefinition für das am internen Interface angeschlossene Netzwerk. Die Konfiguration für Internet und DMZ verläuft analog dazu.

```
DEV_INT=eth0
IP_INT=‘ifconfig $DEV_INT |grep inet |cut -d : -f 2|cut -d \  -f 1‘
MASK_INT=‘ifconfig $DEV_INT |grep Mask |cut -d : -f 4‘
NET_INT=‘ifconfig $DEV_INT |grep inet |cut -d : -f 2 |cut -d \-f 1 | \
         cut -d . -f 1,2,3‘.0/$MASK_INT
BC_INT=‘ifconfig $DEV_INT|grepBcast |cut -d : -f 3| cut -d \  -f 1‘
```

Die verschiedenen Adressen der Adapter werden hier über eine Reihe von Unix Befehlen aus dem Output von `ifconfig` gewonnen und in Variablen gespeichert. In Zukunft kann nun

zum Beispiel DEV_INT anstelle von eth0 verwendet werden, wenn das Interface angesprochen werden soll. Dies ist vor allem hilfreich, sollte sich das interne Interface ändern, sodass zum Beispiel eth2 anstelle von eth0 mit dem internen Netzwerk verbindet. In diesem Fall müsste der Wert nur an einer einzigen Stelle geändert werden, und nicht in jeder Regel.

Variablen sind auch hilfreich, um Adressen häufig verwendeter Hosts zu speichern. Wie schon beim Interface, ist der Konfigurationsaufwand bei etwaigen Änderungen wesentlich geringer. Der folgende Auszug zeigt eine Liste wichtiger Hosts der Teststellung.

```
# Interne Hosts

IT="10.0.0.1"
MARKETING="10.0.0.2"
HR="10.0.0.3"

# Externe Hosts

FILIALE="192.168.0.2"

# Server

WEBSERVER="172.16.1.1"
FTPSERVER="172.16.1.2"
MAILSERVER="172.16.1.3"
```

Die Verwendung von Variablen erleichtert nicht nur den Konfigurationsaufwand, sondern trägt auch unterstützend zur Übersichtlichkeit und Verständlichkeit der Konfigurationsdatei bei. Soll beispielsweise der Filiale Zugriff auf interne Ressourcen gewährt werden, so ist die Lesbarkeit der Regel viel besser, wenn FILIALE anstelle einer IP Adresse steht.

Wie bereits in 3.2.2 beschrieben, besteht iptables aus einer Reihe von Modulen. Diese müssen zu Beginn mittels modprobe geladen werden. modprobe ist ein Tool, mit dessen Hilfe Module dynamisch bei Bedarf in den Kernel geladen werden können. Der unten stehende Auszug zeigt nur einen Teil der zu ladenden Module.

```
# Module laden
...
modprobe ip_tables
modprobe ip_conntrack
modprobe iptable_filter
modprobe iptable_mangle
modprobe iptable_nat
...
```

Die geladenen Module erfüllen eine Vielzahl an unterschiedlichen Aufgaben. So ist beispielsweise das ip_conntrack Modul zuständig für die Connection Tracking Fähigkeit von iptables.
Anschließend müssen noch einige Kernelparameter gesetzt werden, um sicherzustellen, dass iptables ordnungsgemäß funktionieren kann. Dies wird anhand des /proc Filesystems durchgeführt, das eine Reflexion aller laufenden Prozesse darstellt, und in dem gewisse Kernelparameter verändert werden können.

```
echo "1" > /proc/sys/net/ipv4/ip_forward
echo "1" > /proc/sys/net/ipv4/conf/all/rp_filter
echo "1" > /proc/sys/net/ipv4/conf/all/proxy_arp
echo "1" > /proc/sys/net/ipv4/ip_dynaddr
```

Chains und Rules

Nach der Konfiguration der Umgebungsvariablen beginnt die eigentliche Konfiguration der Firewall. Zu Beginn jeder Konfiguration müssen zuerst alle Regeln aus allen Chains gelöscht werden, da die Regeln in iptables sequentiell abgearbeitet werden. Es müssen also alle vorhanden Regeln entfernt werden, da diese sonst bestehen bleiben würden, und neue Regeln einfach angehängt würden, und somit nutzlos wären.
Anschließend werden die *Default Policies* (siehe 3.2.2) gesetzt, und zwar so, dass jedes Paket, auf das keine Regel zutrifft, verworfen wird (siehe 3.2.1).
Um die Flexibilität und die Skalierbarkeit der Konfiguration zu erhöhen, ermöglicht iptables die Erstellung eigener Chains. Im letzten Abschnitt des unten stehenden Konfigurationsauszugs werden eigene Chains für TCP, UDP und ICMP Pakete erzeugt, sowie Chains für ungültige und gültige Pakete.

```
# Chains leeren

$IPTABLES -F
$IPTABLES -t nat -F
$IPTABLES -t mangle -F
$IPTABLES -X

# Default policies setzen

$IPTABLES -P INPUT DROP
$IPTABLES -P FORWARD DROP
$IPTABLES -P OUTPUT DROP

# Eigene Chains erstellen

$IPTABLES -N allowed
$IPTABLES -N tcp_packets
$IPTABLES -N udp_packets
$IPTABLES -N icmp_packets
$IPTABLES -N bad_tcp_packets
```

Aufgabe der `allowed` Chain ist es, festzulegen, was mit TCP Paketen, die von anderen Regeln als gültig befunden wurden, passieren soll. Tritt ein TCP Paket auf, auf das eine vorhandene Regel zutrifft, so wird dieses zur Durchführung weiterer Schritte an die `allowed` Chain übergeben. Sie stellt ein Bindeglied zwischen der aufrufenden Regel und dem `ACCEPT` Target dar, das mehr Flexibilität bei der Abarbeitung der Pakete erlaubt.
Im konkreten Fall werden die Pakete in der `allowed` Chain auf ihren Status überprüft. Hat ein Paket das `SYN`-Flag gesetzt (ist es also neu), oder ist es Teil einer bestehenden Verbindung, wird es zugelassen. Alle anderen Pakete werden mit `tcp-reset` verworfen.

```
# allowed Rules
$IPTABLES -A allowed -p tcp --syn -j ACCEPT
$IPTABLES -A allowed -p tcp -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A allowed -p tcp -j REJECT --reject-with tcp-reset
```

Der nächste Abschnitt beschäftigt sich mit der Konfiguration der Regeln für TCP, UDP und ICMP Pakete, die für die Firewall selbst bestimmt sind, also von der `INPUT` Chain aufgerufen werden. Da es sich um ein Sicherheitssystem handelt, werden nur wenige Ports offen gehalten. So ist der Zugriff nur auf den TCP Ports für SSH und den Web Proxy Server zugelassen. Die Regeln für ICMP lassen nur Ping und Time Exeeded (vgl. Postel 1981) zu, und blockieren alle anderen ICMP Signale.

```
# TCP Rules

# sshd (Filiale)
$IPTABLES -A tcp_packets -i $DEV_EXT -p tcp -s $FILIALE -d $IP_EXT \
--dport 22 -j allowed

# sshd (IT)
$IPTABLES -A tcp_packets -i $DEV_INT -p tcp -s $IT -d $IP_INT \
--dport 22 -j allowed

# Squid (Filiale)
$IPTABLES -A tcp_packets -i $DEV_EXT -p tcp -s $FILIALE -d $IP_EXT \
--dport 3128 -j allowed


# ICMP rules

# ICMP Echo Request
$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 8 -j ACCEPT
$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 11 -j ACCEPT
```

Um die oben erstellten Regeln zum Einsatz zu bringen, muss in der `INPUT` Chain der entsprechende *Jump* gesetzt werden. Der folgende Abschnitt legt fest, dass alle Pakete, die Teil einer bestehenden Verbindung sind, akzeptiert werden, während alle anderen zu den entsprechenden Chains weitergeleitet werden.

```
$IPTABLES -A INPUT -p ALL -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A INPUT -p TCP -j tcp_packets
$IPTABLES -A INPUT -p UDP -j udp_packets
$IPTABLES -A INPUT -p ICMP -j icmp_packets
```

Neben dem Filtern von Paketen ist die Network Address Translation (NAT) eine wichtige Aufgabe der Firewall. NAT wird verwendet, um Hosts, die sich hinter der Firewall befinden, den Zugang zum Internet zu erlauben. Es wird bei iptables ganz einfach mittels der folgenden Zeile aktiviert.

```
# NAT fuer LAN Internet Access
$IPTABLES -t nat -A POSTROUTING -o $DEV_EXT -j SNAT --to-source $IP_EXT
```

Ein weiterer wichtiger Teil der Arbeit einer Firewall ist das so genannte *Port Forwarding*. Diese, auch *Virtual Server* genannte, Vorgehensweise beschreibt das Weiterleiten einer Verbindung an einen hinter der Firewall gelegenen Rechner. Soll zum Beispiel ein Webserver, der in der DMZ steht, seine Dienste nach außen hin anbieten, so können Verbindungen, die auf der Firewall auf Port 80 ankommen, an den Server in der DMZ weitergeleitet werden. Nach außen hin hat es den Anschein, als wäre der Server direkt an das Internet angeschlossen.

Um dies zu realisieren, bedarf es dreier Einträge in der Konfigurationsdatei. Zuerst muss in der `PREROUTING` Chain, also noch bevor eine Routingentscheidung getroffen wurde, die Zieladresse zu jener des Webservers geändert werden. Dies geschieht mittels *Destination NAT*. Nachdem die Routingentscheidung getroffen wurde, und das Paket durch die Firewall hindurch geleitet wird, passiert es die `FORWARD` Chain. An dieser Stelle muss festgelegt werden, dass ein Paket, welches am externen Interface angekommen ist und die Firewall über das DMZ Interface verlassen will, zugelassen wird, sofern es als Zieladresse den Webserver auf TCP Port 80 hat. Zum Schluss, bevor das Paket die Firewall wieder verlässt, kann in der `POSTROUTING` Chain noch die Quelladresse des Paketes auf jene des DMZ Interface der

Firewall geändert werden. Dies erleichtert die Konfiguration etwaiger weiterer Zugriffskontrollmechanismen auf dem Server, birgt jedoch ein gewisses Risiko in sich, da es schwieriger wird, Angreifer zu identifizieren.

Anschließend wird noch festgelegt, dass alle Pakete, die vom internen Interface ankommen, sowie alle bestehenden Verbindungen von allen Interfaces, weitergeleitet werden.

```
# Virtual Server

# HTTP forward zu Webserver

$IPTABLES -t nat -A PREROUTING -i $DEV_EXT -p tcp -d $IP_EXT --dport 80 -j DNAT \
--to-destination $WEBSERVER
...
$IPTABLES -A FORWARD -i $DEV_EXT -o $DEV_DMZ -d $WEBSERVER -p tcp --dport 80 -j ACCEPT
...
$IPTABLES -t nat -A POSTROUTING -p tcp -d $WEBSERVER --dport 80 -j SNAT \
--to-source $IP_DMZ

# Alle Pakete vom LAN forwarden
$IPTABLES -A FORWARD -i $DEV_INT -p ALL -s $NET_INT -j ACCEPT

# Alle Established/Related Verbindungen forwarden
$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Neben Filterregeln bietet iptables auch die Möglichkeit, Pakete zu verändern. So wird im Folgenden beispielsweise die TTL jedes Pakets, welches vom Internen Netz ankommt, um 1 erhöht. Dies bewirkt, dass ein externer Beobachter nicht unterscheiden kann, ob ein Paket von der Firewall selbst, oder vom Netz, welches sich dahinter befindet, ausging. Somit ist es von außen nicht möglich, überhaupt noch zu erkennen, dass sich hinter der Firewall ein Netz befindet.

```
# TTL erhöhen um LAN zu verstecken
$IPTABLES -t mangle -A PREROUTING -i $DEV_INT -j TTL --ttl-inc 1
```

Ein weiterer, für Unternehmen nicht unwichtiger Punkt, der sich mit Hilfe der Firewall realisieren lässt, sind transparente Proxyserver. Ein transparenter Proxy ist einer, der vom Anwender nicht wahrgenommen werden kann. Dies ist hilfreich um sicherzustellen, dass ein Anwender den Proxyserver nicht umgehen kann.

Üblicherweise werden Proxyserver vom Anwender im Browser konfiguriert. Wird ein transparenter Proxy verwendet, so muss vom Anwender nichts konfiguriert werden. Aus seiner Perspektive besteht eine direkte Verbindung ins Internet. Wenn jedoch die Verbindungsanforderung auf TCP Port 80 an der Firewall ankommt, so wird diese nicht wie sonst an den entsprechenden Webserver im Internet weitergeleitet, sondern auf einen Proxyserver umgeleitet. Bei dem unten stehenden Konfigurationsauszug läuft der Proxy Server direkt auf der Firewall. Die Verbindung wird einfach mit Hilfe des `REDIRECT` Statements umgeleitet.

```
# Transparenter Proxy
$IPTABLES -t nat -A PREROUTING -i $DEV_INT -p tcp -dport 80 -j REDIRECT --to-port 3128
```

Firewall Fazit

Mit den oben stehenden Konfigurationsauszügen wurde eine Firewall implementiert, die alle wichtigen grundlegenden Aufgaben erfüllt: Pakete filtern, Network Address Translation, und Port Forwarding. Das vollständige iptables Konfigurationsscript befindet sich im Anhang der Arbeit (siehe Anhang B).
Im Anschluss an die Fertigstellung aller sicherheitsrelevanten Komponenten wird am Ende des Kapitels eine Sicherheitsüberprüfung durchgeführt, die ermitteln wird, ob die gewählte Firewallkonfiguration den Anforderungen entspricht.

Der nächste Abschnitt beschäftigt sich mit der Konfiguration und dem Betrieb von Intrusion Detection Systemen.

4.5 NIDS

Nachdem das System durch die Firewall abgesichert wurde, ist es als nächster Schritt ratsam, Intrusion Detection Systeme in Betrieb zu nehmen, um nicht im Dunkeln zu stehen, sollte es doch jemandem gelingen, die Firewall zu überwinden.
Der folgende Abschnitt wird sich mit der Implementierung eines NIDS, als ersten Schritt zu einem sicheren System, beschäftigen. Im nächsten Abschnitt kommt dann zusätzlich ein HIDS zum Einsatz, das Bereiche abdecken soll, die ein NIDS nicht überwachen kann.

Für die netzwerkbasierte Überwachung fiel die Wahl auf Snort (2.6.0), da es sehr vielseitig ist, und das am weitesten verbreitetste NIDS darstellt.

4.5.1 Intrusion Detection mit Snort

Wie in 3.3.2 bereits beschrieben, bietet Snort drei verschiedene Betriebszustände. Den Packet Sniffer Mode, den Packet Logger Mode sowie den Intrusion Detection Mode.
Im Packet Sniffer Modus verhält sich Snort wie jedes andere Sniffing Programm (zum Beispiel tcpdump oder Ethereal). Um Snort in diesem Modus zu starten, wird es einfach mit folgender Option aufgerufen

```
[root@firewall ~]# snort -v
```

Dies bewirkt, dass Snort alle ankommenden und ausgehenden Datenpakete registriert, und auf der Standardausgabe darstellt. Wird der Modus beendet, so zeigt das Programm eine Zusammenfassung aller registrierten Pakete.
Um Datenpakete nicht nur anzuzeigen, sondern zur späteren Analyse auch aufzuzeichnen, muss Snort im Packet Logger Modus gestartet werden.

```
[root@firewall ~]# snort -v -l /var/log/snort
```

Der Vorteil dieses Modus ist, dass die aufgezeichneten Datenpakete zu einem späteren Zeitpunkt in Ruhe analysiert werden können. Da oft viele tausende Datenpakete in wenigen Sekunden über ein Interface kommen, ist dies im Packet Sniffer Mode eigentlich nicht möglich. Zusätzlich bietet Snort noch die Option, den Datenstrom in einem binären tcpdump kompatiblen Format zu speichern. Das bringt einerseits den Vorteil, dass viel schneller aufgezeichnet werden kann, da der rohe Datenstrom gespeichert wird, und nicht gefiltert oder

auf den Bildschirm geschrieben werden muss, andererseits besteht die Möglichkeit, die binäre Datei in jeder beliebigen Sniffer Software darzustellen. Selbstverständlich ist es auch möglich, die binäre Datei mit Snort zu einem späteren Zeitpunkt darzustellen.

Der umfangreichste und wichtigste Modus von Snort ist jedoch der Intrusion Detection Modus. In diesem Modus kann Snort nicht nur Pakete registrieren, sondern diese auch aufgrund bestimmter Regeln analysieren. Wie in Kapitel 3.3.2 jedoch bereits beschrieben, ist es notwendig, Snort genau auf das System, auf dem es zum Einsatz kommt, zu konfigurieren, da sonst falsche Alarme ausgelöst werden können, und tatsächliche Bedrohungen ignoriert werden könnten. Die Konfiguration von Snort wird über eine Konfigurationsdatei und eine Reihe von Regeln gehandhabt.

Die Konfigurationsdatei ist dabei in Abschnitte gegliedert. Der erste Abschnitt dient der Konfiguration der Umgebungsvariablen, also Parametern wie interne Netzwerkadresse oder Server, die sich in diesem Netz befinden. Zusätzlich lässt sich noch bestimmen, auf welchen Ports welche Dienste eingesetzt werden. Dies ist hilfreich, wenn non-standard Ports verwendet werden, um Snort mitzuteilen, was überwacht werden soll.
Die nächsten beiden Abschnitte befassen sich mit dynamischen Bibliotheken, sowie Pre-Processor Konfiguration, deren Aufgabe es ist, die Funktionalität von Snort zu erweitern. So können zum Beispiel Pre-Processors eingesetzt werden, um Snort zu ermöglichen, Port Scans oder gewisse Trojaner erkennen zu können.
Der darauf folgende Abschnitt dient der Konfiguration der Ausgabeoptionen. Hier ist es möglich, Snort anzuweisen, wie Alarme ausgegeben werden sollen. Zwei Möglichkeiten wären zum Beispiel die Weitergabe der Alarme an einen Syslog Daemon oder die Ausgabe in eine SQL Datenbank.
Aufgabe des letzten Abschnitts, ist die Konfiguration der zu verwendenden Regeln. Wie bereits oben erwähnt, wird Snort über eine Reihe an Regeln konfiguriert, die dem Programm mitteilen, was überwacht werden soll, und was als Bedrohung angesehen wird. Snort wird bereits mit einer Vielzahl an Regeln ausgeliefert. Diese umfassen viele Bereiche, von der Erkennung von Trojanern bis hin zu Chat oder P2P Programmen. Zusätzlich werden auf `www.snort.org` regelmäßig aktuelle Regeln veröffentlicht, und es besteht natürlich die Möglichkeit, selbst Regeln zu verfassen.

Ist Snort entsprechend konfiguriert, kann es im Intrusion Detection Modus gestartet werden.

```
[root@firewall ~]#  snort -c /etc/snort/snort.conf -i eth1 -b -l /var/log/snort/
```

Wird nun zum Beispiel ein Port Scan von einem anderen Hot aus durchgeführt, so wird von Snort ein Alarm ausgelöst und die Sicherheitsverletzung aufgezeichnet.

```
...
[**] [122:17:0] (portscan) UDP Portscan [**]
xx/xx-13:54:24.454662 192.168.0.2 -> 192.168.0.10
PROTO255 TTL:0 TOS:0xC0 ID:47805 IpLen:20 DgmLen:159
...
```

Abschließend kann Snort noch als Daemon gestartet werden, um ständig im Hintergrund nach Sicherheitsverletzungen suchen zu können.

Wird Snort in einer realen Umgebung eingesetzt, so ist die Konfiguration meist etwas aufwändiger, da es einige Zeit und Erfahrung benötigt, die richtigen Regeln zu aktivieren, und Snort genau so einzustellen, dass es dem Netzwerk, in dem es eingesetzt wird, entspricht. Zur Erleichterung der Konfiguration sowie der Auswertung der Alarme kann Snort auch über ein graphisches Interface angesteuert werden. Zwar bietet Sourcefire kein natives GUI für Snort an, jedoch existiert ein Webmin Plug-In, über das die meisten Funktionen von Snort kontrolliert werden können. Dieses ermöglicht unter anderem *Form-Based* Konfiguration sowie *point and click* Aktivierung bzw. Deaktivierung von Regeln.

(vgl. Howlett 2004)

4.6 HIDS

Die Absicherung durch Firewall und NIDS bietet bereits ein relativ hohes Maß an Sicherheit. Gelingt es einem Eindringling dennoch, Zugriff auf ein System zu erlangen, oder wird ein System von innen her angegriffen, so kann dies weder durch die Firewall verhindert, noch durch ein NIDS entdeckt werden. Um das Sicherheitskonzept noch zu erweitern, wird Snort durch ein HIDS ergänzt. Wie bereits beschrieben, ist es Aufgabe eines HIDS, Verletzungen der Systemintegrität zu detektieren und aufzuzeichnen. Wird also von einem Eindringling eine Passwortdatei manipuliert, so soll dies von einem HIDS erkannt und gemeldet werden.

Auf dem Testsystem kommt Tripwire als HIDS zum Einsatz. Die Installation erfolgt ganz einfach mittels RPM.

```
[root@firewall ~]# wget http://fedoraproject.org/extras/4/i386/tripwire-2.3.1-22.rpm
...
[root@firewall ~]# rpm -vhU tripwire-2.3.1-22.rpm
```

Anschließend kann bereits mit der Konfiguration von Tripwire begonnen werden.

4.6.1 Konfiguration und Funktionsweise

Wie in 3.3.4 bereits erwähnt, ist es notwendig, die Policy Datei für jedes System genau anzupassen, da Tripwire sonst keine sinnvollen Berichte abliefern kann. In der Policy Datei wird festgelegt, welche Dateien überwacht werden sollen, und welcher Sicherheitsstufe die Datei zugeordnet werden soll. Tripwire bietet dafür einige voreingestellte Sicherheitsstufen an:

Stufe	Beschreibung
SEC_CRIT	Critical files that cannot change
SEC_SUID	binaries with the SUID or SGID flags set
SEC_BIN	Binaries that should not change
SEC_CONFIG	Config files that are changed infrequently but accessed often
SEC_LOG	Files that grow, but that should never change ownership

Stufe	Beschreibung
SEC_INVARIANT	Directories that should never change permission or ownership
SIG_LOW	Non-critical files that are of minimal security impact
SIG_MED	Non-critical files that are of significant security impact
SIG_HI	Critical files that are significant points of vulnerability

(Tripwire Inc. 2003)

Zusätzlich lassen sich auch eigene Masken definieren, mit denen festgelegt werden kann, welche Attribute einer Datei überprüft werden sollen

Option	Attribut
a	Letzter Zugriff
b	Anzahl der Blöcke
c	Erstellungsdatum
d	DeviceID des Inodes
g	GID des Besitzers
i	Inode Nummer
l	Wachsende Datei
m	Veränderungsdatum
n	Anzahl der Links

Option	Attribut
p	Permissions
r	Inode Device ID
s	Größe
t	Dateityp
u	UID des Besitzers
C	CRC32 Hashwert
H	Haval Hashwert
M	MD5 Hashwert
S	SHA Hashwert

(vgl. Tripwire Inc. 2003)

Auf diese Weise können alle nur denkbaren Kombinationen von Überwachungsparametern erzeugt werden.

Vorerst wird mit der Standardpolicy fortgefahren und die *Passphrases* erstellt, deren Aufgabe es ist, verschieden Dateien, wie zum Beispiel die Policy Datei oder die Konfigurationsdatei zu verschlüsseln und digital zu signieren.

```
[root@firewall tripwire]# ./twinstall.sh
...
Creating key files...
...
Enter the site keyfile passphrase:
Verify the site keyfile passphrase:

Generating key (this may take several minutes)...Key generation complete.
...
Enter the local keyfile passphrase:
Verify the local keyfile passphrase:
Generating key (this may take several minutes)...Key generation complete.
...
Signing configuration file...
Please enter your site passphrase:
Wrote configuration file: /etc/tripwire/tw.cfg
...
```

```
Signing policy file...
Please enter your site passphrase:
Wrote policy file: /etc/tripwire/tw.pol
```

Anschließend muss die Datenbank initialisiert werden. Das bedeutet, das Tripwire eine Datenbank anlegt, die Informationen über alle Dateien, die überwacht werden sollen, enthält. Danach wird die Datenbank verschlüsselt und dient fortan als Referenz, mit der der aktuelle Zustand bei einem *Integrity Check* verglichen wird.

```
[root@firewall tripwire]# tripwire --init
...
Wrote database file: /var/lib/tripwire/firewall.twd The database
was successfully generated.
```

Nachdem ein *Integrity Check* durchgeführt wurde, wird schnell ersichtlich, warum es so wichtig ist, die Policy Datei peinlich genau an das System anzupassen. Die Überprüfung hatte eine große Zahl an Verletzungen, verursacht durch fehlende Dateien zur Folge. Der Grund dafür ist, dass die Standard Policy viele Einträge über Dateien enthält, die auf diesem System gar nicht existieren.
Aus Gründen der Übersichtlichkeit, und um die Funktion von Tripwire besser demonstrieren zu können, wird sich die Policy, neben der standardmäßigen Überwachung der Tripwire Programmdateien, auf die Überwachung einer einzigen Datei (`/root/testfile`), beschränken. Dazu muss, nach entfernen aller nicht benötigten Einträge, folgende Passage in die Policy Datei eingefügt werden:

```
# Test files

(
  rulename = "Test␣files",
  severity = $(SIG_HI)
)
{
     /root/testfile                          -> $(SEC_CRIT) ;
}
```

Der Eintrag bewirkt, dass eine Gruppe namens *„Test files“* eingeführt wird, deren Aufgabe es ist, die Datei `/root/testfile` zu überwachen. Die Datei wurde in die Sicherheitsstufe `SEC_CRIT` (siehe oben) eingestuft.

Anschließend ist es notwendig, die Policy Datei zu aktualisieren, das bedeutet, dass aus der eben editierten Klartextdatei `twpol.txt` die verschlüsselte Policy Datei `tw.pol` erstellt wird. Um die Änderungen in die Datenbank zu übernehmen, muss zuerst die alte gelöscht, und dann eine neue Datenbank initialisiert werden.

```
[root@firewall tripwire]# twadmin --create-polfile /etc/tripwire/twpol.txt
Please enter your site passphrase:
Wrote policy file: /etc/tripwire/tw.pol

[root@firewall tripwire]# rm /var/lib/tripwire/firewall.twd
rm: remove regular file ‘/var/lib/tripwire/firewall.twd’? y

[root@firewall tripwire]# tripwire --init
Please enter your local passphrase:
Parsing policy file: /etc/tripwire/tw.pol
Generating the database...
*** Processing Unix File System ***
```

```
Wrote database file: /var/lib/tripwire/firewall.twd
The database was successfully generated.
```

Nachdem die neue Datenbank erstellt wurde, kann ein *Integrity Check* durchgeführt werden.

```
[root@firewall tripwire]# tripwire --check
...
-------------------------------------------------------------------------------
  Section: Unix File System
-------------------------------------------------------------------------------

  Rule Name                       Severity Level    Added    Removed  Modified
  ---------                       --------------    -----    -------  --------
  Tripwire Binaries               100               0        0        0
* Tripwire Data Files             100               1        0        0
  Test files                      100               0        0        0
  (/root/testfile)

Total objects scanned:  12
Total violations found:  1
...
Added:
"/var/lib/tripwire/firewall.twd"
```

Das Ergebnis zeigt, das die Testdatei `/root/testfile` nicht geändert oder gelöscht wurde. Weiters ist zu erkennen, dass in der Kategorie *Tripwire Data Files* eine Datei (`/var/lib/tripwire/firewall.twd`) hinzugefügt wurde. Dies erklärt sich dadurch, dass die Datenbankdatei `firewall.twd` logischerweise erst geschrieben wurde, *nachdem* das Datenbankupdate vollzogen war. aus diesem Grund wird sie als neu erkannt. Um diese Verletzung zu beseitigen, muss die Datenbank mit folgendem Befehl aktualisiert werden, wobei anstelle von `<name>` der Name des aktuellsten Reports angegeben werden muss.

```
[root@firewall tripwire]# tripwire --update --twrfile /var/lib/tripwire/report/<name>
```

Tripwire zeigt eine Liste, in der alle Verletzungen verzeichnet sind. Diese können an dieser Stelle akzeptiert oder abgelehnt werden. Wird die aufgetretene Verletzung akzeptiert, so zeigt Tripwire bei einem erneuten *Integrity Check* keinerlei Verletzungen mehr an.

```
-------------------------------------------------------------------------------
  Section: Unix File System
-------------------------------------------------------------------------------

  Rule Name                       Severity Level    Added    Removed  Modified
  ---------                       --------------    -----    -------  --------
  Tripwire Binaries               100               0        0        0
  Tripwire Data Files             100               0        0        0
  Test files                      100               0        0        0
  (/root/testfile)

Total objects scanned:  13
Total violations found:  0
```

4.6.2 Eindringlingserkennung mit Tripwire

Nachdem sich das System aus der Perspektive von Tripwire nun in einem als sicher befundenem Zustand befindet, soll als nächstes ein Eingriff simuliert werden. Die überwachte Datei `/root/testfile`, ist ein Dokument mit folgendem Inhalt:

```
[root@firewall tripwire]# cat /root/testfile
This file is being watched!
```

Zuerst soll der Inhalt der Datei verändert werden. Dies könnte zum Beispiel der Fall sein, wenn ein Eindringling die Datei `/etc/passwd` verändert, um sich selbst ein Benutzerkonto anzulegen.

```
[root@firewall tripwire]# echo 'h4x0r was here!' >> /root/testfile
[root@firewall tripwire]# cat /root/testfile
This file is being watched!
h4x0r was here!
```

Anschließend wird ein weiterer *Integrity Check* durchgeführt, um zu überprüfen, ob Tripwire die Veränderung registriert.

```
-------------------------------------------------------------------------------
  Section: Unix File System
-------------------------------------------------------------------------------

  Rule Name                       Severity Level    Added    Removed  Modified
  ---------                       --------------    -----    -------  --------
  Tripwire Binaries               100               0        0        0
  Tripwire Data Files             100               0        0        0
* Test files                      100               0        0        1
  (/root/testfile)

Total objects scanned:  13
Total violations found:  1
...
Modified:
"/root/testfile"
```

Wie zu erwarten, registriert Tripwire die Datei als modifiziert. Das gleiche Ergebnis tritt auf, wenn zum Beispiel die Zugriffsrechte oder die Besitzverhältnisse der Datei geändert werden. Waren diese Änderungen gewollt, so müssen einfach die Datenbank aktualisiert, und die Änderungen akzeptiert werden. Danach zeigt Tripwire wieder keine Verletzungen an.
Beim Aktualisieren der Datenbank ist jedoch Vorsicht geboten. Sie darf nur erneuert werden, wenn zu hundert Prozent sicher ist, dass keine Verletzung aufgetreten ist, da diese sonst nach der Aktualisierung für Tripwire nicht mehr erkennbar wäre.

Tripwire ist ein relativ einfach zu bedienendes Programm, das sehr gut geeignet ist, um die Dateiintegrität eines Systems zu schützen. Durch die Ergänzung der Firewall und des NIDS durch ein HIDS, befindet sich das System bereits in einem sehr sicheren Zustand. Um ein ganzes Netzwerk abzusichern, müsste ein HIDS allerdings auf jedem zu überwachenden Rechner installiert sein.

4.7 Port Scanner

Wie in Kapitel 3.4 bereits beschrieben, sind Portscanner Werkzeuge, die dazu dienen, den Zustand der Ports eines Systems aufzuzeigen. Einer der vielseitigsten und am meisten benützten Vertreter dieser Kategorie, Nmap (3.4.1), soll im folgenden auf seine Funktionen und seine Möglichkeiten hin analysiert werden.

4.7.1 Security mit Nmap

Im Alltag einer Security Fachkraft ist Nmap ein unverzichtbares Werkzeug. Es ermöglicht das Aufzeigen der Zustände aller Ports eines Systems. Nachdem Nmap oft von Hackern eingesetzt wird, ist es ratsam, dieses Programm auch für Sicherheitsüberprüfungen zu verwenden.
Um sicher zu gehen, dass die Ergebnisse des Port Scans der realen Situation entsprechen, sollte ein Scan möglichst von der Perspektive eines potentiellen Angreifers, also von einem externen System aus, durchgeführt werden. Im konkreten Fall von einem Rechner der *Filiale* (siehe Netzplan, 4.4.1).

Soll ein Scan durchgeführt werden, so wird Nmap mit folgenden Parametern aufgerufen: `nmap [Scan Type(s)] [Options] target specification` (Fyodor 2005). Wie bereits erwähnt, hat Nmap viele verschiedene Scan Modi (siehe 3.4.1). Der effizienteste ist dabei der TCP SYN Scan (`-sS`), welcher auch im konkreten Fall zur Anwendung kommen soll. Zusätzlich wird auch ein UDP Port Scan (`-sU`) durchgeführt. Als Optionen werden `-sV -O -T3 -v` gewählt. Dies bewirkt, dass auf offenen Ports eine Diensterkennung durchgeführt wird (`-sV`), dass eine Betriebsystemerkennung (`-O`) versucht wird, und dass nicht mit maximaler Geschwindigkeit gescannt wird (`-T3`), um etwaige Leitungsüberlastungen zu vermeiden. Der letzte Parameter (`-v`) betrifft die Ausgabe von Nmap und bewirkt, dass das Ergebnis detaillierter dargestellt wird. Bei der Angabe der Zielparameter wird die externe Adresse des Servers (`192.168.0.10`) gewählt, sowie das gesamte Spektrum aller Ports (`-p1-65535`). Der gesamte Befehl zur Initialisierung des gewünschten Scans nimmt also folgende Form an:

```
nmap -sS -sU -sV -O -T3 -v 192.168.0.10 -p1-65535
```

Bei diesem Durchlauf wurde der Scan jedoch nach etwa 10 Minuten vom Benutzer abgebrochen, da Nmap eine geschätzte Scan Dauer von über sechzehn Stunden angab. Grund dafür ist der UDP Port Scan. Aus in 3.4.1 bereits beschriebenen Gründen, kann ein UDP Scan sehr viel Zeit in Anspruch nehmen. Aus diesem Grund ist es ratsam, nur kleine Gruppen von UDP Ports auf einmal zu scannen. Um dennoch einen vollständigen TCP Scan zu erhalten, wird der Scan in zwei Teile unterteilt. Um einen reinen TCP Scan durchzuführen, muss die Syntax wie folgt abgeändert werden:

```
nmap -sS -sV -O -T3 -v 192.168.0.10 -p1-65535
```

Der resultierende Scan läuft dann sehr schnell ab:

```
Starting Nmap 4.10 ( http://www.insecure.org/nmap ) at 2006-xx-xx 13:34 CEST

Initiating ARP Ping Scan against 192.168.0.10 [1 port] at 13:34
The ARP Ping Scan took 0.19s to scan 1 total hosts.
DNS resolution of 1 IPs took 0.05s.
```

```
Initiating SYN Stealth Scan against 192.168.0.10 [65535 ports] at 13:34
Discovered open port 22/tcp on 192.168.0.10
Discovered open port 5850/tcp on 192.168.0.10
Discovered open port 5950/tcp on 192.168.0.10
The SYN Stealth Scan took 14.69s to scan 65535 total ports.

Initiating service scan against 3 services on 192.168.0.10 at 13:35
The service scan took 6.80s to scan 3 services on 1 host.

For OSScan assuming port 22 is open, 1 is closed, and neither are firewalled
Host 192.168.0.10 appears to be up ... good.

Interesting ports on 192.168.0.10:
Not shown: 65532 closed ports

PORT        STATE   SERVICE     VERSION
22/tcp      open    ssh         OpenSSH 4.0 (protocol 2.0)
5850/tcp    open    vnc-http    Ultr@VNC (Name p4nb;
                                Resolution 1400x1082; VNC TCP port: 5950)
5950/tcp    open    vnc         VNC (protocol 3.6)

MAC Address: 00:0E:2E:7E:29:93 (Edimax Technology Co.)
Device type: general purpose
Running: Linux 2.6.X
OS details: Linux 2.6.0-test5 x86, Linux kernel 2.6.5 - 2.6.8
Uptime 0.036 days (since Thu xxx xx 12:43:37 2006)

TCP Sequence Prediction: Class=random positive increments
                         Difficulty=2899755 (Good luck!)
IPID Sequence Generation: All zeros

Nmap finished: 1 IP address (1 host up) scanned in 24.781 seconds
               Raw packets sent: 65553 (2.885MB) | Rcvd: 65549 (3.015MB)
```

Nach einem nur wenige Sekunden dauernden TCP SYN Scan, dem Service Scan, sowie dem OS Scan, liefert Nmap eine Zusammenfassung aller analysierten Ports. Wie anhand der Firewallkonfiguration (4.4.4) zu erwarten war, wird TCP Port 22 als `open` erkannt. Die entsprechende Version des Dienstes (OpenSSH 4.0, SSHv2) wurde ebenfalls korrekt zugeordnet. Die beiden in das interne Netz weitergeleiteten Ports 5850 und 5950 werden von Nmap auch richtigerweise als `open` erkannt. Wiederum funktionierte auch die Zuordnung des Dienstes problemlos. Alle weiteren Ports wurden, aufgrund des `-reject-with tcp-reset` Parameters der Firewallkonfiguration (siehe 4.4.4), als `closed` klassifiziert. Folglich ist aus diesem Ergebnis nicht erkennbar, dass am gescannten Host eine Firewall in Betrieb ist.
Weiters war es Nmap möglich, sowohl das Betriebsystem (Linux), als auch die Plattform (x86), und sogar eine relativ genaue Einschätzung der kernel Version (2.6.5 - 2.6.8), sowie die Uptime (0.036 Tage) in Erfahrung zu bringen. Ein potentieller Angreifer hätte also bereits nach einem einfachen Port Scan eine nicht unbeträchtliche Anzahl an wichtigen Informationen.

Um das Ergebnis zu vervollständigen, wird noch ein UDP Port Scan durchgeführt, der diesmal jedoch auf die *well-known-ports* (1-1024) beschränkt wird. Die Syntax muss also wieder entsprechend abgeändert werden:

```
nmap -sU -sV -O -T3 -v 192.168.0.10 -p1-1024
```

Nach der Beschränkung auf die ersten 1024 UDP Ports dauerte der Scan nur noch einige Sekunden, der Dienst Scan allerdings hätte einige Stunden beansprucht. Das liegt daran, dass dieser Scan nur auf als `open` deklarierte Ports angewandt wird. Da bei UDP Scans

jedoch auch gefilterte Ports als `open|filtered` aufscheinen, wird der Dienst Scan für alle 1024 gescannten Ports durchgeführt, auch, wenn kein einziger Port wirklich geöffnet war. Aus diesem Grund wurde der Dienst Scan, für UDP Ports nicht durchgeführt. Um zu erreichen, dass die Ports als `closed` erscheinen, müsste einfach die Firewall konfiguriert werden, auf eine UDP Anfrage mit einem ICMP Port Unreachable zu antworten.

```
Starting Nmap 4.10 ( http://www.insecure.org/nmap ) at 2006-08-10 20:22 CEST

Initiating ARP Ping Scan against 192.168.0.10 [1 port] at 20:22
The ARP Ping Scan took 0.17s to scan 1 total hosts.
DNS resolution of 1 IPs took 0.06s.

Initiating UDP Scan against 192.168.0.10 [1024 ports] at 20:22
The UDP Scan took 23.58s to scan 1024 total ports.

Host 192.168.0.10 appears to be up ... good.

All 1024 scanned ports on 192.168.0.10 are open|filtered

MAC Address: 00:0E:2E:7E:29:93 (Edimax Technology Co.)

Nmap finished: 1 IP address (1 host up) scanned in 24.359 seconds
               Raw packets sent: 2049 (57.386KB) | Rcvd: 1 (42B)
```

Mit Nmap ist es auf relativ einfache Weise möglich, Informationen über ein System zu erlangen. Da Angreifer dieses Tool häufig nutzen, ist es sinnvoll, bei Sicherheitsüberprüfungen Nmap einzusetzen, um bereits im Voraus in Erfahrung zu bringen, welche Informationen einem potentiellen Angreifer preisgegeben werden, und entsprechend zu reagieren.

4.8 Anti-Viren Software

Als Anti-Viren Software wurde für den Praxistest ClamAV gewählt. Zum einen, weil die in Kapitel 3.6.3 durchgeführte Analyse den Schluss zuließ, es handle sich um ein seriöses Produkt, und zum anderen mangels Alternativen.
Die Installation verlief, dank RPM, einfach und problemlos. Nach abgeschlossenem Installationsvorgang startet ClamAV automatisch einen Update Prozess, um die Datenbank mit den neuesten Viren Signaturen zu ergänzen.

```
[root@firewall ~]# rpm -vhU clamav-0.88.4-1.i386.rpm
warning: clamav-0.88.4-1.i386.rpm: Header V3 DSA signature: NOKEY, key ID 6cdf2cc1
Preparing...                ########################################### [100%]
   1:clamav                 ########################################### [100%]
...
ClamAV update process started at Sat xxx xx 14:13:29 2006
Querying current.cvd.clamav.net
TTL: 634
Software version from DNS: 0.88.4
Retrieving http://db.at.clamav.net/main.cvd
Downloading main.cvd [*]
main.cvd updated (version: 39, sigs: 58116, f-level: 8, builder: tkojm)
Retrieving http://db.at.clamav.net/daily.cvd
Downloading daily.cvd [*]
daily.cvd updated (version: 1649, sigs: 7081, f-level: 8, builder: ccordes)
Database updated (65197 signatures) from db.at.clamav.net (IP: 81.223.20.171)
```

4.8.1 Konfiguration und Betrieb

Clamscan, der Stand-Alone Viren Scanner des ClamAV Pakets, funktioniert sofort nach der Installation ohne jegliche Konfiguration. Zum Testen liegen dem Paket einige Dateien bei, die Test Signaturen enthalten, die von Clamscan als Viren erkannt werden. Diese sind noch zusätzlich in verschiedene Datei- und Archivtypen (`rar, cab und zip` verpackt, um die Fähigkeiten der Scan Engine demonstrieren zu können.

Funktionstest

Um die Funktionalität von ClamAV zu überprüfen, wird ein Scan der beiliegenden Testsignaturen durchgeführt.

```
[root@firewall ~]# clamscan ~/clamav-0.88.4

[root@firewall ~]# clamscan ~/clamav-0.88.4/test/
/root/clamav-0.88.4/test/clam.rar: ClamAV-Test-File FOUND
/root/clamav-0.88.4/test/clam.exe.bz2: OK
/root/clamav-0.88.4/test/clam.cab: ClamAV-Test-File FOUND
/root/clamav-0.88.4/test/README: OK
/root/clamav-0.88.4/test/clam-error.rar: RAR module failure
/root/clamav-0.88.4/test/clam.exe: ClamAV-Test-File FOUND
/root/clamav-0.88.4/test/clam.zip: ClamAV-Test-File FOUND

----------- SCAN SUMMARY -----------
Known viruses: 65610
Engine version: 0.88.4
Scanned directories: 1
Scanned files: 7
Infected files: 4
Data scanned: 0.00 MB
Time: 4.549 sec (0 m 4 s)
```

Die Überprüfung geht recht schnell vonstatten, und ClamAV hat erfolgreich alle platzierten Signaturen identifiziert. Nach Beendigung des Scans, gibt das Programm eine Übersicht der infizierten Dateien aus, sowie Statistiken zum Scanvorgang.

Um `freshclam`, das Update Tool von ClamAV, oder `clamd`, den Daemon Prozess, verwenden zu können, müssen zuerst die entsprechenden Konfigurationsdateien editiert werden.

Aktualität ist für Anti-Viren Programme stets am wichtigsten. Aus diesem Grund soll zuerst `freshclam` konfiguriert werden. Die minimale Konfiguration, die vorgenommen werden muss, damit `freshclam` funktioniert, ist das Wort `Example` in der Datei `freshclam.conf` zu entfernen bzw. auszukommentieren. Ansonsten funktioniert das Programm mit Standard Einstellungen einwandfrei. Neben dem Speicherort der Signaturendatenbank, und den Aktualisierungshäufigkeit, lassen sich noch einige Optionen konfigurieren, die sich mit Logging beschäftigen. Außerdem können noch Proxyeinstellungen gesetzt werden, und konfiguriert werden, ob `freshclam` den Daemon davon informieren soll, wenn die Datenbank aktualisiert wurde.
Sind alle Einstellungen vorgenommen, kann `freshclam` gestartet werden. Vorerst sollte das Programm im interaktiven Modus betrieben werden, um festzustellen, ob es korrekt funktioniert.

```
[root@firewall ~]# freshclam
ClamAV update process started at Wed xxx xx 14:00:49 2006
SECURITY WARNING: NO SUPPORT FOR DIGITAL SIGNATURES
```

```
See the FAQ at http://www.clamav.net/faq.html for an explanation.
main.cvd is up to date (version: 39, sigs: 58116, f-level: 8, builder: tkojm)
Downloading daily.cvd [*]
daily.cvd updated (version: 1668, sigs: 7494, f-level: 8, builder: ccordes)
Database updated (65610 signatures) from database.clamav.net (IP: 129.27.65.27)
```

Nachdem die Aktualisierung erfolgreich abgeschlossen wurde, kann `freshclam` im Daemon Modus gestartet werden. Dies funktioniert entweder mittels `freshclam -d` oder über einen Cronjob.

Als nächster Schritt kann `clamd` konfiguriert werden. In der Konfigurationsdatei befinden sich wiederum einige Optionen, die das Logging beeinflussen. Zusätzlich lässt sich hier jedoch genau steuern, was bei einem Viren Scan untersucht werden soll. So lässt sich zum Beispiel konfigurieren, ob Archive oder Macros in MS Office Dokumenten gescannt werden sollen. Außerdem lassen sich noch einige Einstellungen für *On-Access Scan* vornehmen, für den allerdings ein eigenes Modul installiert werden muss. Wie bei der Konfiguration von `freshclam`, muss auch hier das Wort `Example` aus der Konfigurationsdatei entfernt werden, da `clamd` sonst den Start verweigert.
Ist die Konfiguration abgeschlossen, kann `clamd` gestartet werden. Um Scans durchzuführen, kann direkt via TCP/IP zum Daemon verbunden und eine Reihe von Befehlen aufgerufen werden (siehe 3.6.3). Eine andere Möglichkeit, den Daemon zu Nutzen, ist `clamdscan`, ein Client, über den sich die Scan Engine von `clamd` nützen lässt. `clamdscan` hat prinzipiell die gleichen Fähigkeiten wie `clamscan`, mit dem Unterschied, dass es sich nicht um eine Stand-Alone Software handelt, sondern `clamdscan` auf `clamd` für seine Scans zurückgreift. Zum Vergleich wurde noch einmal das gleiche Testverzeichnis gescannt. Am Ergebnis lässt sich erkennen, dass der Scan mit `clamdscan` deutlich schneller vonstatten ging, als der der Stand-Alone Variante.

```
[root@firewall clamav-0.88.4]# clamdscan -v test/
/root/clamav-0.88.4/test//clam.rar: ClamAV-Test-File FOUND
/root/clamav-0.88.4/test//clam.cab: ClamAV-Test-File FOUND
/root/clamav-0.88.4/test//clam.exe: ClamAV-Test-File FOUND
/root/clamav-0.88.4/test//clam.zip: ClamAV-Test-File FOUND

----------- SCAN SUMMARY -----------
Infected files: 4
Time: 0.021 sec (0 m 0 s)
```

ClamAV ist ein sehr einfaches und dennoch sehr vollständiges Anti-Viren Programm. Es verfügt über einen Stand-Alone Scanner sowie über einen Hintergrund Daemon und ein Tool für automatische Updates. Die Möglichkeit, verschiedene Module Dritter einzubinden, macht ClamAV zu einem äußerst vielseitigen Werkzeug. Es stehen Module für die Integration mit praktisch jedem MTA, POP3 oder Webproxy zur Verfügung, sowie Module zur Integration mit verschiedenen anderen Software Produkten. Zusätzlich gibt es auch eine Reihe graphischer Interfaces.

4.9 Verschlüsselung

Nachdem das System gegen Eingriffe von außen abgesichert wurde, gilt es noch sicherzustellen, dass Informationen, die die sichere Umgebung verlassen, auf eine Weise geschützt werden können, dass Unbefugte diese weder einsehen, noch verändern können. Um das zu ermöglichen, kommen Verschlüsselungstechnologien zum Einsatz.

Sollen Schlüssel erstellt, Hashwerte von Dokumenten erzeugt, Dateien verschlüsselt oder gar Zertifikate ausgestellt werden, so ist OpenSSL das geeignete Software Paket. Es ist bei den meisten Linux Distributionen bereits vorinstalliert, da viele Programme, wie zum Beispiel der Apache Webserver, OpenSSL nutzen, um sichere Verbindungen zu ermöglichen.
Neben der OpenSSL Library, enthält das OpenSSL Paket (Version 0.9.7f) auch ein Command Line Interface (CLI), das es ermöglicht, alle Funktionen der Library über Kommandozeilenbefehle zu nutzen.

4.9.1 OpenSSL CLI

Das OpenSSL CLI bietet eine Vielzahl an Funktionen, vom Erstellen von RSA Schlüssel, über symmetrische Verschlüsselung, bis hin zu Zertifikaten. Im Folgenden werden diese Funktionen analysiert und praktisch eingesetzt.

Public Key Verschlüsselung

OpenSSL bietet die Möglichkeit, ein asymmetrisches RSA Schlüsselpaar zu erstellen, das anschließend verwendet werden kann, um Dokumente mit einem Public Key Verfahren zu verschlüsseln. Dazu muss zuerst ein privater Schlüssel erstellt werden.

```
[root@firewall openssl]# openssl genrsa -out private.pem 1024
```

Der erstellte 1024bit Schlüssel wird in der Datei `private.pem` abgelegt. Anschließend wird aus dem privaten Schlüssel ein öffentlicher Schlüssel generiert.

```
[root@firewall openssl]# openssl rsa -in private.pem -out public.pem \
                         -outform PEM -pubout
```

Dieser wird in der Datei `public.pem` gespeichert. Um nun Dateien mit RSA zu verschlüsseln, bietet OpenSSL das Tool RSA Utility. Zum Test der Verschlüsselung wird zuerst eine Datei erstellt, in die Text geschrieben wird.

```
[root@firewall openssl]# echo 'Nicht weitersagen!!!' > bigsecret
```

Danach kann die Datei mit RSA, unter Zuhilfenahme des öffentlichen Schlüssels verschlüsselt werden.

```
[root@firewall openssl]# openssl rsautl -encrypt -inkey public.pem \
                         -pubin -in bigsecret -out bigsecret.enc
```

Der Inhalt der Datei `bigsecret` steht nun verschlüsselt in der Datei `bigsecret.enc`:

```
[root@firewall openssl]# less bigsecret.enc
"bigsecret.enc" may be a binary file.  See it anyway?
^C,^O<D1><87><88>-OA^Zm‘=Gk<EB>O<C6>~<E6><87><DF>^N<AD><99>M}=qTCBmS^V:<B3>j<A4>
<F5><F8><A4><AC><FC>^OK^Y<E7><F7>^G=<80><9A><E7>/<DA><DA>T^X^Eny<F0><BC>i*]<FA>
^KK<AE>v<F4><80><84>^N<D8><A1><D8><C7><E8><83>T<F5>hr<C7><8A><8A><B1>/^N<BE><8A>
R<CA>|]<B9>^L<88><A7><EC>^L<BF>]<CD><A9>_<D2><C1>.w*<84><8C><E2><85>F^R<F6><F4>
'<C2><81><E3><AE>
```

Um die Datei wieder zu entschlüsseln, muss der private Schlüssel angewendet werden.

```
[root@firewall openssl]# openssl rsautl -decrypt -inkey private.pem \
                         -in bigsecret.enc -out bigsecret.dec
```

Die Datei `bigsecret.dec` sollte nun wieder den gleichen Inhalt besitzen wie die Datei `bigsecret`. Um dies zu beweisen, kann einfach ein MD5 Hash der Datei erstellt werden.

```
[root@firewall openssl]# openssl dgst -md5 bigsecret
MD5(bigsecret)= de1908a296541ddfab92eb53925c5f24
[root@firewall openssl]# openssl dgst -md5 bigsecret.dec
MD5(bigsecret.dec)= de1908a296541ddfab92eb53925c5f24
```

Die Tatsache, dass die beiden Hashwerte identisch sind, beweist, dass `bigsecret.dec` wieder den gleichen Inhalt aufweist wie `bigsecret`.
An dieser Stelle muss allerdings noch einmal erwähnt werden, dass RSA kaum zur Verschlüsselung von größeren Datenmengen, sondern praktisch nur zur Verschlüsselung symmetrischer Schlüssel eingesetzt wird. Dass hier eine Datei verschlüsselt wurde, diente nur dem Zweck der besseren Veranschaulichung.

Symmetrische Verschlüsselung

Neben Public Key Verfahren, können mir OpenSSL natürlich auch symmetrische Verschlüsselungen realisiert werden. Das Paket beinhaltet eine Vielzahl verschiedener Algorithmen, wie zum Beispiel DES, TripleDES oder AES.

Um die Datei `bigsecret` in 256bit AES zu verschlüsseln muss folgender Befehl, gefolgt von einem Passwort angegeben werden.

```
[root@firewall openssl]# openssl enc -aes-256-cbc -salt -in bigsecret \
                         -out bigsecret.symenc
```

Um die Datei `bigsecret.symenc` wieder zu entschlüsseln, muss natürlich wiederum das vorher festgelegte Passwort angegeben werden.

```
[root@firewall openssl]# openssl enc -d -aes-256-cbc -in bigsecret.symenc \
                         -out bigsecret.symdec
```

Über den MD5 Hashwert kann wieder bestätigt werden, dass die Datei korrekt entschlüsselt wurde.

```
[root@firewall openssl]# openssl dgst -md5 bigsecret
MD5(bigsecret)= de1908a296541ddfab92eb53925c5f24
[root@firewall openssl]# openssl dgst -md5 bigsecret.symdec
MD5(bigsecret.dec)= de1908a296541ddfab92eb53925c5f24
```

Symmetrische Verschlüsselungen werden, aufgrund ihrer hohen Geschwindigkeit, meist zum Verschlüsseln größerer Datenmengen eingesetzt.

Message Digests

Wie bereits in den beiden letzten Abschnitten demonstriert, unterstützt OpenSSL auch das erstellen von Message Digests. Dies ist hilfreich, wenn man Dateien zum Download anbieten möchte, und anderen die Möglichkeit geben will, die Integrität der Datei zu überprüfen.

```
[root@firewall openssl]# openssl dgst -md5 testfile
MD5(testfile)= 8d788dedcb0650a94998d06e009d0320
```

Zum Erstellen dieser Hashwerte stehen neben MD5 noch MD4, RMD160, SHA und SHA1 zur Verfügung.
Um neben der Integrität auch noch Authentizität zu bieten, kann der gewonnene Hashwert auch gleich mit dem privaten Schlüssel verschlüsselt bzw. signiert werden.

```
[root@firewall openssl]# openssl dgst -sha1 -sign private.pem \
                         -out testfile.sha1 testfile
```

Die Authentizität kann dann ganz einfach mit Hilfe des öffentlichen Schlüssels überprüft werden.

```
[root@firewall openssl]# openssl dgst -sha1 -verify public.pem \
                        -signature testfile.sha1 ../testfile
```

Passwort Hashes

Eine weitere Funktion von OpenSSL, ist die Möglichkeit, Passwort Hashes im Stil von `/etc/passwd`,

```
[root@firewall openssl]# openssl passwd Passwort
PbER4NpYIL6Fk
```

bzw. auch im Stil von `/etc/shadow` zu erstellen.

```
[root@firewall openssl]# openssl passwd -1 Passwort
$1$wA1gC/Ls$8ekT.hvZyxyIwCdyHYpJH1
```

Zertifikate mit OpenSSL

Ein wichtiger Punkt von OpenSSL, ist die Fähigkeit, Zertifikate auszustellen. Das folgende Beispiel erstellt ein selbst unterschriebenes Zertifikat.

```
[root@firewall openssl]# openssl req -x509 -nodes -days 365 -newkey rsa:1024 \
                        -keyout cert.pem -out cert.pem
```

Anschließend werden einige Daten zum Zertifikatshalter, wie zum Beispiel Name der Firma oder der Standort, abgefragt. Um das erstellte Zertifikat zu testen, kann ein Server gestartet werden, auf den via Web Browser und HTTPS verbunden werden kann.

```
[root@firewall openssl]# openssl s_server -cert cert.pem -www
```

Zusätzlich zu allen erwähnten Funktionen, bietet das OpenSSL Paket auch einen Client, mit dem zu den verschiedensten sicheren Diensten (zum Beispiel HTTPS) verbunden werden kann. Diese Funktion dient hauptsächlich dem Test dieser Dienste.

Im nächsten und letzten Abschnitt des praktischen Teils der Arbeit, wird eine Sicherheitsüberprüfung durchgeführt, die zeigen soll, ob das System den geforderten Sicherheitsbedürfnissen entspricht.

4.10 Sicherheitsüberprüfung

Nachdem das System nun mit den entsprechenden Sicherheitstechnologien ausgestattet wurde, gilt es festzustellen, ob diese einem Einbruchsversuch standhalten. Das suchen von Lücken in einem IT System, sowie das versuchte Eindringen in dieses, zum Zwecke der Überprüfung der Sicherheitsvorkehrungen, wird Sicherheitsüberprüfung genannt. Ziel dieses Vorhabens ist es, zu beweisen oder zu widerlegen, dass das System den Anforderungen, die der Test an dieses stellt, entspricht.

Im Folgenden sollen die Methoden erläutert werden, mit denen der Test durchgeführt werden wird.

4.10.1 Werkzeuge

Es gibt eine Vielzahl verschiedener Kriterien, anhand denen ein System als *sicher* eingestuft werden kann. All diese haben jedoch zum Ziel, ein System auf mögliche Lücken zu untersuchen, und diese auszunutzen, um möglicherweise Zugriff zu erlangen.
Eine Möglichkeit, um diesen Prozess zu beschleunigen, ist die Verwendung automatisierter Tools.

Vulnerability Scanner

Wie in 3.5 bereits beschrieben, sind Vulnerability Scanner Programme, die eine Vielzahl an bekannten Schwächen automatisch überprüfen können. Einer der bekanntesten Vertreter dieser Gruppe ist Nessus (siehe 3.5.1).

Im Folgenden soll Nessus eingesetzt werden, um einen Überblick über mögliche Sicherheitsrisiken zu geben.

Nessus

Der Vulnerability Scanner Nessus bietet de Möglichkeit, aus einem breiten Spektrum bereits vordefinierter Test zu wählen. Diese beinhalten betriebssystemspezifische sowie generische Test, Ports Scans und vieles mehr.
Um ein besseres Bild des Sicherheitszustandes des System zu erlangen, wird Nessus mehrmals ausgeführt: Die ersten beiden Male von einem externen Rechner, um den Angriff eines externen Eindringlings nachzustellen, wobei die Firewall einmal aktiv, und einmal inaktiv sein

wird. Der dritte Versuch soll am System selbst durchgeführt werden, um Sicherheitslücken aufzuspüren, die ein Angreifer, der bereits Zugang zum System hat, ausnützen könnte.

Für alle drei Testläufe werden alle für Linux relevanten Tests, sowie alle generischen Tests und alle gefährlichen Exploits, die zu Systemabstürzen führen können, aktiviert.

Externer Scan mit Firewall Wird Nessus von einem externen Host ausgeführt, und ist die Firewall aktiv, so liegen keine Ergebnisse vor. Ein potentieller Eindringling hätte keine Verwertbaren Informationen zum gescannten System, und könnte von diesem Punkt aus nicht weiter. Der einzig mögliche Schritt wäre, zu versuchen, auf anderem Wege an Informationen über das System zu kommen.

Externer Scan ohne Firewall Selbst mit deaktivierter Firewall, wird nur ein offener Port, der SSH Dienst, sichtbar. Nessus ist mit einem Port nicht in der Lage, einen akkuraten Betriebssystem Fingerabdruck zu erkennen und tippt auf *Infoblox DNSone* oder *MikroTik Router*. Der einzige Angriffspunkt, der einem Eindringling bleibt, ist die Tatsache, dass Nessus die Version des SSH Protokolls erkannt hat. Der nächste Schritt eines Hackers wäre es nun, sich über Schwachstellen in diesem Protokoll zu Informieren, und zu versuchen, diese auszunutzen.

Lokaler Scan Um auch alle Aspekte des Systems auf mögliche Lücken zu überprüfen, wird der dritte Scan am System selbst durchgeführt. Nach dem ersten Durchlauf meldet Nessus 39 Sicherheitsverletzungen. Diese stellten sich jedoch alle als Updatewarnungen heraus. Nur wenige der zu aktualisierenden Pakete boten Angriffsflächen für mögliche Eindringlinge.
Nach einem vollständigen Systemupdate mittels `yum update`, konnte auch der lokale Nessus Scan nur noch den offenen SSH Port finden.

Raccess

raccess ist, wie Nessus, ein Vulnerability Scanner, der eine Reihe von bekannten Exploits untersucht, und versucht, diese auszunutzen. Findet raccess einen Weg, Root Zugang auf ein System zu erhalten, so wird dem Benutzer auch gleich eine Shell präsentiert. Nach eigenen Angaben wurde das Tool nicht für Hacker entwickelt, da es keinerlei Stealth Modi besitzt, und damit sehr auffällig ist. (vgl. Ramos 2002) Beim Testlauf wurde das Tool direkt auf dem Testsystem ausgeführt. Außer dem offenen SSH Port, wurde jedoch keine Schwachstelle gefunden.

GFI LANguard

GFI LANguard ist zwar ein kommerzielles Tool, jedoch wird es auch nur zum Test des implementierten Sicherheitssystem herangezogen, und ist nicht Teil dieses. Es besteht also kein Konflikt zwischen den Zielen, die diese Arbeit verfolgt, und dem Einsatz dieser Software.

Wie die meisten kommerzielle Sicherheitsüberprüfungstools, ist GFI LANguard nicht kostenlos. Allerdings stand, im Unterschied zu vielen anderen, eine zeitlich begrenzte Testversion zur Verfügung. Preislich liegen die meisten kommerziellen Varianten bei mehreren Tausend Dollar.
Der Scan läuft bei LANguard vollständig automatisch ab. Es kann aus einer Reihe von Presets gewählt werden, oder selbst bestimmt werden, welche Ports und Vulnerabilities

gescannt werden sollen. Nach der Durchführung des Scans zeige LANguard zwölf offene TCP Ports, unter anderem HTTP, POP, SMTP und einige andere, sowie 3 Vulnerabilities. Dies war anfänglich verwunderlich, da besagte Dienste auf dem Zielsystem nicht aktiv sind, ja sogar teilweise nicht einmal installiert sind.
Bei gleichzeitiger Analyse des Datenstroms am Zielsystem, wurde jedoch ersichtlich, dass Verbindungen auf allen besagten Ports abgelehnt wurden.

```
192.168.0.1 -> 192.168.0.10  TCP 4743 > 80 [SYN] Seq=359698581 Ack=0 Win=65535 Len=0
192.168.0.10 -> 192.168.0.1  TCP 80 > 4743 [RST, ACK] Seq=0 Ack=359698582 Win=0 Len=0
```

Keiner der als offen angezeigten Ports akzeptierte Verbindungen. Es kann sich bei der Diagnose von LANguard also nur um einen Fehler handeln. Bezüglich der drei Vulnerabilities, muss gesagt werden, dass diese vom Programm offensichtlich nur angenommen wurden, ohne sie tatsächlich zu überprüfen. So wurde zum Beispiel ein Problem mit dem POP3 Server gemeldet, der auf dem gescannten System jedoch gar nicht existent ist.

Wie sich später nach Überprüfung mittels Nmap herausstellte, lag das Problem am TCP Connect Scan, der bei beiden Programmen zu falschen Ergebnissen kommt. Ein TCP SYN Scan zeigt bei Nmap richtige Ergebnisse. Leider lässt sich bei der kommerziellen Software kein SYN Scan durchführen.

Passwort Cracking

Zusätzlich zu den durchgeführten Vulnerability Scans, soll auch noch versucht werden, auf unerlaubtem Wege an das Root Passwort zu kommen. Zu diesem Zweck wird ein *Passwort Cracker* eingesetzt.

John the Ripper *John the Ripper* ist eine Passwort Cracking Software, die das Knacken von UNIX Passwörtern ermöglichen soll. Um dies zu erreichen werden sowohl *Brute Force* Attacken als auch *Dictionary* Attacken genutzt (siehe 3.1.1). Als Passwort für das Testsystem wurde eine achtstellige alphanumerische *non-dictionary* Kombination, das bedeutet, ein Passwort, das in keinem Wörterbuch zu finden ist, gewählt, die Groß- und Kleinbuchstaben, sowie Zahlen und Sonderzeichen enthält. Wäre es einem Angreifer möglich, Zugang zur Datei `/etc/shadow` zu erhalten, so könnte er *John the Ripper* nutzen, um an das Root Passwort zu kommen. Um die Erfolgswahrscheinlichkeit einer solchen Attacke zu testen, wurde das Programm auf die Passwortdatei des Systems angewandt. Selbst nach 24 Stunden stellte sich jedoch kein Erfolg ein. Ein Passwort der oben erwähnten Komplexität, ergibt eine Anzahl von 6.634.204.312.890.625 möglichen Kombinationen, was bei einer angenommenen Versuchsgeschwindigkeit von etwa 500.000 Kombinationen pro Sekunde, eine Zeitspanne von 420 Jahren in Anspruch nähme. Dies kann guten Gewissens als sicher angesehen werden.

4.11 Praxistest Fazit

Im Zuge des praktischen Teils der Arbeit, wurden Programme aus den verschiedensten Kategorien der Informationssicherheit implementiert und auf ihre Funktionsweise hin überprüft, um festzustellen, ob diese die an sie gestellten Anforderungen zu erfüllen im Stande sind. Zusätzlich wurde noch das System in seiner Gesamtheit auf mögliche Lücken und Schwachstellen getestet.

Die Ergebnisse dieser Tests zeigten, dass alle eingesetzten Open Source Tools, wenngleich in manchen Kategorien auch die Auswahl gering war, den Erwartungen gerecht wurden, und die an sie gestellten Aufgaben meisterten. Zusätzlich zeigte eine *Sicherheitsüberprüfung* des gesamten Systems, dass es keine leicht auszunützenden Lücken gibt, und es nur mit großem Aufwand möglich wäre, in das System einzudringen. Wird jedoch davon ausgegangen, dass ein großer Teil der potentiellen Eindringlinge so genannte *Script Kiddies* sind, das sind Angreifer, die nicht über die Fähigkeiten verfügen, um selbstständig in ein System einzudringen, und aus diesem Grund auf vorgefertigte Tools zurückgreifen, und nur ein sehr kleiner Teil aus *echten Hacker* besteht, so kann das vorliegende System durchaus als sicher angesehen werden.

Zu bemerken bleibt, dass die gewonnenen Ergebnisse selbstverständliche nur für die vorliegende Konfiguration gültig ist, und nicht garantiert werden kann, dass die Sicherheit gewährleistet ist, wenn zusätzlich noch andere Dienste eingesetzt werden, die neue Sicherheitslücken mit sich bringen könnten.
Trotzdem bietet das System eine solide Grundlage, einen sicheren Zustand zu erreichen, da zwar neue Dienste auch neue Bedrohungen mit sich bringen würden, jedoch Systeme wie die Firewall oder Intrusion Detection Systeme an diese neue Situation angepasst werden können, und auf diese Weise für Schwächen anderer Programme kompensieren können.

Abschließend bleibt festzuhalten, dass mit dem entsprechenden Aufwand und der nötigen Expertise, das Eindringen in wahrscheinlich jedes System möglich ist, und sei es noch so sicher. Da der Großteil der potentiellen Eindringlinge ihre Angriffe jedoch auf vorgefertigte Standardtools stützt, wird ein System wie das vorliegende in der Lage sein, den meisten Angriffen Stand zu halten.

5 Fazit

Kommerzielle Sicherheitsprodukte bedeuten für Unternehmen oft sehr hohe Anschaffungskosten. Zwar gibt es alternativ dazu kostengünstige Open Source Produkte, jedoch ist oft nicht klar, ob diese Produkte auch entsprechende Sicherheit bieten können. Diese Tatsache führte zur Entwicklung der forschungsleitenden Fragestellung:

Sind reine Open Source Security Lösungen von ihrem Leistungsumfang geeignet, kommerzielle Produkte vollständig zu ersetzen, und somit eine kostengünstige Alternative zu bieten?

Zusätzlich war noch von Interesse, ob der Aufwand, der betrieben werden muss, um eine reine Open Source Lösung zu implementieren, den von kommerziellen Lösungen übersteigt. Dieses Kapitel soll sich nun der Beantwortung dieser Fragen annehmen. Zu diesem Zweck werden zuerst die wichtigsten Aspekte der Arbeit zusammengefasst, um anschließend basierend auf den gewonnenen Erkenntnissen, zu einer Schlussfolgerung zu finden, die der Beantwortung der Forschungsfrage dienen wird.

Die Arbeit gliedert sich prinzipiell in drei Teile. Zu Beginn war es nötig, die grundlegenden Aspekte von Open Source und Security zu erforschen, um Rahmen zu definieren, in denen sich die Arbeit bewegt. Dazu wurden verschieden Aspekte von Open Source beleuchtet, sowie die theoretischen Hintergründe der Informationssicherheit analysiert.
Nachdem die theoretischen Grundlagen der Informationssicherheit erarbeitet wurden, befasste sich der zweite Teil mit einer Analyse verfügbarer Open Source Security Tools aus allen relevanten Bereichen der IT Sicherheit, mit dem Ziel, festzustellen, welche Produkte bereits zur Verfügung standen, welche positiven Aspekte zu erkennen waren, und wo Defizite bestanden. Zusätzlich diente die Analyse dem Vergleich der einzelnen Vertreter einer Kategorie, mit dem Ziel, jeweils ein Programm für den praktischen Einsatz, den dritten Teil der Arbeit, auszuwählen. Die Analyse zeigte, dass sich die einzelnen Bereiche stark unterscheiden. So gibt es beispielsweise eine große Zahl an Open Source Firewall Lösungen, und Open Source Varianten von Intrusion Detection Systemen sind ihren kommerziellen Gegenspielern heutzutage in punkto Verbreitung oft überlegen. Im Gegensatz dazu, ist die Zahl der Open Source Port Scanner schon wesentlich geringer, und bei Anti-Viren Programmen war es auch nach längerer Recherche nicht möglich, mehr als nur ein vollwertiges Produkt zu finden.

Der dritte, und zur Beantwortung der Forschungsfrage bedeutendste Teil, befasste sich mit der praktischen Implementierung der in den beiden vorhergehenden Abschnitten theoretisch erarbeiteten Inhalte. Die Idee hinter dieser Arbeit war, ein IT System zu erstellen, das ausschließlich auf Open Source Software basiert, und dennoch zumindest die gleichen Sicherheitsstandards bietet, wie eine kommerzielle Lösung.

Zur Umsetzung dieser Idee, war es zuvor notwendig, jeweils eines der analysierten Produkte jeder Kategorie, das am besten geeignet schien, auszuwählen. Anschließend wurden die Programme auf einem System, das auf einem Open Source Betriebsystem basierte, implementiert, und entsprechend den Erkenntnissen, die aus den vorhergehenden Abschnitten gewonnen wurden, konfiguriert.
Um feststellen zu können, ob das System den Anforderungen, die daran gestellt wurden, gerecht werden kann, wurden die einzelnen Komponenten, sowie das System in seiner Gesamtheit, auf Funktion und Schwachstellen getestet.

- **Firewall** - Als Firewall wurde iptables eingesetzt. Um nach der Konfiguration zu ermitteln, ob die entsprechenden Regeln auch erwartungsgemäß funktionieren, wurden Port Scans durchgeführt.
- **Netzwerk IDS** - Bei der Entscheidung über ein Netzwerk basiertes Intrusion Detection System fiel die Wahl auf Snort, da es das am häufigsten eingesetzte System dieser Art ist und ein umfangreiches Repertoire an Funktionen bietet. Um die Leistungsfähigkeit der Software zu testen, wurde das Auskundschaften eines Systems mit Hilfe eines Vulnerability Scanners gewählt. Dies sollte einen Hacker simulieren, der vor einem Einbruch versucht, Information über ein System zu beschaffen.
- **Host IDS** - Zur Überprüfung der Tauglichkeit des Host basierenden IDS, im konkreten Fall Tripwire, wurde eine vom Programm überwachte Datei vorsätzlich verändert, um einen Eindringling zu simulieren, der sich an wichtigen Systemdateien zu schaffen macht.
- **Anti-Viren Software** - ClamAV wurde als Anti-Viren Programm eingesetzt. Getestet wurde es, indem Testsignaturen im System platziert wurden, und mit Hilfe der Software aufgespürt wurden.
- **Verschlüsselung** - Mit Hilfe von OpenSSL wurden verschiedenste Verschlüsselungskonzepte sowie Hashalgorithmen und Zertifikate getestet, um festzustellen, ob der Funktionsumfang des Pakets, den Anforderungen einer realen Anwendung gerecht wird.

Zusätzlich zu den Tests der einzelnen Komponenten, wurde am Ende des Abschnitts auch das gesamte System auf mögliche Lücken untersucht. Dies geschah und Zuhilfenahme eines Vulnerability Scanner, Nessus. Um zu erreichen, dass wirklich alle Facetten des Systems untersucht werden, wurden mehrere Überprüfungen durchgeführt. Die ersten beiden Scans erfolgten von einem externen Host, um aus der Perspektive eines Hackers zu scannen. Der dritte wurde lokal am System durchgeführt, um etwaige Schwächen zu finden, die ein Eindringling ausnützen könnte, der bereits Zugang zum System hat. Beim ersten Test war die Firewall aktiv, bei den beiden letzten wurde sie deaktiviert.
Zusätzlich zum Scan mit Nessus kamen noch weitere Open Source, aber auch kommerzielle Sicherheitsüberprüfungstools zum Einsatz. Die dabei gewonnenen Ergebnisse deckten sich nahezu vollständig mit denen, die mit Nessus gefunden wurden.

Alle durchgeführten Tests hatten zum Ergebnis, dass das System als äußerst sicher gegenüber Angriffen mit üblichen Werkzeugen ist. Die eingesetzten Mittel waren nicht in der Lage, ernsthafte Lücken im System aufzuspüren. Ein Eindringen in ein System, wie dem untersuchten, wäre also nur unter großem Aufwand und den entsprechenden Fertigkeiten realisierbar. Zwar könnte der Einsatz zusätzlicher Programme, die in der Teststellung nicht berücksichtigt wurden, neue Schwächen mit sich bringen, jedoch wäre das vorliegende System flexibel genug, um sich den geänderten Umständen anzupassen.

Die im Zuge der Arbeit gewonnenen Erkenntnisse, führen schlussendlich zu folgender Beantwortung der Forschungsfrage: Reine Open Source Security Lösungen, das bedeutet Programme, die vollständig den Prinzipien freier Software unterliegen, sind geeignet, um IT Systeme vollwertig abzusichern, und bieten eine kostengünstige Alternative zu kommerziellen Produkten. Zusätzlich ist anzumerken, dass der Installations- und Konfigurationsaufwand nur geringfügig höher ist, als bei kommerziellen Produkten. Die gängige Meinung, man müsse ein IT Spezialist sein, um mit Linux und Open Source Produkten umgehen zu können, hat sich nicht bestätigt. Die meisten Programme sind als selbstinstallierende Pakete verfügbar, die weder konfiguriert, noch manuell kompiliert werden müssen. Der geringe Mehraufwand ist im Verhältnis zur Kostenminderung also durchaus akzeptabel.

Weiters ist zu erwähnen, dass Open Source Produkte im Server Bereich ohnehin schon lange Einzug halten, und immer populärer werden. So sind gerade im Bereich der IT Security aber auch in anderen Bereichen, Open Source Produkte durch ihre Verbreitung oft schon als *Quasi-Standards* anzusehen.

Zusätzliche Aspekte, die vor allem für Unternehmen von Interesse sind, sind die Plattformunabhängigkeit sowie die Wartung der Produkte. Zwar sind die meisten Open Source Produkte gut dokumentiert, jedoch gibt es nicht für alle Wartungsverträge vom Hersteller. Diese können bei größeren Projekten jedoch meist von Dritten kostenpflichtig erworben werden.
Bezüglich Plattformunabhängigkeit sind die meisten in der Arbeit behandelten Produkte eher unflexibel. Zwar gibt es von manchen Produkten, wie zum Beispiel Nessus, Versionen für andere, kommerzielle Betriebsysteme (z.B. Microsoft Windows), jedoch sind diese dann meist auch selbst kommerziell. Nachdem sich die Arbeit jedoch ausschließlich mit Produkten für Open Source Betriebsysteme beschäftigte, ist die Plattformunabhängigkeit der analysierten Produkte nicht repräsentativ.

Abschließend soll gesagt sein, dass auch Open Source Produkte selbstverständlich keine hundertprozentige Absicherung garantieren können. Es wird, bei freier wie bei kommerzieller Software, immer Produkte geben, die Eindringlingen Wege bieten, sich unbefugten Zugang zu einem System zu beschaffen. Werden jedoch die richtigen Methoden eingesetzt, und von entsprechend qualifiziertem Personal gehandhabt, so können Open Source Produkte durchaus eine Alternative bieten.

A | Open Source Lizenzen

A.1 GNU General Public License

Version 2, June 1991

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU General Public License

Terms and Conditions For Copying, Distribution and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The “Program”, below, refers to any such program or work, and a “work based on the Program” means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term “modification”.) Each licensee is addressed as “you”.

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program’s source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when

you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore,

by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.
To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.>
> Copyright (C) <year> <name of author>
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Also add information on how to contact you by electronic and paper mail.
If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) <year> <name of author>
> Gnomovision comes with ABSOLUTELY NO WARRANTY;
> for details type 'show w'.
> This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w` and `show c`; they could even be mouse-clicks or menu items—whatever suits your program.
You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program
> 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> <signature of Ty Coon>, 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

A.2 BSD Lizenz

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the University of California, Berkeley nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

B | iptables Konfiguration

Listing B.1: iptables config script

```
#               DA Sample iptables config script - June 2006
#                                Peter Brachtl
#
#
#
#
#
#
#########################################################################



#########################################################################
#
# 1. Konfiguration der Umgebungsvariablen
#

#
# 1.1 Pfad definieren
#

IPTABLES=‘which iptables‘

#
# 1.2 Internes Netz konfigurieren (Intranet)
#

DEV_INT=eth0
IP_INT=‘ifconfig $DEV_INT |grep inet |cut -d : -f 2 |cut -d \  -f 1‘
MASK_INT=‘ifconfig $DEV_INT |grep Mask |cut -d : -f 4‘
NET_INT=‘ifconfig $DEV_INT |grep inet |cut -d : -f 2 |cut -d \  -f 1 | \
         cut -d . -f 1,2,3‘.0/$MASK_INT
BC_INT=‘ifconfig $DEV_INT |grep Bcast |cut -d : -f 3| cut -d \  -f 1‘

#
# 1.3 DMZ konfigurieren
#

DEV_DMZ=eth2
IP_DMZ=‘ifconfig $DEV_DMZ |grep inet |cut -d : -f 2 |cut -d \  -f 1‘
MASK_DMZ=‘ifconfig $DEV_DMZ |grep Mask |cut -d : -f 4‘
NET_DMZ=‘ifconfig $DEV_DMZ |grep inet |cut -d : -f 2 |cut -d \  -f 1 | \
         cut -d . -f 1,2,3‘.0/$MASK_DMZ
BC_DMZ=‘ifconfig $DEV_DMZ |grep Bcast |cut -d : -f 3| cut -d \  -f 1‘


#
# 1.4 Externes Netz konfigurieren (Internet)
#

DEV_EXT=eth1
```

```
IP_EXT=‘ifconfig $DEV_EXT |grep inet |cut -d : -f 2 |cut -d \  -f 1‘
MASK_EXT=‘ifconfig $DEV_EXT |grep Mask |cut -d : -f 4‘
NET_EXT=‘ifconfig $DEV_EXT |grep inet |cut -d : -f 2 |cut -d \  -f 1 | \
          cut -d . -f 1,2,3‘.0/$MASK_EXT
BC_EXT=‘ifconfig $DEV_EXT |grep Bcast |cut -d : -f 3| cut -d \  -f 1‘

#
# 1.5 Localhost konfigurieren
#

DEV_LO=lo
IP_LO="127.0.0.1"

#
# 1.6 Hosts definieren
#

# 1.6.1 Interne Hosts

IT="10.0.0.1"
MARKETING="10.0.0.2"
HR="10.0.0.3"

# 1.6.2 Externe Hosts

FILIALE="192.168.0.2"

# 1.6.3 Server

#WEBSERVER="172.16.1.1"
#FTPSERVER="172.16.1.2"
#MAILSERVER="172.16.1.3"

############################################################################
#
# 2. Module laden
#

modprobe ip_tables
modprobe ip_conntrack
modprobe iptable_filter
modprobe iptable_mangle
modprobe iptable_nat
modprobe ipt_ttl
modprobe ipt_LOG
modprobe ipt_limit
modprobe ipt_state
modprobe ip_conntrack_ftp
modprobe ip_conntrack_irc
modprobe ip_nat_irc
modprobe ip_nat_ftp



############################################################################
#
# 3. /proc Setup
#

echo "1" > /proc/sys/net/ipv4/ip_forward
echo "1" > /proc/sys/net/ipv4/conf/all/rp_filter
echo "1" > /proc/sys/net/ipv4/conf/all/proxy_arp
echo "1" > /proc/sys/net/ipv4/ip_dynaddr



############################################################################
#
# 4. Rules konfigurieren
#

#
```

```
# 4.1 Chains leeren
#

$IPTABLES -F
$IPTABLES -t nat -F
$IPTABLES -t mangle -F
$IPTABLES -X


##################
# 4.2 Filter Table
#

#
# 4.2.1 Default policies setzen
#

$IPTABLES -P INPUT DROP
$IPTABLES -P FORWARD DROP
$IPTABLES -P OUTPUT DROP

#
# 4.2.2 Eigene Chains erstellen
#

$IPTABLES -N allowed
$IPTABLES -N tcp_packets
$IPTABLES -N udp_packets
$IPTABLES -N icmp_packets
$IPTABLES -N bad_tcp_packets

#
# 4.2.3 Rules fuer eigene Chains erstellen
#

# 4.2.3.1 bad_tcp_packets Rules
$IPTABLES -A bad_tcp_packets -p tcp --tcp-flags SYN,ACK SYN,ACK -m state \
--state NEW -j REJECT --reject-with tcp-reset
$IPTABLES -A bad_tcp_packets -p tcp ! --syn -m state --state NEW -j LOG \
--log-prefix "New␣not␣syn:"
$IPTABLES -A bad_tcp_packets -p tcp ! --syn -m state --state NEW -j DROP

# 4.2.3.2 allowed Rules
$IPTABLES -A allowed -p tcp --syn -j ACCEPT
$IPTABLES -A allowed -p tcp -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A allowed -p tcp -j REJECT --reject-with tcp-reset

# 4.2.3.3 TCP Rules

# sshd (Filiale)
$IPTABLES -A tcp_packets -i $DEV_EXT -p tcp -s $FILIALE --dport 22 -j allowed

# sshd (IT)
$IPTABLES -A tcp_packets -i $DEV_INT -p tcp -s $IT --dport 22 -j allowed

# Squid (Filiale)
$IPTABLES -A tcp_packets -i $DEV_EXT -p tcp -s $FILIALE --dport 3128 -j allowed

# 4.2.3.4 UDP Rules



# 4.2.3.5 ICMP rules

$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 8 -j ACCEPT
$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 11 -j ACCEPT


#
# 4.2.4 INPUT Chain
#
```

```
# Block "bad packets"
$IPTABLES -A INPUT -p tcp -j bad_tcp_packets

# Spezielle rules fuer localhost
$IPTABLES -A INPUT -i $DEV_LO -p ALL -s $IP_LO -j ACCEPT
$IPTABLES -A INPUT -i $DEV_LO -p ALL -s $IP_INT -j ACCEPT
$IPTABLES -A INPUT -i $DEV_LO -p ALL -s $IP_EXT -j ACCEPT

# Rules fuer Pakete von den Interfaces
$IPTABLES -A INPUT -p ALL -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A INPUT -p TCP -j tcp_packets
$IPTABLES -A INPUT -p UDP -j udp_packets
$IPTABLES -A INPUT -p ICMP -j icmp_packets

# DHCP Requests am externen Interface blocken (wird nicht geloggt)
$IPTABLES -A INPUT -p UDP -i $DEV_EXT -d 255.255.255.255 \
--destination-port 67:68 -j DROP

# Alles LOGgen auf das keine der oberen rules zutrifft
$IPTABLES -A INPUT -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT␣INPUT␣packet␣died:␣"

# TCP Port closed
$IPTABLES -A INPUT -i $DEV_EXT -p TCP -j REJECT --reject-with tcp-reset

# UDP Port closed
$IPTABLES -A INPUT -i $DEV_EXT -p UDP -j REJECT --reject-with icmp-port-unreachable


#
# 4.2.5 FORWARD chain
#

# Block "bad packets"
$IPTABLES -A FORWARD -p tcp -j bad_tcp_packets

# Alle Pakete vom LAN forwarden
$IPTABLES -A FORWARD -i $DEV_INT -p ALL -s $NET_INT -j ACCEPT


# 4.2.5.1 Virtual Server

# VNC forward zu  MARKETING (fuer Filiale)
$IPTABLES -A FORWARD -i $DEV_EXT -o $DEV_INT -p tcp -s $FILIALE -d $MARKETING \
--dport 5950 -j ACCEPT
$IPTABLES -A FORWARD -i $DEV_EXT -o $DEV_INT -p tcp -s $FILIALE -d $MARKETING \
--dport 5850 -j ACCEPT

# HTTP forward zu Webserver (DMZ)
$IPTABLES -A FORWARD -i $DEV_EXT -o $DEV_DMZ -d $WEBSERVER -p tcp --dport 80 \
-j ACCEPT

# POP forward zu Mailserver (DMZ)
$IPTABLES -A FORWARD -i $DEV_EXT -o $DEV_DMZ -d $WEBSERVER -p tcp --dport 110 \
-j ACCEPT

# SMTP forward zu Mailserver (DMZ)
$IPTABLES -A FORWARD -i $DEV_EXT -o $DEV_DMZ -d $WEBSERVER -p tcp --dport 25 \
-j ACCEPT


# 4.2.5.2 Host Relaying

# Host to LAN forwarding
#$IPTABLES -A FORWARD -i $DEV_EXT -o $DEV_INT -s x.x.x.x -j ACCEPT

# Host to Host forwarding
#$IPTABLES -A FORWARD -i $DEV_EXT -o $DEV_INT -s x.x.x.x -d x.x.x.x -j ACCEPT

# Bounce Host
#$IPTABLES -A FORWARD -i $DEV_EXT -o $DEV_EXT -s x.x.x.x -j ACCEPT
```

```
#Alle Established/Related Verbindungen forwarden
$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

# Alles LOGgen auf das keine der oberen rules zutrifft
$IPTABLES -A FORWARD -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT␣FORWARD␣packet␣died:␣"


#
# 4.2.6 OUTPUT chain
#

# Block "bad packets"
$IPTABLES -A OUTPUT -p tcp -j bad_tcp_packets

# OUTPUT rules (Alles erlaubt)
$IPTABLES -A OUTPUT -p ALL -s $IP_LO -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -s $IP_INT -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -s $IP_DMZ -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -s $IP_EXT -j ACCEPT

# Alles LOGgen auf das keine der oberen rules zutrifft
$IPTABLES -A OUTPUT -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT␣OUTPUT␣packet␣died:␣"


###################
# 4.3 NAT Table
#

#
# 4.3.1 PREROUTING Chain
#

# 4.3.1.1 Virtual Server

# VNC forward zu Marketing (fuer Filiale)
$IPTABLES -t nat -A PREROUTING -i $DEV_EXT -p tcp -s $FILIALE -d $IP_EXT --dport 5950 \
-j DNAT --to-destination $MARKETING
$IPTABLES -t nat -A PREROUTING -i $DEV_EXT -p tcp -s $FILIALE -d $IP_EXT --dport 5850 \
-j DNAT --to-destination $MARKETING

# HTTP forward zu Webserver
$IPTABLES -t nat -A PREROUTING -i $DEV_EXT -p tcp --dport 80 -j DNAT \
--to-destination $WEBSERVER

# POP forward zu Mailserver
$IPTABLES -t nat -A PREROUTING -i $DEV_EXT -p tcp --dport 110 -j DNAT \
--to-destination $MAILSERVER

# SMTP forward zu Mailserver
$IPTABLES -t nat -A PREROUTING -i $DEV_EXT -p tcp --dport 25 -j DNAT \
--to-destination $MAILSERVER

# Transparent Web Proxy Server
$IPTABLES -t nat -A PREROUTING -i $DEV_INT -p tcp --dport 80 -j REDIRECT \
--to-ports 3128


#
# 4.3.2 POSTROUTING Chain
#

# 4.3.2.1 NAT fuer LAN Internet Access
$IPTABLES -t nat -A POSTROUTING -o $DEV_EXT -j SNAT --to-source $IP_EXT

# 4.3.2.2 Virtual Server

# VNC forward zu Marketing (fuer Filiale)
$IPTABLES -t nat -A POSTROUTING -p tcp -s $FILIALE -d $MARKETING --dport 5950 -j SNAT \
--to-source $IP_INT
```

```
$IPTABLES -t nat -A POSTROUTING -p tcp -s $FILIALE -d $MARKETING --dport 5850 -j SNAT \
--to-source $IP_INT

# HTTP forward zu Webserver
$IPTABLES -t nat -A POSTROUTING -p tcp -d $WEBSERVER --dport 80 -j SNAT \
--to-source $IP_INT

# POP forward zu Webserver
$IPTABLES -t nat -A POSTROUTING -p tcp -d $MAILSERVER --dport 110 -j SNAT \
--to-source $IP_INT

# SMTP forward zu Mailserver
$IPTABLES -t nat -A POSTROUTING -p tcp -d $MAILSERVER --dport 25 -j SNAT \
--to-source $IP_INT


######################
# 4.4  Mangle Table
#

# TTL erhöhen um LAN zu verstecken
$IPTABLES -t mangle -A PREROUTING -i $DEV_INT -j TTL --ttl-inc 1
```

Literaturverzeichnis

Bücher

[Barrett et al. 2003]
D. J. Barrett, R. G. Byrnes, R. Silverman. *Linux Security Cookbook*. O'Reilly, USA, Juni 2003. ISBN 0-596-00391-9.

[Bauer 2002]
M. D. Bauer. *Building Secure Servers with Linux*. O'Reilly, USA, Oktober 2002. ISBN 0-596-00217-3.

[Bishop 2003]
M. Bishop. *Computer Security*. Addison-Wesley, USA, November 2003. ISBN 0-201-44099-7.

[Chandra et al. 2002]
P. Chandra, M. Messier, J. Viega. *Network Security with OpenSSL*. O'Reilly, USA, Juni 2002. ISBN 0-596-00270-X.

[Cox, Gerg 2004]
K. J. Cox, C. Gerg. *Managing Security with Snort and IDS Tools*. O'Reilly, USA, August 2004. ISBN 0-596-00661-6.

[Flickenger 2003]
R. Flickenger. *Linux Server Hacks*. O'Reilly, USA, Jänner 2003. ISBN 1-565-92871-7.

[ISO 2005]
ISO. *ISO/IEC 17799:2005*. Juni 2005.

[Howlett 2004]
T. Howlett. *Open Source Security Tools*. Prentice Hall, USA, July 2004. ISBN 0-321-19443-8.

[McNab 2004]
C. McNab. *Network Security Assessment*. O'Reilly, USA, März 2004. ISBN 0-596-00611-X.

[Odom 2003]
W. Odom. *CCNA INTRO*. Cisco Press, USA, November 2003. ISBN 1-587-20094-5.

[Tanenbaum 2003]
A. S. Tanenbaum. *Computer Networks*. Prentice Hall, USA, 2003. ISBN 0-130-38488-7.

[Williams 2002]
S. Williams. *Free as in Freedom*. O'Reilly, USA, März 2002. ISBN 0-596-00287-4.

Internetquellen

[Apache Website 2005]
Apache Website. *Apache HTTP Server Project Website*. November 2005. URL `http://httpd.apache.org/`. Stand: 29.05.2006.

[Free Software Foundation 1991]
Free Software Foundation. *GNU General Public License Version 2*. 1991. URL `http://www.fsf.org/licensing/licenses/gpl.html`. Stand: 29.05.2006.

[Fyodor]
Fyodor. *Who is Fyodor?* URL `http://insecure.org/myworld.html`. Stand: 20.08.2006.

[Gates 2002]
B. Gates. *Trustworthy Computing Memo*. Jänner 2002.

[GNU GPG Website 2006]
GNU GPG Website. *GNU GPG Website*. 2006. URL `http://www.gnupg.org/`. Stand: 20.08.2006.

[ISO 2000]
ISO. *ISO/IEC 7498-1:1994*. 2000. URL `http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=20269&ICS1=35&ICS2=100&ICS3=1`. Stand: 23.05.2006.

[Link 2006]
R. Link. *OpenAntiVirus Website*. 2006. URL `http://www.openantivirus.org`. Stand: 20.08.2006.

[Netfilter Website 2005]
Netfilter Website. *Project History*. 2005. URL `http://www.netfilter.org/about.html#history`. Stand: 30.06.2006.

[OAV Crew 2006]
OAV Crew. *OpenAntiVirus Website*. 2006. URL `http://www.openantivirus.org/status.php`. Stand: 20.08.2006.

[OpenBSD Website 2006]
OpenBSD Website. *OpenBSD*. Mai 2006. URL `http://www.openbsd.org`. Stand: 09.06.2006.

[OpenSSL Website 2006]
OpenSSL Website. *OpenSSL Website*. 2006. URL `http://www.openssl.org/support/faq.html#LEGAL2`. Stand: 20.08.2006.

[Perens 1997]
B. Perens. *The Open Source Definition*. 1997. URL `http://opensource.org/docs/definition\_plain.php`. Stand: 27.05.2006.

[Postel 1981]
J. Postel. *RFC792*. September 1981.

[Sourcefire 2006]
Sourcefire. *About Snort*. 2006. URL `http://snort.org/about_snort/`. Stand: 24.08.2006.

[Unicornscan]
Unicornscan. *Unicornscan FAQ*. URL `http://www.unicornscan.org/text/unicornscan_faq.txt`. Stand: 20.08.2006.

[Wikipedia 2006]
Wikipedia. *GNU*. Mai 2006. URL `http://de.wikipedia.org/wiki/GNU`. Stand: 29.05.2006.

[Wikipedia.org 2006a]
Wikipedia.org. *BSD License*. Mai 2006a. URL `http://en.wikipedia.org/wiki/BSD_License`. Stand: 29.05.2006.

[Wikipedia.org 2006b]
Wikipedia.org. *GNU General Public License*. Mai 2006b. URL `http://en.wikipedia.org/wiki/gpl`. Stand: 29.05.2006.

Handbücher

Bastille Linux-Manpage. *Bastille Linux Manpage*, 2005.

H. Eychenne. *iptables Manpage*, März 2002.

A. Frigido. *Manuale Turtle Firewall*, 2002.

Fyodor. *Nmap Reference Guide*, 2005.

T. Kojm. *Clam AntiVirus 0.88.3 User Manual*, 2006.

PF Manual. *PF: The OpenBSD Packet Filter*, 2003.

A. Ramos. *raccess Readme*, 2002.

SmoothWall Limited. *Smoothwall Administration Guide*, 2003.

Sourcefire. *Snort Users Manual 2.4.0*, 2005.

Tripwire Inc. *Tripwire Manpage*, 2003.

Abbildungsverzeichnis

Abkürzungsverzeichnis

3DES	Triple DES
AES	Advanced Encryption Standard
ASCII	American Standard Code for Information Interchange
BSD	Berkley Software Distribution
CA	Certificate Authority
CIA	Confidentiality, Integrity, Availability
CLI	Command Line Interface
CPU	Central Processing Unit
DDos	Distributed Denial of Service
DES	Data Encryption Standard
DHCP	Dynamic Host Configuration Protocol
DMZ	Demilitarized Zone
DNAT	Destination NAT
DoS	Denial of Service
DSL	Digital Subscriber Line
FSF	Free Software Foundation
FTP	File Transfer Prototcol
GNU	GNU's Not Unix
GPG	GNUA Privacy Guard
GPL	General Public License
HIDS	Host Intrusion Detection System
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
ICMP	Internet Control Message Protocol
IDEA	International Data Encryption Algorithm
IDS	Intrusion Detection System
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IOS	Internetwork Operating System
IP	Internet Protocol
IPC	Inter Process communication
IPSec	IP Security
IPX	Internetwork Packet Exchange
IRC	Internet Relay Chat
ISO	International Organization for Standardization
ISP	Internet Servie Provider
IT	Information Technology
JPG	Joint Photographic Experts Group
LAN	Local Area Network

MD5	Message Digest 5
MIT	Massachusetts Institute of Technology
MTA	Mail Transport Agent
NASL	Nessus Attack Scripting Language
NAT	Network Address Translation
NFS	Network File System
NIDS	Network Intrusion Detection System
NMS	Network Monitoring System
ODBC	Open Database Connectivity
OS	Operating System
OSI	Open Systems Interconnect
PAT	Port Adress Translation
PGP	Pretty Good Privacy
PHP	PHP: Hypertext Preprocessor
POP	Post Office Protocol
PPP	Point-to-Point Protocol
QoS	Quality of Service
RFC	Request for Comment
RPM	Red Hat Package Manager
RSA	Rivest Shamir Adleman
SGID	Set Group ID
SHA	Secure Hash Algorithm
SMB	Server Message Block
SMTP	Simple Mail Transfer Protocol
SNAT	Source NAT
SQL	Structured Query Language
SSH	Secure Shell
SSL	Secure Socket Layer
SUID	Set User ID
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TOS	Type Of Sevice
TTL	Time To Live
UDP	User Datagram Protocol
VNC	Virtual Network Computing
VPN	Virtual Private Network

Stichwortverzeichnis