Implementation of Snort on a Network Processor

Diplomarbeit

Für den Fachbereich

Elektrotechnik

Im Studiengang

Kommunikationstechnik

An der

Hochschule für Technik und Wirtschaft Dresden (FH)

Betreuer: Prof. Dr.-Ing Sven Zeisberg

Geschrieben an der

Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Betreuer: Prof.Dr.-Ing. Stephan Robert

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Zum Erlangen des ersten akademischen Grades

Diplomingenieur (FH)

Vorgelegt von

Christoph Lenk

Inhaltsverzeichnis

Abkürzungsverzeichnis

ALU: Arithmetic Logic Unit

ARM: Advanced RISC Machines

ARP: Address Resolution Protocol

ASCII: American Standard Code for Information Interchange

ATX: Advanced Technology Extended

BSP: Board Support Package

CGI: Common Gateway Interface

CIDR: Classless Inter-Domain Routing

CPU: Central Processing Unit

CRC: Cyclic Redundancy Check

DDR: Double Data Rate

DSA: Digital Signature Algorithm

GBit: Gigabit

GHz: Gigahertz

GPP: General Purpose Processor

HTTP: Hyper Text Transfer Protocol

IDS: Intrusion Detection System

I/O: Input/Output

IP: Internet Protocol

IXP: Intel Exchange Processor

MB: Megabyte

MBit: Megabit

MHz: Megahertz

NFS: Network File System

NPU: Network Processor Unit

NTP: Network Time Protocol

PC: Personal Computer

PCI: Peripheral Component Interconnect

PHP: Akronym von: Hypertext Preprocessor

POSIX: Portable Operating System Interface

RISC: Reduced Instruction Set Computing

ROM: Read Only Memory

RPC: Remote Procedure Call

Rx: Receiver

SDRAM: Synchronous Dynamic Random Access Memory

SSL: Secure Sockets Layer

TCP: Transmission Control Protocol

TFTP: Trivial File Transfer Protocol

Tx: Transmitter

UART: Universal Asynchronous Receiver Transmitter

UDP: User Datagram Protocol

USB: Universal Serial Bus

Abbildungsverzeichnis

0 Einleitung

Intrusion Detection Systems haben sich in den letzten Jahren zu einer wichtigen und kritischen Komponente zur Sicherung von Netzwerken entwickelt. Im Jahr 2007 wurden in Deutschland 4829 Delikte erfasst, bei denen es sich um unerlaubtes Ausspähen von Daten handelt (Quelle: Polizeiliche Kriminalstatistik 2007 BKA). Gerade mittlere und größere Firmen mit ausgeprägten und somit auch unübersichtlichen Netzwerken müssen viel Wert auf die Sicherung der sich in den Netzwerken befindlichen Daten legen. Ohne diverse Sicherheitsmaßnahmen könnten Eindringlinge problemlos sensible und wirtschaftlich wichtige Daten von den Servern der Firmen stehlen. Das Spektrum der gebotenen Sicherheitsmaßnahmen geht dabei von Firewalls über Proxyserver bis hin zu den Intrusion Detection Systems. Da bei sehr großen Netzwerken ein extrem hohes Datenaufkommen herrscht, ist es wichtig, dass die Sicherheitsmaßnahmen auf schnellen und rechenstarken Systemen laufen. Für diesen Fall wurden Netzwerkprozessoren entwickelt. Sie sollen spezifische Aufgaben im Bereich der Netzwerktechnik wesentlich schneller und effizienter erfüllen, als herkömmliche Desktop-PCs. Ziel dieser Diplomarbeit soll es sein, die Vorteile von Netzwerkprozessoren zu ermitteln. Dies geschieht anhand des Intel IXP 2350 als Vertreter der Netzwerkprozessoren und dem Intrusion Detection System Snort.

Übersicht über den Ablauf der Diplomarbeit

Die Kapitel eins und zwei dienen der theoretischen Betrachtung der wichtigsten, während der Arbeit verwendeten Software und Hardware. Da Snort selbst auch als Software gilt, befindet sich eine genauere Erläuterung zu dessen Funktion und Arbeitsweise in Kapital zwei.

Die vorgenommenen Schritte zur Installation und Implementierung von Snort auf den verschiedenen Systemen werden in den Kapiteln drei und vier geschildert.

Im Kapitel fünf wird Snort auf den verschiedenen Systemen einem vergleichenden Performancetest unterzogen. Verfahrensweise und Ergebnisse sind ausführlich in diesem Kapitel beschrieben.

Um Verständnis über die Arbeitsweise und Funktion des Netzwerkprozessors sowie dessen Simulation zu erhalten, enthalten die Kapitel sechs und sieben eine nähere Beschreibung zum Netzwerkprozessor und der Entwicklungsumgebung von Intel. Außerdem befinden sich in diesen Kapiteln alle gewonnenen Erkenntnisse über die spezielle Hardware und Software.

Das Kapitel acht fasst die Diplomarbeit zusammen. Es gibt einen Überblick über Probleme, Ergebnisse und den Ausblick für die Zukunft.

1 Hardwaretechnische Grundlagen

1.1 Netzwerkprozessoren

1.1.1 Allgemein

Netzwerkprozessoren sind integrierte Schaltkreise, deren Hauptfunktion auf die Paketverarbeitung von Netzwerken spezialisiert wurde. Sie werden in der Regel softwarebasiert programmiert und haben allgemein eine Charakteristik ähnlich den Central Processing Units (CPUs).

Die Spezialisierung der Netzwerkprozessoren auf die Pakete im Netzwerkverkehr ermöglicht es, die Paketverarbeitung zu optimieren und zu erweitern. Aus diesem Grund finden Netzwerkprozessoren ihre Anwendung in Routern bzw. Switches und Firewalls.

Einige der typischen Funktionen, die in einen Netzwerkprozessor integriert werden, sind:

- Möglichkeit der Mustererkennung in Bitfolgen
- Übernahme von Routingaufgaben
- Manipulation von Bitfolgen
- Queue Management, also das Speichern von Paketen in Warteschlangen
- Berechnung von Prüfsummen und Streuwerten

Um mit hohen Datenraten besser umzugehen, haben sich einige Standards in der Architektur durchgesetzt:

- Pipelining, also die Zerlegung der Aufgaben in mehrere Teilaufgaben, die dann parallel ausgeführt werden
- Multithreading, also die parallele Verarbeitung mehrerer Programme
- Programmierbare Mikroprozessoren zur Spezialisierung auf einzelne Aufgaben

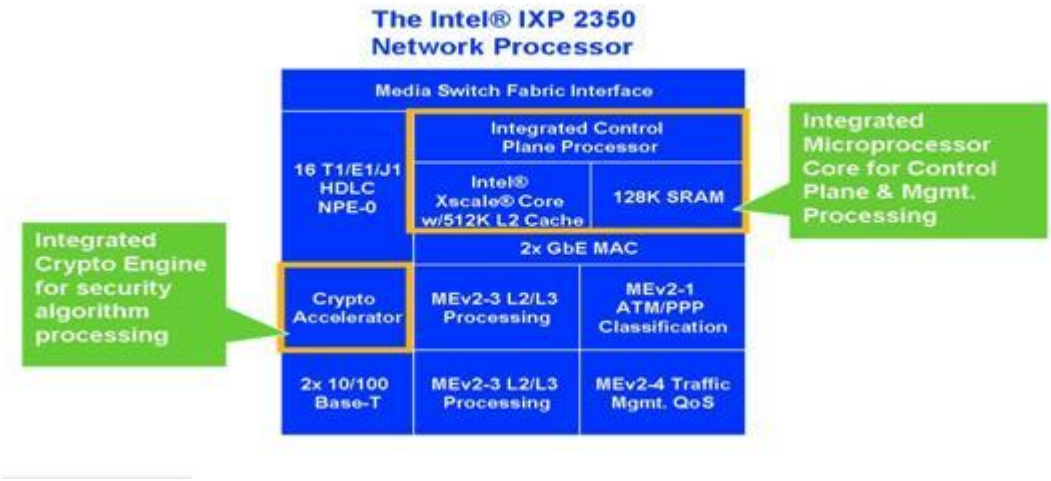

Abbildung 1.1: Aufbau der Prozessorarchitektur des IXP2350 (Quelle: Intel Corp., 2005)

In Abbildung 1.1 ist die Prozessorarchitektur vereinfacht dargestellt. Zu erkennen ist der Intel XScale Core im Control Layer und die Microengines, in der Abbildung MEv2-x genannt, die sich im Data Layer befinden. Der XScale ist ein RISC Prozessor ist und die Microengines dienen zur Spezialisierung einzelner Aufgaben, beispielsweise der Paketverarbeitung.

1.1.2 Intel IXP 2350

Der Intel IXP 2350 stammt aus der Produktlinie der IXP23xx Netzwerk Prozessoren. Er wurde für ein weites Spektrum von Zugriffs- und Flankenapplikation geschaffen. Hardware- und Softwarearchitektur bauen auf der Intel IXP2xxx Produktlinie auf. Der Netzwerkprozessor unterstützt bis zu 4,9 Milliarden Operationen pro Sekunde mit vier unabhängigen 32-Bit Mikroprozessoren, die in der Intel-Terminologie “Microengines“ genannt werden. Ausgestattet mit dem verbrauchsarmen Intel XScale Core und einer möglichen Taktfrequenz von 900 MHz ist der Netzwerkprozessor in der Lage, komplexe Software, beispielsweise ein Betriebssystem, auszuführen. Desweiteren werden zusätzlich Features durch Koprozessoren hinzugefügt, wie zum Beispiel Timer und Input/Output Controller. Zusätzlich verfügt der Netzwerkprozessor über verschiedene externe Verbindungen, wie zum Beispiel einen seriellen Anschluss und mehrere Ethernetanschlüsse.

1.2 Roadrunner IXP2350 Referenzplattform

Der Hauptimpetus des von ADI Engineering entwickelten Roadrunner liegt darin, eine Referenz- bzw. Evaluationsplattform für den Intel IXP2350 Netzwerk Prozessor zu kreieren. Das Design zielt auf ein möglichst breites Anwendungsfeld. So steht dem Nutzer eine große Auswahl an Input/Output Optionen, sowie maximale Speicherkapazität zur Verfügung, um das Potential des Netzwerkprozessors voll auszuschöpfen. Als Referenzplattform erlaubt es der Roadrunner, entwickelte Software zu verifizieren und zu quantifizieren. Ist die Verifizierung der geschriebenen Algorithmen und Kodes erfolgreich, kann davon ausgegangen werden, dass sich die Software für den Intel IXP2350

Netzwerkprozessor eignet. Des Weiteren stehen dem Nutzer noch diverse Erweiterungsmöglichkeiten, zum Beispiel PCI-Slots zur Verfügung um spezifische Hardware zu nutzen.

Vom Hersteller angegebene Schlüsselattribute sind unter anderem:

- fokussiert auf Gigabit Konnektivität
- Erweiterungsmöglichkeiten für Media Interface Karten etc.
- optimierte Bytereihenfolge
- optimiert für Testbarkeit und Reproduzierbarkeit
- Emission- und Sicherheitszertifikate
- Linux BSP und Redboot werden unterstützt und zur Verfügung gestellt von ADI Engineering

Ausgewählte technische Daten:

Prozessor	Intel IXP2350 900MHz / 900MHz Intel XScale Core
Speicher	512 MB DDR SDRAM on DIMM 256 MB DDR SDRAM on board 64 MB Flash
Ethernet (I/O)	2 x 10/100/1000 Base-T 1 x 10/100 Base-T
USB	4 USB 2.0 Ports
Stromversorgung	300W ATX power supply

Abbildung 1.2: Roadrunner Referenzplattform

2 Softwaretechnische Grundlagen

2.1 Verwendete Software

2.1.1 Die Linux-Distribution Ubuntu 8.10

Die Bezeichnung Ubuntu stammt aus dem afrikanischen Sprachraum und bedeutet soviel wie „Menschlichkeit gegenüber anderen“. Das Konzept von Ubuntu basiert auf der Zusammenarbeit aller in einer Gemeinschaft, um eine freie bzw. quelloffene Software zu schaffen, die allen nützt.

Die Wahl für die Hostsysteme fiel auf Ubuntu, da es zu den bekanntesten Linux Distributionen gehört. Dadurch ist die Unterstützung der genutzten Software gewährleistet. Des Weiteren gibt es viele Austauschplattformen und Communities, um im Falle eines Problems eine schnelle und einfache Lösung zu finden. In regelmäßigen Abständen werden Sicherheitsupdates veröffentlicht. Aus diesem Grund bleibt Ubuntu auf dem neusten technischen Stand und bietet daher ideale Voraussetzungen für eine Entwicklungsumgebung, die trotzdem einfach zu bedienen ist.

Die geforderten Bibliotheken und Entwicklungs-Kits können einfach über den Synaptic Package Manager, ein Tool zum Download von Linuxkomponenten, in das System integriert werden. Für den speziellen Bedarf, vor allem für andere Systemarchitekturen, können die Bibliotheken auch als Quelltext geladen und kompiliert werden.

Während der Entstehung der Diplomarbeit wurde eine neue Version - Ubuntu 9.04 - veröffentlicht. Da die Version 8.10 zu der Zeit schon in Benutzung war, wurde darauf verzichtet, um Problemen und Versionskonflikten vorzubeugen.

2.1.2 MySQL Community Edition 5.1

MySQL ist eine Open-Source Datenbank, die sich weltweit zur populärsten Software ihrer Art entwickelt hat. Durch gute Leistungsfähigkeit und hohe Benutzerfreundlichkeit und Zuverlässigkeit ausgezeichnet, wird MySQL von privaten Nutzern sowie Unternehmen bevorzugt. Namhafte Unternehmen, die MySQL nutzen sind unter anderen Yahoo!, neckermann.de, Nokia, Siemens und T-Systems.

Für die Nutzung von MySQL in dieser Diplomarbeit sind keine tieferen Kenntnisse in der Datenbankprogrammierung erforderlich. Das Einloggen in die Datenbank und das Speichern der Daten findet komplett automatisiert statt. Des Weiteren werden mit Snort einige Skripte mitgeliefert, welche die automatische Erstellung der Datenbanken ermöglichen. Dadurch werden im Voraus Probleme vermieden, die aus einer fehlerhaften Datenbankarchitektur resultieren können.

2.1.3 Apache HTTP Webserver Version 1.3

Apache HTTP Server ist eine frei erhältliche Software zur Erstellung und Wartung von Webservern. Die Software wurde mit der Programmiersprache C geschrieben und ist für alle Betriebssysteme erhältlich. Die aktuellste Version 2.2.11 bietet auch für Microsoft Windows eine stabile Möglichkeit zur Erstellung der Webserver. Die in der Arbeit genutzte Version 1.3 bietet ausreichende Stabilität und Möglichkeiten für Linuxsysteme und wird nach wie vor vom Hersteller gepflegt. Aus diesem Grund wurde auf eine Verwendung der neueren Version verzichtet.

Die Software ist modular aufgebaut, d.h. durch entsprechende Module können Funktionen hinzugefügt oder entfernt werden. Beispielsweise lässt sich die Kommunikation zwischen Server und Client mittels SSL verschlüsseln, was extrem nützlich bei der Nutzung von Intrusion Detection Systems ist. In der Diplomarbeit wurde allerdings auf die meisten Module verzichtet, da das System nur zu Testzwecken aufgebaut ist und keine sichere Verbindung benötigt wird.

2.1.4 Basic Analysis and Security Engine (BASE)

Die Meldungen und gewonnenen Daten vom Snortsensor werden in einem Format gespeichert, das für den Menschen nicht lesbar ist. Außerdem ist das manuelle Auslesen der Datenbanken sehr zeitaufwändig und bietet kaum Übersicht. Aus diesem Grund muss eine Möglichkeit zur einfachen und übersichtlichen Ausgabe und Analyse der gewonnenen Daten genutzt werden. BASE bietet diese Möglichkeit. Die Applikation schafft ein Webinterface, welches automatisch den Snortsensor im Netzwerk ortet und ausgibt. Das Webinterface hat eine integrierte Benutzer-Authentifikation und kann Dateien und Datenbänke direkt editieren. BASE basiert auf der Skriptsprache PHP, weshalb es notwendig ist, diese als Modul dem Apache HTTP Server mitzugeben. In Abbildung 2.1 ist die Oberfläche dargestellt.

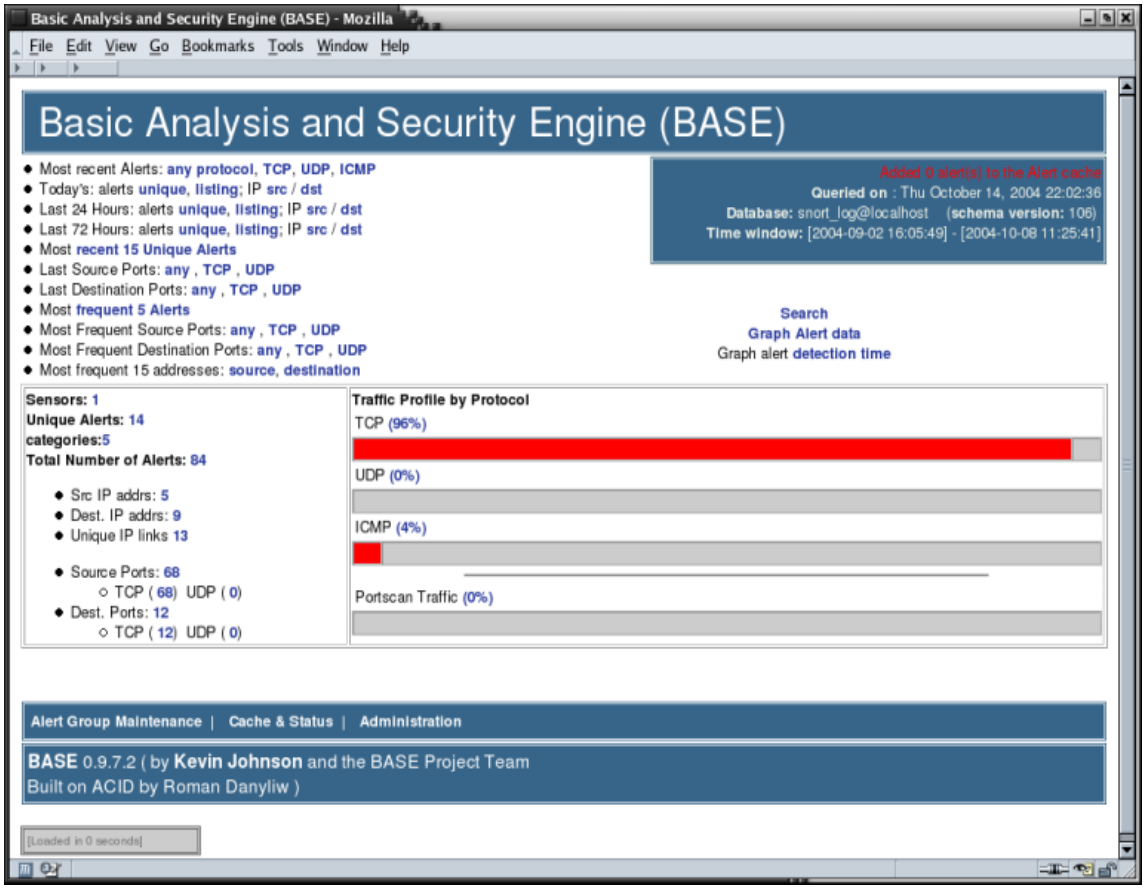

Abbildung 2.1: BASE Konsole

2.1.5 Verschiedene Toolchains

Eine Toolchain (engl. für Werkzeugkette) ist ein Gesamtsystem, das zur Programmierung von Anwendungen und Betriebssystemen verwendet wird. In dem Gesamtsystem sind alle wichtigen Werkzeuge vereint, um Anwendungen und Linuxkernel zu konfigurieren, kompilieren und installieren. In der Diplomarbeit wird eine Toolchain genutzt, die die Kompilierung von einer Systemarchitektur auf eine andere ermöglicht. Dieses „cross-compiling“ (engl. für Kreuzkompilierung) bietet Vorteile gegenüber der Kompilierung auf der

Zielarchitektur. In der Regel sind die Zielarchitekturen weniger leistungsfähige Systeme. Ausgerüstet mit weniger Speicher als handelsübliche Desktoparchitekturen würde es wesentlich mehr Zeit in Anspruch nehmen, die Anwendungen zu kompilieren. Des Weiteren müssten sich alle Bibliotheken und Toolchainkomponenten im Speicher des Zielsystems befinden.

Die Toolchain von Snapgear besteht aus den Werkzeugen:

1) Make: zur Automatisierung des Build- und Kompiliervorgangs
2) Cross Compiler Collection: Cross Compiler verschiedener Programmiersprachen (zum Beispiel: C, C++ und Fortran)
3) Binutils: Linker, Assembler und andere
4) Debugger
5) Build System: Autotools (Autoconf, Automake, Autoheader, libtool)

2.1.6 Weitere Software

Zur Realisierung der geforderten Aufgaben sind noch viele weitere Anwendungen, Softwarepakete und Entwicklungs-Kits notwendig. Dies liegt zu einem erheblichen Teil daran, dass die genutzten Anwendungen diverse Software auf dem System voraussetzen. Um ein stabiles Funktionieren zu gewährleisten, müssen alle Voraussetzungen erfüllt sein. Dies sind im Wesentlichen Bibliotheken, auf die während des Kompiliervorgangs zugegriffen wird. Außerdem enthalten sie Daten und Verlinkungen, die von mehreren Anwendungen genutzt und umgesetzt werden. Weiterhin gibt es optionale Anwendungen, die zwar für die Nutzung nicht unbedingt erforderlich sind, aber

einige Vorteile hinsichtlich Performance, Sicherheit und Vereinfachung bieten. Folgend ist Software aufgelistet, die nicht standardmäßig in Ubuntu integriert ist. Allerdings wurde auf die Verwendung der Programme zur Verbesserung der Sicherheit im Fall des IXP 2350 Hostsystem verzichtet, da eine Anpassung aller Anwendungen für die Hostarchitektur einen unverhältnismäßigen Aufwand bedeutete.

Bastille Linux

Die Anwendung dient dazu ein ausgewähltes Betriebssystem zu “härten“. Damit ist die Verbesserung der Sicherheit des Betriebssystems gemeint. Bastille Linux übernimmt dabei automatisiert die Konfiguration der sicherheitsrelevanten Einstellungen. Dies betrifft unter anderem die Benutzerverwaltung, Datenverwaltung, Bootsicherheit und Firewall. Für eine ernsthafte Nutzung von Intrusion Detections Systems ist ein solches Werkzeug absolut unerlässlich.

OpenSSL

Um eine sichere Kommunikation zwischen zwei Systemen zu schaffen, müssen die übermittelten Nachrichten unbedingt gesichert sein. Dies geschieht im Falle von OpenSSL durch eine Verschlüsselung der Nachrichten. Es können dabei verschiedene Schlüssel verwendet werden, die unter anderem mit den Algorithmen RSA und DSA berechnet werden.

libpcap

Eine freie Programmbibliothek, die den hardwarenahen Zugriff auf die Netzwerkkarte ermöglicht. Pakete, die über das Netzwerk transportiert werden, können von libpcap unter Umgehung des Protokollstacks entgegen genommen und weitergeleitet werden. Auf diese Weise kann eine Auswertung der Pakete erfolgen, bevor sie vom Betriebssystem verarbeitet werden. Diese Programmbibliothek ist ein wichtiger Bestandteil von Snort.

pcre

„Perl-Kompatible Reguläre Ausdrücke“ ist eine Programmbibliothek, die reguläre Ausdrücke auswertet. Syntax und Semantik entsprechen dabei den regulären Ausdrücken von Perl 5 (eine Programmiersprache) und POSIX. Reguläre Ausdrücke sind Beschreibungen von Mengen bzw. Untermengen unter Verwendung bestimmter syntaktischer Regeln.

NTP

Zur Synchronisation zwischen mehreren physikalisch getrennten Geräten wird ein Network Time Protokoll genutzt. Synchronisierte Zeit ist notwendig um ein bestimmtes Geschehnis einer bestimmten Zeit zuzuordnen. Das heißt, Zeit und Geschehnis müssen korrelieren. Wenn keine Zeitsynchronität herrscht, kann es passieren, dass bestimmte Geschehnisse einer falschen Zeit zugeordnet werden. Geschieht dies, lässt sich das Geschehnis, im Falle von Snort ein Alarm, nicht dem korrekten Netzwerkpaket zuordnen.

2.2 Intrusion Detection Systems (IDSs)

2.2.1 Allgemein

Intrusion Detection Systems (engl. für Angriff detektierende Systeme) sind Systeme, die auf das Erkennen von Eingriffen in, durch IDS gesicherte, Netzwerke spezialisiert sind. Sie dienen dabei lediglich der Detektion von Eingriffen, das heißt sie agieren nicht. Sie protokollieren lediglich verdächtige Pakete. IDSs können also keine Eingriffe bzw. Angriffe verhindern, wie beispielsweise Firewalls. Ein gut gesichertes Netzwerk besteht also immer aus mehreren Anwendungen und Systemen. Idealerweise könnte sich ein IDS vor

und hinter einer Firewall befinden. Dadurch können mögliche Angriffe erkannt und gebannt werden, sowie eventuelle Lücken in der Sicherung aufgezeigt werden. IDSs werden im Wesentlichen in zwei Gruppen unterteilt: Netzwerk-IDSs (NIDS) und Host-IDSs (HIDS).

HIDS

Host-IDSs erkennen Angriffe auf Betriebssysteme, Anwendungen und Kernel-Daten. Die Systeme haben Zugriff auf Log-Dateien, Fehlermitteilungen, Anwendungen und alle anderen Ressourcen, die sich auf dem Host befinden. Zusätzlich können Daten von Anwendungen überwacht werden. In diesem Fall weiß das HIDS, wie die normalen Daten der Anwendungen aussehen müssen. Wird eine Abweichung von dieser Norm festgestellt, schlägt das HIDS Alarm. HIDSs verfügen über genaues Wissen ihres Hosts und welches Verhalten normal für ihn ist. Kleinste Abweichungen können erkannt und protokolliert werden. Auf diesem Wege können HIDSs, sofern sie richtig konfiguriert sind, fast alle Angriffe erkennen.

NIDS

Netzwerk-IDSs dienen zur Überwachung ganzer Netzwerke. Das heißt, sie erkennen Angriffe, bevor sie an ihr spezifisches Ziel gelangen. Dadurch wird ein Blick auf das gesamte Netzwerk möglich. NIDSs können an verschieden Stellen im Netzwerk platziert werden. Vor dem Netzwerk detektieren sie, was in das Netzwerk hinein geht und was heraus kommt. NIDSs in einem Netzwerk überwachen den Datenverkehr, der zwischen verschiedenen Systemen in Netzwerken stattfindet. Im Idealfall werden also Sicherheitsverstöße im Netzwerk genauso protokolliert, wie Angriffe von außen. Um ein größeres

Netzwerk effizient zu schützen, muss der Sensor eine außerordentlich hohe Bandbreite besitzen und fähig sein, riesige Datenmengen zu verarbeiten.

Während HIDSs also eine maximale Erfolgsrate mit relativ wenig Leistung erlauben, brauchen NIDSs für eine annähernd ähnliche Erfolgsrate ungeheure Mengen an Ressourcen. Im Gegensatz zu HIDSs sind NIDSs effizienter hinsichtlich Ressourcen und Finanzen. Jedes HIDS kann immer nur genau einen Host überwachen, wohingegen ein NIDS mehrere Hosts bzw. ganze Netzwerke überwachen kann. Da, wie oben beschrieben, HIDSs kleinste Unstimmigkeiten erkennen können, werden weniger „false positives“ erzeugt. „False positives“ sind Alarme, die irrtümlich ausgelöst werden. Da NIDSs nicht die genauen Spezifikationen ihrer zu schützenden Netzwerke kennen und es prinzipiell keinen „Normalzustand“ gibt, ist es sehr schwierig bösartige Pakete von gutartigen zu unterscheiden. Im Gegensatz dazu gibt es sogenannte „false negatives“. Dies sind nichterkannte, möglicherweise schadhafte Pakete. Ein „false positiv“ verursacht also schlimmstenfalls mehr Traffic, während ein „false negativ“ ein großes Sicherheitsproblem darstellen kann. Um das Generieren von „false positves“ auf ein verträgliches Niveau zu verringern und die Erzeugung von „false negatives“ möglichst ganz zu unterbinden, ist es notwendig das IDS perfekt abzustimmen. Dies geschieht über sogenannte Detection Rules, welche im nächsten Abschnitt näher beschrieben sind.

IDSs besitzen verschiedene Methoden, um Angriffe zu detektieren. Jede Methode hat ihre Stärken bei unterschiedlichen Arten von Angriffen. Gut abgestimmte IDSs sollten also immer mehrere Typen der Detektion in sich vereinen.

Signature Detection

Dieser Typ der Detektion erkennt schadhafte Pakete mit Hilfe einer dem Paket typischen Signatur. Ist die Charakteristik eines schadhaften Pakets einmal bekannt, wird diese Signatur angefertigt und in einer Datenbank gespeichert. Um false positives zu vermeiden, müssen die Signaturen die Charakteristik so exakt wie möglich beschreiben. Jedes einkommende Paket wird nun mit der Signaturendatenbank verglichen. Trifft ein Paket auf eine Signatur zu, wird ein Alarm generiert, der die Beschreibung der Signatur enthält. Signature Detection ist somit die genaueste Art der Detektion von bekannten Attacken. Jedes schadhafte Paket kann erkannt werden. Voraussetzung dafür ist aber immer, dass eine Signatur existiert. Es gibt nur wenige Möglichkeiten diesen Typ der Detektion zu umgehen und diese können durch andere Detektionstypen erkannt werden.

Als Beispiel für Signature Detection kann die Detektion des Wurmes “SQL Slammer“ genannt werden. Der Wurm befällt Microsoft SQL Server und nutzt Pufferüberläufe. Der Wurm besteht aus nur einem einzigen UDP-Paket. Dieses Paket hat einen spezifischen Inhalt, der den Wurm eindeutig identifiziert. Diese besondere Signatur erkennt Snort durch den Vergleich mit der Signaturdatenbank und meldet somit einen Angriff.

Voraussetzung für diesen Typ der Detektion ist das Wissen über spezifische Angriffe. Neue Angriffe benötigen neue Signaturen, womit die Signaturdatenbank zum Abgleichen stetig wächst. Jedes Paket muss mit jeder Signatur in der Datenbank abgeglichen werden. Dadurch entsteht ein sehr hoher Bedarf an Rechenleistung und Bandbreite. Übertrifft der Bedarf die zur Verfügung stehenden Ressourcen des IDSs, können Pakete übersehen oder gar verworfen werden. In einer solchen Situation besteht die Möglichkeit eines false negative. Da Signature Detection immer nur paketweise analysiert und die Ergebnisse der Analyse nicht miteinander verknüpft, können mehrstufige

Attacken, basierend auf mehreren verschiedenen Paketen, nicht erkannt werden.

Anomaly Detection

Diese Art der Detektion erstellt eine Norm nach einer spezifischen Messzeit ohne Angriffe. Spätere Pakete werden mit dieser Norm verglichen. Weicht das Muster einer Aktivität von dieser Norm ab, wird ein Alarm generiert. Dadurch können auch die Aktivitäten einzelner Anwender überprüft werden. Meldet sich beispielsweise ein Nutzer über mehrere Monate während der Geschäftszeiten an seinem Rechner an, entspricht dies der Norm. Ändert der Nutzer dann plötzlich sein Anmeldeverhalten und loggt sich spät nachts ein, erkennt das IDS dies als Abweichung der Norm und ein Alarm wird generiert. Außerdem können Überschreitungen von Privilegien von Nutzeraccounts überwacht werden. Greift zum Beispiel ein Nutzer auf Systemdateien zu, für die er eigentlich keine Rechte besitzt, wird das als Normverstoß erkannt und ein Alarm generiert. Im Gegensatz zur Signature Detection kann der Angreifer nicht vorhersehen, welche Angriffsmethoden bekannt sind, da er nicht die erstellte Norm kennt.

Als Nachteil ist zu sehen, dass während der Normerstellung sicherzustellen ist, dass nur normale Aktivität herrscht. Sollte beispielsweise ein Nutzer während dieser Zeit immer nachts Firmengeheimnisse stehlen, wird das IDS dies zur Norm erklären und in späteren Fällen keinen Alarm generieren. Dadurch ist diese Art der Detektion für viele false negatives und postives verantwortlich. Dies ist ein großes Problem dieses Typs und er wird daher seltener oder weniger umfangreich, als die Signature Detection eingesetzt.

Integrity Verification

Diese Art der Detektion ist eine einfache, aber effiziente Methode, Angriffe zu erkennen. Für jede Datei auf dem System wird eine Prüfsumme erstellt. In periodischen Abständen wird diese Prüfsumme dann mit den Dateien verglichen. Gab es einen unautorisierten Eingriff in eine Datei, dann unterscheiden sich die beiden Prüfsummen und ein Alarm wird generiert. Während des normalen Systemablaufs werden viele Dateien auf dem System verändert. Das IDS muss darauf vorbereitet sein um false positives zu vermeiden. Die Prüfsumme muss jedesmal, wenn eine Datei legal verändert wird, angepasst werden.

Dieses Verfahren, mit all seinen Nachteilen funktioniert nur mit HIDSs. Außerdem müssen die Prüfsummen der Originaldateien irgendwo gespeichert werden. Für den Angreifer entsteht so die Möglichkeit, diese Prüfsummen zu verändern. Darin besteht das große Risiko dieses Verfahrens. Durch Verwendung einer gesicherten Datenbank kann dieses Risiko verringert werden.

Vor- und Nachteile von IDS

Vorteile:

- Komplexere Analysen von Datenpaketen, als zum Beispiel Firewalls
- Überwachung von externen und internen Verkehr möglich
- Hohe Flexibilität bei Anpassung an spezielle Netzwerke und Systeme

Nachteile:

- Große Anzahl von false positives
- Komplexe und sehr vielseitige Meldungen, die unter anderem spezielle Analysesoftware benötigen um verarbeitet zu werden
- Hohe Anforderung an Nutzer von Snort

2.2.2 Snort IDS

Snort ist ein Intrusion Detection System, das 1998 von Martin Roesch, dem Gründer von Sourcefire, erstmals herausgegeben wurde. Es ist ein freies und nicht-proprietäres Programm zur Echtzeitanalyse und Protokollierung von Netzwerkdatenpaketen. Die Beliebtheit von Snort begründet sich durch die Unterstützung aller Detektionstypen, sowie der Tatsache, dass es als Open Source Software gilt und somit stetig verbessert wird. Dadurch ist Snort zum de facto Standard im Bereich der Intrusion Detection Systems geworden.

Der Name „Snort“ leitet sich von dem Maskottchen, einem Ferkel mit großer schnaubender (engl.: snort) Nase, ab.

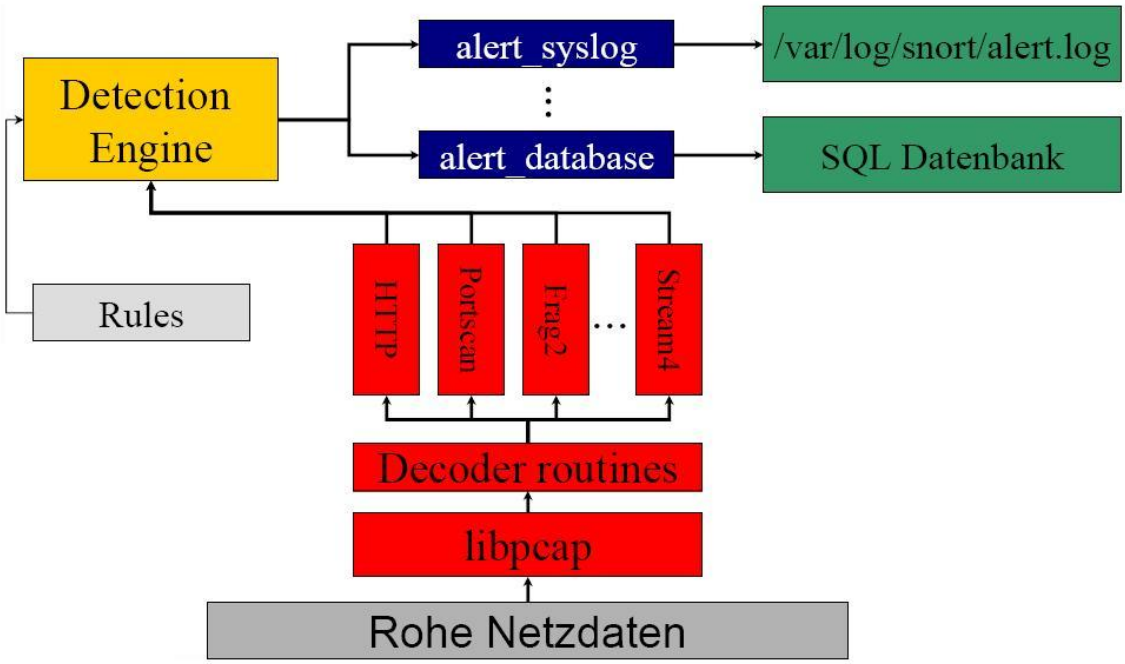

Abbildung 2.2: Übersicht der Paketverarbeitung von Snort (Quelle: @traktiv GmbH: Vortrag über Intrusion Detection)

Wie in Abbildung 2.2 aufgezeigt, durchlaufen die ankommenden Pakete diverse Zwischenschritte. Die rohen Netzdaten werden von dem Programm “libpcap“ entgegen genommen und an Snort weitergeleitet. Die Pakete werden von verschiedenen Präprozessoren auf ihre Eigenschaften untersucht und bestimmten Paketarten zugeordnet. Nachdem das geschehen ist, werden die Pakete mit den Detektionsregeln von Snort verglichen und das Ergebnis wird in verschiedene Datenbanken eingetragen.

Basierend auf der Protokollanalyse und dem Vergleichen und Durchsuchen von Inhalten kann Snort zu Erkennung von diversen bösartigen Angriffen genutzt werden. So können beispielsweise Pufferüberläufe, Portscans und CGI Attacken erkannt werden. Durch Anwenden von Detektionsregeln und der Fähigkeit, eine Fülle an Plugins zu nutzen, ist es äußerst flexibel einsetzbar. Die gesammelten Daten und gemeldeten Alarme können in Datenbanken

geschrieben werden und sind somit durch den Menschen lesbar und auswertbar.

In dieser Diplomarbeit wurde nicht die aktuelle Version 2.8.4 vom 7.4.2009 genutzt, da diese Probleme mit der Datenbank und BASE hatte. Stattdessen kommt die Version 2.8.3.2 zum Einsatz.

3 Etablierung des Servers für Snort

3.1 Einleitung

Das Snort System besteht aus einem Server und einem Sensor. Der Server dient der Speicherung der gewonnenen Analysedaten und der möglichen Fernsteuerung des Sensors. Der Sensor selbst verarbeitet die ankommenden Datenpakete und analysiert deren Inhalt. Es ist theoretisch möglich, dass sich Server und Sensor auf ein und demselben Computersystem befinden. Dies ist allerdings aus Sicherheitsgründen nicht empfehlenswert. Außerdem sollten im Sensor alle Rechenressourcen zur Prozessierung der Pakete verwendet werden.

Der Sensor kann an verschiedenen wichtigen Punkten im Netzwerk platziert werden. Dies hängt von der Hauptaufgabe des Sensors ab. Zur Überwachung des internen Datenverkehrs kann der Sensor innerhalb eines Netzwerkes, zum Beispiel vor einem Datenserver, platziert werden. Um Pakete aus dem Internet, oder von anderen externen Datenquellen zu analysieren, muss der Sensor sich am Eingang des internen Netzes befinden. Da ein solches Netzwerk in der Regel durch eine Firewall geschützt ist, bietet es sich an zwei Sensoren zu platzieren. Wobei sich ein Sensor vor der Firewall befindet und ein anderer dahinter. Dadurch kann zusätzlich zu der Analyse des Datenverkehrs noch die Funktion der Firewall überwacht werden. In Abbildung 3.1 ist der allgemeine Aufbau des Snort Sensor-Server-Netzwerkes veranschaulicht.

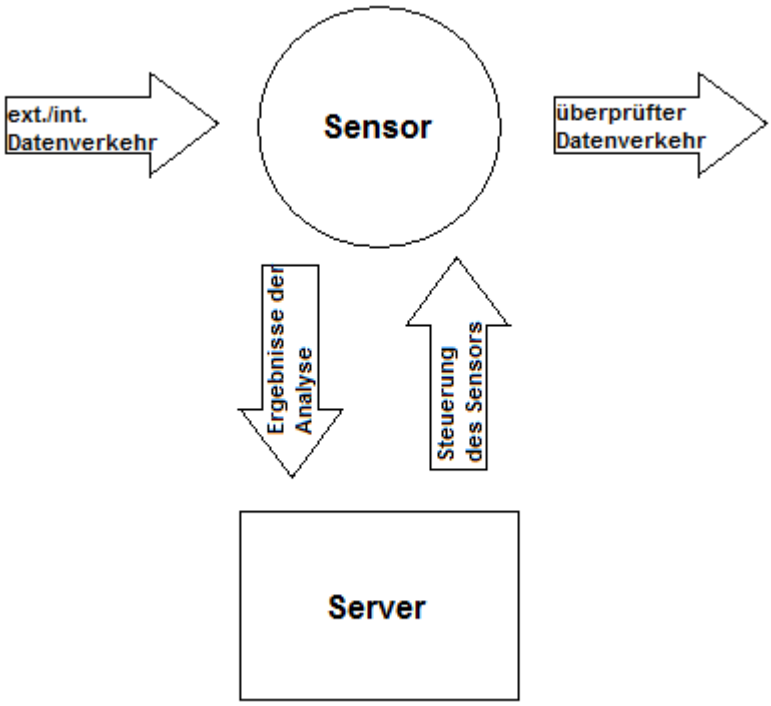

Abbildung 3.1: Sensor/Server Architektur

In diesem und dem folgenden Kapitel wird die Installation des Servers und Sensor erläutert. Wobei zuerst der Server etabliert wird und danach der Sensor. Um die Verhaltensweise der Sensors und dessen Installation zu verstehen, wird er zuerst auf einem Desktop-PC etabliert. Darauf folgend und mit besseren Verständnis für Snort und dem Sensor wird er auf dem Netzwerkprozessor installiert.

3.2 Voraussetzungen für den Server

Als Betriebssystem des Servers wurde Ubuntu 8.10 gewählt. Dies geschah ausschließlich aus Kompatibilitätsgründen. Prinzipiell muss das Betriebssystem

nicht dem des Sensors entsprechen. Es kann also auch eine andere Linux-Distribution, oder sogar Microsoft Windows genutzt werden.

Für den Server ist ein Netzwerkadapter ausreichend, da er ausschließlich mit dem Sensor verbunden sein muss. Die IP-Adresse sollte im gleichen Adressraum liegen, wie die des Sensors.

Vor der Installation der einzelnen Serverkomponenten kann das System, bei Bedarf, mit Bastille Linux gehärtet werden.

3.3 Installation der Serverkomponenten

3.3.1 Installation von MySQL

Die aktuelle Version von MySQL kann von der Homepage http://www.mysql.com bezogen werden. Glücklicherweise bietet Sun Microsystems, der Entwickler von MySQL, das Programm als bereits vorkompiliertes Paket für viele Betriebssysteme und Rechnerarchitekturen an. So müssen lediglich "libncurses5-dev“ – Bibliotheksdateien zusätzlich installiert werden und MySQL ist bereit konfiguriert zu werden.

3.3.2 Konfiguration von MySQL

Zur Konfiguration wird das MySQL Kommandofenster verwendet. Um dies zu nutzen, muss der MySQL Daemon gestartet werden. Darauffolgend wird das Kommandofenster gestartet. Hierbei ist zu beachten, dass der Nutzer als Administrator im MySQL eingeloggt ist. Nun müssen zwei Datenbanken für Snort erstellt werden. Die Datenbank „snortdb“ dient zur Verwaltung der aktuellen, durch Snort gewonnenen Daten, während die Datenbank

„snortarchivedb“ zum archivieren der angefallenen Daten dient. Im Paket von Snort ist ein Skript enthalten, welches automatisch die vollständige Datenbankstruktur erstellt. Es ist nicht zwingend notwendig dieses Skript zu nutzen, aber sehr empfehlenswert. Snort und BASE sind für dieses Datenbankschema vorkonfiguriert und bedürfen somit keiner manuellen Anpassung hinsichtlich MySQL. Zuletzt müssen zwei Nutzer mit unterschiedlichen Rechten Zugangsberechtigung erhalten: Ein Nutzer stellvertretend für Snort und einer für BASE. Dies ist erforderlich um Daten in die Datenbanken einzufügen und auszulesen. MySQL ist damit fertig konfiguriert und bereit, Daten von dem Sensor zu erhalten.

3.3.3 Verbesserung der Sicherheit von MySQL

Da die Datenbank ein sicherheitsrelevanter Bereich innerhalb des Snort Sensor-Server-Netzwerks ist, sollte in der Praxis darauf geachtet werden, MySQL ausreichend zu sichern. Wichtig hierfür sind vor allem starke Passwörter und eine sinnvolle Vergabe von Rechten. Desweiteren werden bei der Installation von MySQL automatisch Benutzer mit angelegt, die es zu entfernen gilt. Während der Nutzer „Snort“ Lese- und Schreibrechte benötigt, reicht es für den Nutzer „BASE“ völlig aus, nur Leserechte zu erhalten. Das Passwort für den Administratorzugang von MySQL sollte nicht aus ganzen Wörtern bestehen. Im günstigsten Falle besteht das Passwort aus Buchstaben, Zahlen und ASCII-Zeichen.

3.3.4 Installation von BASE

Die Installation von BASE ist vergleichsweise einfach. Es gilt lediglich zu beachten, dass BASE auf PHP basiert und mit Apache arbeitet. Aufgrund dessen müssen PHP und Apache auf dem Server installiert sein. Folgend braucht BASE nur noch in ein Unterverzeichnis von Apache kopiert werden. Ist dies geschehen, muss die sehr umfangreiche Konfigurationsdatei von BASE bearbeitet werden. Im Anhang A.1 dieser Arbeit ist ein Ausschnitt einer vorkonfigurierten Datei hinterlegt. Ausschlaggebend für das reibungslose Funktionieren von BASE sind nur wenige Einstellungen. BASE muss vor allem den Namen und Ort der Datenbanken kennen. Die restlichen Einstellungen sind sehr speziell und dienen der visuellen Verbesserung sowie der Personifikation.

3.3.5 Erhöhung der Sicherheit des Servers

Um sicherzustellen, dass der Server selbst und der Datenaustausch zwischen Sensor und Server keine Sicherheitslücken aufweist, können einige zusätzliche Hilfsmittel verwendet werden. Zur Verschlüsselung des Datenaustausches zwischen Server und Sensor kann OpenSSL genutzt werden. Dies ist insofern wichtig, da die Authentifizierung für die Datenbanken unverschlüsselt übertragen wird und es ein leichtes wäre, diese auszulesen. Desweiteren könnte ein potenzieller Angreifer eigene Datenpakete in die Kommunikation einbringen.

Damit nur privilegierte Nutzer auf die Webserver von Apache zugreifen können, wird eine sogenannte „.htaccess“-Datei erstellt und verwendet. Mit Hilfe dieser Datei werden Restriktionen auf einzelne Dateiordner und ihre Inhalte gelegt. BASE muss nun eine Authentifizierung durchführen, um auf den Apacheserver zugreifen zu dürfen. Die Informationen für diesen BASE-Benutzer werden in

einer kleinen Datei hinterlegt, die lediglich Benutzernamen und Passwort enthält.

4 Etablierung des Sensors

4.1 Ein Desktop-PC als Sensor

Zunächst wird das IDS auf einem herkömmlichen Desktop-PC installiert. Dies dient dem Zweck eines Vergleiches der Möglichkeiten und Performance zwischen Snort auf einem Desktop-PC und auf dem Netzwerkprozessor. Als Desktop-PC wurde ein System mit einem Intel Pentium 4 Prozessor und 1024 MB Arbeitsspeicher verwendet. Ubuntu 8.10 als Linuxvertreter dient dabei als Betriebssystem. Der genutzte Desktop-PC ist im Vergleich zum aktuellen Stand der Technik als eher langsames System einzuordnen, reicht aber für Vergleichszwecke völlig aus.

Die Installation der einzelnen Komponenten, abgesehen von Snort, ist der Installation des Servers sehr ähnlich. Dasselbe gilt auch für die Voraussetzungen des Sensors.

4.1.1 Installation von Snort

Zur Installation von Snort gibt es zwei verschiedene Möglichkeiten. Zum einem ist Snort als vorkompiliertes Paket erhältlich. Dieses Paket müsste einfach ausgeführt werden und die, im Paket enthaltenen Installationsroutinen übernehmen die Konfiguration und Installation. Diese Möglichkeit ist sehr bequem und einfach für den Nutzer und sollte unbedingt verwendet werden. Einzige Voraussetzung ist ein spezifisches Betriebssystem und eine bestimmte Systemarchitektur. Verwendet man, wie im Fall dieser Arbeit, ein anderes Betriebssystem kann das vorkompilierte Paket nicht genutzt werden.

Die zweite Möglichkeit besteht darin, den Quellkode von Snort selbstständig für das spezifische Betriebssystem und die verwendete Systemarchitektur zu kompilieren. Voraussetzung für Snort ist die Installation von „libpcap“ und „pcre“. Diese Bibliothekspakete werden von Snort verwendet. Sie können problemlos als Quelldateien heruntergeladen werden. Es wird empfohlen, die Bibliotheken nicht in ein selbst gewähltes Verzeichnis zu installieren. Stattdessen sollten die Standardverzeichnisse belassen werden. Dadurch bedarf die Installation von Snort keiner speziellen Angaben hinsichtlich des Orts der installierten Bibliotheken.

Desweiteren sollte sich ein MySQL Klient auf dem Sensor befinden, damit Snort direkt mit MySQL-Unterstützung kompiliert und installiert werden kann.

Sind alle Voraussetzungen für Snort geschaffen, kann mit der Installation begonnen werden. Bei der Kompilation und Installation ist zu beachten, dass mit Hilfe von “--with-mysql“ ein Verweis zu MySQL für Snort angegeben wird. Kompilation und Installation werden durch das Ausführen der Befehle “make“ und “make install“ realisiert. Nach der Installation ist Snort laufbereit, wenn auch nur mit sehr rudimentären Funktionen.

4.1.2 Empfehlung für Konfiguration des Sensors in der Praxis

Die gewählten Einstellungen des Desktop-PC sind für eine seriöse Nutzung nicht praktikabel. Dies liegt darin begründet, dass der Netzwerkprozessor nur eingeschränkte Möglichkeiten besitzt und der Desktop-PC somit an diese Gegebenheiten angepasst werden muss. Um ein effektives Intrusion Detection System aufzubauen, sind einige zusätzliche Handlungen notwendig.

Platzierung des Sensors im Netzwerk

Idealerweise sollte der Sensor über drei Netzwerkschnittstellen verfügen. Eine Schnittstelle dient der Verbindung zum Server. Die beiden anderen Schnittstellen dienen dem Durchsatz des Netzwerkverkehrs. In Abbildung 4.1 ist dies verdeutlicht.

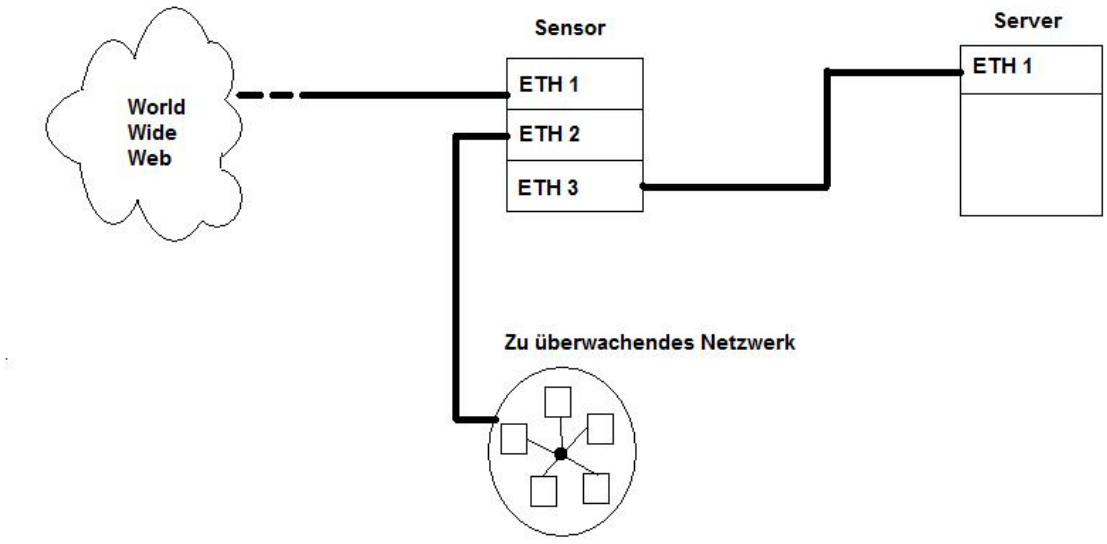

Abbildung 4.1: Platzierung des Sensors und Verwendung der Schnittstellen

Es existieren weitere Möglichkeiten des Aufbaus. Die beste Wahl ist schwer zu treffen, da verschiedene Faktoren berücksichtigt werden müssen. Beispielsweise ist das in Abbildung 4.1 gegebene Beispiel nicht für sehr starken Netzwerkverkehr geeignet. Im Falle hoher Netzwerklast sollte ein professionelles Switch gewählt werden, welches befähigt ist, Datenverkehr auf mehrere Ports zu duplizieren.

Verbesserung der Performance mit Hilfe von Barnyard

Die direkte Übergabe der gewonnen Daten in die MySQL Datenbank kostet Snort viele Ressourcen, die an anderen Stellen wesentlich sinnvoller eingesetzt werden könnten. Aus diesem Grund wurde Barnyard entwickelt. Snort kann die gewonnen Daten im binären Format an Barnyard übergeben. Dies spart Rechenleistung, da Snort die Daten nicht mehr in ein für Menschen lesbares Format umwandeln muss. Stattdessen formatiert Barnyard die Daten und schreibt sie in die MySQL Datenbank.

Erweiterung des Regelwerks von Snort

In dem Quellarchiv von Snort sind viele Standardregeln zur Erkennung von schadhaften Zugriffen enthalten. Diese rudimentären Regeln decken aber nicht die volle Bandbreite der Möglichkeiten ab, die Snort zur Verfügung stehen. Es gibt mehrere Möglichkeiten zur Verbesserung der Erkennung. Auf der Homepage von Snort werden für registrierte Mitglieder in regelmäßigen Abständen die neuesten Signaturen und Regeln zur Verfügung gestellt. Desweiteren können diese Regeln auch manuell erweitert bzw. komplett neu geschrieben werden. Dies ist vor allem hilfreich, um sehr spezielle Angriffe zu erkennen. Ausführliche Beschreibungen für die Regelsyntax sind auf der Homepage von Snort vorhanden. Es gilt zu beachten, dass die Schaffung ausreichend sicherer Regeln sehr zeitaufwändig und kompliziert ist, weswegen es sehr viel Zeit bedarf ein wirklich effektives IDS zu schaffen.

4.1.3 Fazit der Sensorinstallation auf einem Desktop-PC

Trotz der Tatsache, dass der Quellkode von Snort manuell kompiliert und installiert werden muss, ist es doch vergleichsweise einfach. Es genügen die Standardbefehle von Linux zur Installation und die Nutzung der Konfigurationsdatei von Snort. Der nächste Schritt ist nun, Snort für den Netzwerkprozessor und damit einer anderen Systemarchitektur zu kompilieren. Die Installation von Snort auf dem Desktop-PC hat Erkenntnisse erbracht, die helfen sollen, den nächsten Schritt zu realisieren.

4.2 Implementierung von Snort auf dem Netzwerkprozessor

4.2.1 Vorwort

Als Weg zur Implementierung von Snort auf dem Netzwerkprozessor wurde entschieden, es als Software auf dem ADI Roadrunner auszuführen. Auf der Entwicklungsplattform befindet sich ein rudimentäres Betriebssystem auf Linuxbasis. MontaVista Pro 3.1 wurde von MontaVista speziell für die Anwendung in eingebetteten Systemen entwickelt. Allerdings ist diese Distribution proprietär. Desweiteren handelt es sich dabei um eine stark veraltete Software und MontaVista ließ den Support für dieses Produkt vor Jahren auslaufen.

Weiterhin wurde Snort nicht mit dem Ziel entwickelt, um auf eingebetteten Systemen zu arbeiten. Aus diesem Grund bietet Snort keine Unterstützung während des Kompilations- und Installationsvorgangs für eingebettete Systeme.

4.2.2 Auswahl der richtigen Werkzeuge

In Kapitel 4.1 wurde Snort für eine sehr gängige Plattform kompiliert. Für diesen Zweck reichten die Standardcompiler von Linux vollkommen aus. Der Netzwerkprozessor aber verwendet keinen Prozessor der Intel Pentium Familie. Stattdessen wird ein ARM Prozessor genutzt. In diesem Fall muss Snort also für eine andere Plattform kompiliert werden. Für diese Aufgabe wird der Crosscompiler verwendet. Hierbei handelt es sich um sehr spezielle Software, da sie exakt für die Zielplattform geschaffen sein muss. Weiterhin muss der Crosscompiler befähigt sein, mit Big Endian zu kompilieren. Big Endian ist eine spezielle Speicherorganisation für einfache Zahlenwerte, wobei das Byte mit den höchstwertigen Bit zuerst gespeichert wird. Diese Voraussetzung erfüllt der vorkonfigurierte Crosscompiler von Snapgear. Um den Crosscompiler zu testen, reicht es aus, ein einfach C-Programm für die Zielarchitektur zu kompilieren und auf ihr auszuführen. Es genügt, ein Programm zu schaffen, welches eine Textzeile ausgibt.

Zur Kompilation anderer Software und des Linuxkernels werden wiederrum andere Crosscompiler gebraucht. Leider existieren für diese speziellen Zwecke keine vorkompilierten Crosskompiler, weshalb es notwendig ist, diese selbst zu erstellen. Allerdings existiert ein Tool, welches dem Nutzer dabei hilft, diesen Crosscompiler zu erstellen. Dieses Tool trägt den Namen “crosstool“ und ist inklusive ausführlicher Beschreibung im Internet erhältlich. Dennoch muss damit gerechnet werden, dass die Erstellung des Crosscompilers sehr viel Zeit und Geduld beansprucht.

4.2.3 Kompilation und Installation der Bibliotheken für Snort

Da zum Ausführen von Snort die Bibliotheken “libpcap“ und “pcre“ unabdingbar sind, müssen sie auch auf dem Netzwerkprozessor implementiert werden. Bei der Cross-Kompilation von “libpcap“ muss die verwendete Version des Linuxkernels auf der Zielplattform angegeben werden. Dies geschieht über den Ausdruck:

ac_cv_linux_vers=2

Weiterhin ist es notwendig anzugeben, für welches Betriebssystem die pcap-Bibliotheken genutzt werden. Dies wird über den folgenden Befehl realisiert:

--with-pcap=linux

Abschließend muss noch ein Compilerflag gesetzt werden, welches bestimmt, dass mit Big Endian kompiliert wird. Der Befehl lautet:

-mbig-endian

Es ergibt sich nun folgender Ausdruck, mit Hilfe dessen “libpcap“ erfolgreich kompiliert werden kann:

ac_cv_linux_vers=2 CFLAGS=“-mbig-endian“ LDFLAGS=”-mbig-endian”
./configure --prefix=/.. --host=arm-linux

Zur Cross-Kompilation von “pcre” werden keine weiteren speziellen Befehle benötigt. Dadurch ist der zu verwendende Ausdruck relativ einfach:

LDFLAGS=“-mbig-endian“ CFLAGS=“-mbig-endian“ ./configure
--prefix=/.. --host=arm-linux

Es ist wichtig den gewählten Installationspfad zu vermerken, da dieser bei der späteren Installation von Snort dieser mit angegeben werden muss.

4.2.4 Kompilation und Installation von Snort

Die Aufgabe Snort für diese Zielarchitektur zu kompilieren und installieren, kann verschiedenartig angegangen werden. Tatsache ist, dass Snort nicht für ein derartiges System vorgesehen wurde. Im Gegensatz zu den Bibliotheksdateien ist Snort ein komplexes Programm, welches wiederrum mit anderer Software verknüpft ist.

Es ist theoretisch möglich, Snort direkt auf dem Netzwerkprozessor zu kompilieren. Dies ist allerdings mit erheblichen Problemen verbunden. Zunächst benötigt man eine Toolchain in Linux welches selbst auch für die spezielle Architektur kompiliert werden müsste. Desweiteren beansprucht eine Kompilation sehr viel Rechenleistung, die dem Netzwerkprozessor im Gegensatz zu einem Desktop-PC, nicht zur Verfügung stehen. Diese Möglichkeit wird aus den genannten Gründen nicht genutzt.

Ein vorkompiliertes Paket von Snort für die Zielarchitektur zu finden ist nahezu unmöglich. Es ist ausgesprochen unwahrscheinlich, dass bereits jemand Snort für exakt diese Architektur kompiliert hat.

Es ist also notwendig, mit Hilfe des Crosscompilers Snort für die Zielarchitektur zu kompilieren. Nach der erfolgreichen Installation der Bibliotheken müssen zunächst einige Änderungen in den, zur Installation genutzten, Skripten durchgeführt werden. Die Datei “configure“ dient der automatisierten Ermittlung von Systemparametern und sonstigen Informationen der Entwicklungs-umgebung. Unter anderem wird auch getestet, ob der gewählte Compiler korrekt funktioniert. Hierfür wird ein Beispielkode kompiliert und ausgeführt.

Konnte die Datei erfolgreich ausgeführt werden, ist der Compiler validiert und kann zur Kompilation des Programms genutzt werden. Da anstatt des Standardcompilers ein Crosscompiler verwendet wird, können die Beispielkodes zwar erfolgreich kompiliert, aber nicht ausgeführt werden. Dies liegt der Tatsache zugrunde, dass die Beispielkodes für die Architektur der Zielplattform kompiliert werden und somit auf einer Intel Pentium Plattform nicht ausgeführt werden können. Das Skript sieht dies als groben Fehler an und bricht die Installation ab. Aus diesem Grund muss der Abbruchsbefehl aus dem Skript gelöscht werden. Danach wird das Skript erfolgreich ausgeführt und es bereitet ein funktionsfähiges Installationsskript vor, das nun ausgeführt werden kann. Hierbei ist zu beachten, dass die Lage der Bibliotheken angegeben werden muss. Dieser Schritt ist notwendig, da die Installationsroutine sonst die Standardpfade verwendet. Die zuvor kompilierten Bibliotheken zur Verwendung auf dem Netzwerkprozessor befinden sich allerdings an einem anderen Platz. Mit folgendem Befehl kann die Lage der Bibliotheken angegeben werden:

--with-libpcap-libraries

--with-libpcap-include

--with-libpcre-libraries

--with-libpcre-include

Außerdem dürfen keine Standardpfade für die “include“-Dateien genutzt werden, da die Dateien sonst in der falschen Reihenfolge eingebunden werden. Da keine Standardpfade genutzt werden, müssen sie manuell mit der Option “-I/..“ in die Compilerflags eingefügt werden. Dies bedeutet hohen Aufwand, ist aber unvermeidlich. Zuletzt muss noch die Option “Big Endian“ gewählt werden. Der Finale Kompilations- und Installationsausdruck lautet wie folgt:

CPPFLAGS=“-mbig-endian“ LDFLAGS=“-mbig-endian“

CXXFLAGS=“-mbig-endian“ CFLAGS=“-mbig-endian –nostdinc

-I/..” ./configure --prefix=/.. –host=arm-linux --build=i686-pc-linux-gnu

--with-libpcap-libraries=/.. --with-libpcap-include=/..

--with-libpcre-libraries=/.. –with-libpcre-include=/..

--with-headers=/.. && make && make install

Nach der erfolgreichen Installation ist Snort bereit, auf einem eingebetteten System mit ARM-Architektur ausgeführt zu werden.

4.2.5 Kompilation des Linuxkernels und Erstellung einer Imagedatei

ADI Engineering stellt ein bereits kompiliertes Linux-Image zur Verfügung. Desweiteren stehen Treiber für die Installation der Netzwerkinterfaces bereit. Diese Treiber bedürfen besonderer Erwähnung, da sie für die Microengines programmiert wurden. Für die Nutzung dieser Treiber ist es notwendig den Linuxkernel von MontaVista zu nutzen, der der Entwicklungsplattform in vorkompiliertem Zustand, also als Image, beigelegt wurde. Andere Linuxkernel können nicht mit den Treibern arbeiten. Bei der Erstellung des Images wurde allerdings nicht darauf geachtet, dass Programme unterstützt werden, die Gleitkommaberechnung nutzen. Zur Ausgabe von Statistiken ist es unabdingbar derartige Berechnungen durchzuführen. Somit ist auch Snort darauf angewiesen. Da handelsübliche Netzwerkprozessoren keinen Prozessor zur Gleitkommaberechnung besitzen, müssen diese Prozessoren emuliert werden. Diese Option nennt sich “floating point emulation“, oder kurz “fpe“. Voraussetzung für die Nutzung ist, sie fest in den Linuxkernel zu integrieren. Da ADI das nicht getan hat, muss ein eigener Kernel kompiliert werden. Der Quellkode des Kernels wird nach offiziellen Angaben von ADI nicht bereitgestellt, da er proprietär ist. Allerdings befindet sich der Quellkode

dennoch – wenn auch versteckt – auf dem, der Entwicklungsplattform beigefügten, Speichermedium. Zur Kompilation des Kernels bedarf es einer anderen Toolchain, da die für Snort genutzte einen veralteten Compiler verwendet. Glücklicherweise stellt Snapgear eine Toolchain mit neuerem Compiler bereit.

4.2.6 Erstellung einer Ramdisk mit Snort

Das Linuxsystem auf der Entwicklungsplattform nutzt als Datenträger eine sogenannte Ramdisk, während üblicherweise Festplatten als Datenträger genutzt werden. Im Gegensatz zu den Festplatten werden bei jedem Neustart des Systems alle in der vorherigen Session geschriebenen Daten gelöscht. Bei dem nächsten Start des Systems befindet sich der Datenträger wieder im, auf der Ramdisk gespeichertem, Urzustand. Um Snort bei jedem Systemstart zur Verfügung zu haben, ist es also notwendig, Snort in die Ramdisk zu integrieren. Hierfür muss eine von ADI Engineering bereitgestellte Ramdisk erweitert werden. Dies geschieht mit Hilfe des “mount“-Befehls von Linux. Dadurch wird die Ramdisk mit einem Ordner verknüpft und es ist möglich auf sie zuzugreifen. Die Bibliotheken “libpcap“ und “libpcre“ müssen nun in diesen Ordner installiert werden. Danach muss Snort, unter Angabe der zuvor installierten Bibliotheken, in denselben Ordner installiert werden. Hierbei ist zu beachten, dass die maximale Speichergröße der Ramdisk nur 20 Megabyte beträgt. Dieser Platz reicht für Snort und dem Dateisystem von Linux nicht aus. Für dieses Problem existieren verschiedene Lösungsansätze.

Es ist möglich, die Größe der Ramdisk zu erweitern bzw. eine variable Ramdisk zu erstellen, die ihre Größe automatisch erweitert. Da die Ramdisk im Arbeitsspeicher des Systems läuft, ist dies im Falle der Entwicklungsplattform allerdings nicht empfehlenswert. Je mehr Arbeitsspeicher für die Ramdisk

verbraucht wird, umso weniger Ressourcen stehen Snort zum Ausführen seiner Analyse zur Verfügung. Da der Arbeitsspeicher der Entwicklungsplattform relativ knapp bemessen ist, wird darauf verzichtet, die Ramdisk zu erweitern.

Es kann von einer Funktion der Toolchain Gebrauch gemacht werden. Die sogenannte “Strip“-Funktion ermöglicht es, den Speicherbedarf verschiedener Dateien und Programme zu verringern. Die Funktion entfernt alle Informationen aus der Programmdatei, die ausschließlich zur Fehlersuche (engl.: debugging) benötigt werden. Schlussendlich werden noch alle installierten Anleitungen entfernt und die Grenze von 20 MB wird nicht überschritten. Zwar überschreitet das Programm selbst nicht die 20 MB Grenze, allerdings fehlen noch verschiedene Regeldateien und Signaturbibliotheken. Diese Dateien sind nicht zwingend notwendig um Snort auszuführen, allerdings hat Snort keinerlei Nutzen ohne sie.

Es muss also auf eine Alternative zurückgegriffen werden. Es existiert das sogenannte Network File System. Hierbei handelt es sich um ein von Sun Microsystems entwickeltes Protokoll, das den Zugriff auf Dateien im Netzwerk ermöglicht. Die Dateien werden dabei nicht direkt übertragen, sondern die Nutzer können auf sie zugreifen, als wären sie auf einem lokalen Datenträger. Es ist notwendig, dass eine dauerhafte Netzwerkverbindung zwischen Host und Klienten besteht. Diese Lösung ist zudem ideal zum testen, da nicht bei jeder Änderung von Snort eine neue Ramdisk erstellt werden muss

4.2.7 Test der Funktionalität von Snort

Bevor Snort selbst getestet werden kann, müssen zuerst die Ramdisk und das Linux-Image auf die Entwicklungsplattform gebracht werden. Dazu wird die Plattform über einen seriellen R232-Anschluss an einen Desktop-PC

angeschlossen. Danach kann mit Hilfe einer Terminalanwendung auf den ADI Roadrunner zugegriffen werden. Da die Ramdisk und das Linux-Image mit dem TFTP-Protokoll auf die Entwicklungsplattform übertragen werden, müssen zuerst die IP-Adressen von Host und Roadrunner festgelegt werden. Der Befehl dafür lautet:

ip_address –l <IP der Plattform> -h <IP des Hosts>

So geschehen, kann nun mit der Übertragung begonnen werden. Hierfür wird der folgende Befehl verwendet:

load -v -r -b <Speicheradresse in Hexadezimal> <Ramdisk/zImage>

Nach erfolgreicher Übertragung kann Linux wie folgt gestartet werden:

exec <Speicheradresse des Linuximages>

Man befindet sich nun in der Kommando-Shell von Linux. Hier kann mit Hilfe der mitgelieferten Treiber das Netzwerkinterface “ixp0“ aktiviert werden. Hierfür ist ein Kommando nötig, welches sich vom Linuxstandard leicht unterscheidet.

ifconfig <Interfacename> address <IP> netmask <Subnetzmaske>

Nach der Konfiguration kann Snort gestartet werden. Bei dem Start von Snort ist darauf zu achten, dass das korrekte Netzwerkinterface angegeben wird. Desweiteren benötigt Snort den Pfad zu seiner Config-Datei und einen Pfad für die Log-Dateien. Um Ressourcen zu sparen, sollte Snort außerdem im Daemon-Modus gestartet werden. Daemon-Modus bedeutet, dass das Programm im Hintergrund läuft. Das komplette Kommando zum Starten von Snort lautet:

./snort -i ixp0 -l /log/ -c /snort.conf –D

Konnte Snort erfolgreich gestartet werden, befindet es sich im Wartemodus und ist bereit Datenverkehr zu analysieren.

5 Performancetest beider Systeme mit Snort

5.1 Einleitung

Um Vor- und Nachteile der verschiedenen Systeme zu ermitteln, wird ein Performancetest durchgeführt. Es wird hierfür eine Netzwerklast erzeugt, die Snort verarbeiten und analysieren muss. Ziel ist es, aussagekräftige Ergebnisse zu erhalten, um einen Vergleich zu ziehen und somit gegebenenfalls das bessere System für die Nutzung eines IDS zu küren.

5.2 Bearbeitung der Konfigurationsdatei

Die Datei “snort.conf“ ist die beste Möglichkeit, die Einstellungen von Snort zu konfigurieren. Sie wird unter anderem genutzt um den zu analysierenden IP-Adressraum anzugeben, um einzelne Präprozessoren zu aktivieren, Outputplugins zu selektieren und die Wahl der genutzten Regeln festzulegen. Um vergleichbare Ergebnis zu erhalten, müssen die genutzten Konfigurationsdateien beider Testsysteme möglichst ähnlich sein. Dies gilt vor allem für die genutzten Regeln und Präprozessoren. Je mehr genutzt werden, desto höher ist die Rechenlast, die das System verursacht. Die wichtigsten Einstellungen werden folgend aufgeführt. Im Anhang A.2 befindet sich ein Auszug aus dieser Konfigurationsdatei.

5.2.1 Einstellung der Netzwerkvariablen

Die Netzwerkvariablen geben Snort Basisinformationen über das zu analysierende Netzwerk. Diese Variablen werden auch in den Regeln verwendet, weshalb es sehr wichtig ist, möglichst akkurat zu sein.

var HOME_NET

Diese Variable kennzeichnet den IP-Adressraum, der von Snort analysiert wird. Alle IP-Adressen außerhalb werden von Snort nicht beachtet. Die Adressen müssen im CIDR-Format angegeben werden. Zum Beispiel: “192.168.1.1/24“. Als weitere Möglichkeit kann die Adresse des genutzten Netzwerkinterfaces gewählt werden, also: “$eth0_ADDRESS“. Dadurch wird der IP-Adressraum des genannten Interfaces genutzt. Im Rahmen des Performancetests wird die Variable auf “any“ gesetzt. Damit analysiert Snort alle Netzwerkadressen.

RULE_PATH

Dieser Pfad zeigt Snort an, welcher Stelle sich die zu verwendeten Regeln befinden. Diese Variable sollte unbedingt angepasst werden, da Snort sonst keine Regeln nutzen kann.

Alle anderen Netzwerkvariablen werden bei ihrer Standardeinstellung belassen. Dies genügt für das Testsystem. Für eine praktikable Nutzung von Snort ist es allerdings erforderlich, diese auch anzupassen.

5.2.2 Konfiguration der Präprozessoren

stream5

Dieser Präprozessor wird von Snort genutzt, um den Status von TCP Streams zu erhalten. Hierfür können die Standardeinstellungen beibehalten werden.

http_inspect

Snort verwendet diesen Präprozessor, um abnormalen HTTP Verkehr zu erkennen und zu normalisieren. Diese Einstellung muss nicht angepasst werden. Es sollte lediglich sichergestellt werden, dass sich die Datei “unicode.map“ in dem Ordner befindet, der in der Variable “RULE_PATH“ angegeben wurde.

rpc_decode

Um manipulierten RPC Verkehr zu erkennen, wird dieser Präprozessor genutzt. Damit Snort keine “false negatives“ generiert, müssen hier die Ports angeben werden, die vertrauenswürdig sind. In dem Testnetzwerk werden keine Programme genutzt, die RPC verwenden. Aus diesem Grund können die Standardeinstellungen beibehalten werden.

bo

Dieser Präprozessor detektiert Back Orifice Trojaner. Zwar ist sicher, dass sich in dem Testnetzwerk keine Trojaner befinden, trotzdem wird diese Funktion aktiviert, da von dieser schadhaften Software ein großes Bedrohungspotenzial ausgeht. Um ihn zu aktivieren, muss einfach das Doppelkreuz vor der Linie “preprocessor bo“ entfernt werden.

ftp_telnet

Um zufällig eingefügte Telnetkodierungen aus dem Telnet und FTP Stream zu entfernen, nutzt Snort diesen Präprozessor. Es genügt hierbei, ihn durch einfaches Entfernen des Doppelkreuzes zu aktivieren.

5.2.3 Weitere Einstellungen

Output Plugins

Da Snort in der Regel in Verbindung mit einer Datenbank verwendet wird, müssen auch dazu einige Einstellungen getätigt werden. Für den Performancetest wird allerdings keine Datenbank verwendet. Es kann zwischen mehreren Datenbanksystemen gewählt werden. Beispielsweise: MySQL, Oracle und MS-SQL. Um ein einwandfreies Funktionieren zu garantieren, müssen noch die Namen der Datenbänke, die Adresse des Hosts, sowie Benutzername und Kennwort eingetragen werden.

Aktivierung der Regeln

Der letzte Schritt ist, die zu benutzenden Regeln zu aktivieren. Da für den Performancetest nur Standardregeln genutzt werden, genügt es, die Doppelkreuze vor den jeweiligen Regeln zu entfernen.

5.3 Allgemeiner Aufbau des Testnetzwerkes

Zum Erzeugen einer Netzwerklast wird ein kleines Performancetool namens “IPerf“ verwendet. Dabei werden ausschließlich TCP Pakete versendet. Zum

besseren Vergleich wird das Netzwerk voll ausgelastet, was in dem Beispiel einem Wert von 94 Megabit/Sekunde entsprach. Um IPerf nutzen zu können, bedarf es eines Servers und eines Klienten, auf denen jeweils das Tool installiert ist. Während des Tests sendet der Klient die Datenpakete zum Server. Es ist zu beachten, dass Snort und IPerf nicht gleichzeitig auf derselben Maschine laufen, da dies auf Kosten der Rechenleistung für Snort geht. Aus diesem Grund wurde ein professionelles Switch verwendet, welches die Technik des “port mirroring“ beherrscht. Beim “port mirroring“ werden die Daten eines Ports gespiegelt und zusätzlich auf einem anderen Port ausgegeben. Dadurch werden die ausgetauschten Daten von IPerf an ein drittes System gesendet, wo sie von Snort analysiert werden können. Aufgrund der Tatsache, dass die Entwicklungsplattform über nur einen Netzwerkanschluss verfügt, muss ein Hub dem Aufbau hinzugefügt werden (siehe hierfür Abbildung 4). Dadurch kann die Entwicklungsplattform auf des Network File System des Hosts zugreifen und zeitgleich die zu analysierenden Daten empfangen. Zuletzt müssen die IP-Adressen der einzelnen Systeme unter Beachtung der verschiedenen Subnetzwerke zugewiesen werden. Das ist notwendig, da sich in der Hostmaschine des Network File Systems zwei Netzwerkinterfaces befinden.

5.4 Performancetest von Snort auf der Entwicklungs-plattform

Wie in Abbildung 5.1 zu sehen ist, bedarf es eines relativ aufwändigen Testnetzwerks. Das System, das mit Hilfe von Snort den Datendurchsatz analysiert, ist rot markiert. Außerdem werden die beiden Subnetze mit

unterschiedlichen Farben (grün und rosa) dargestellt. Der Pfeil im Switch deutet an, welcher Port auf welchen gespiegelt wird.

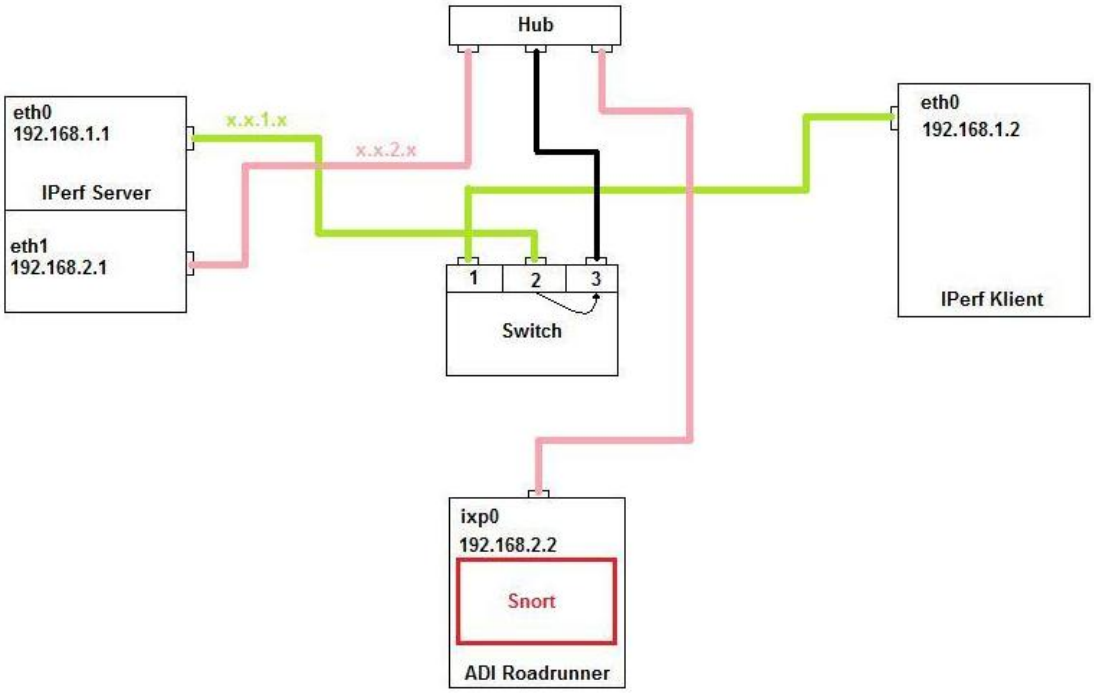

Abbildung 5.1: Testnetzwerk, Snort auf Entwicklungsplattform

Um den IPerf Server zu starten, wird folgendes Kommando verwendet:

iperf -s -w 32k -l 128k

Zum Starten des IPerf Klienten dient folgender Befehl:

iperf -c 192.168.1.1 -w 32k -l 128k -t 20

Hierbei zeigen “-c“ bzw. “-s“, ob es sich um Server oder Klient handelt. Die Attribute “-l“ und “-w“ bestimmen die Größe der TCP-Pakete und des Puffers. “-t“ gibt die Länge der Übertragung in Sekunden an.

Nach Beendigung der Übertragung und einer übertragenen Datenmenge von 220 MB gibt Snort die folgende Statistik aus:

```
===============================================================================
snort[69]: Packet Wire Totals:
snort[69]:    Received:       227761
snort[69]:    Analyzed:        67262 (29.532%)
snort[69]:     Dropped:       160498 (70.468%)
===============================================================================
snort[69]: Breakdown by protocol (includes rebuilt packets):
snort[69]:       ETH: 67262      (100.000%)
snort[69]:   ETHdisc: 0          (0.000%)
snort[69]:      VLAN: 0          (0.000%)
snort[69]:      IPV6: 0          (0.000%)
snort[69]:   IP6 EXT: 0          (0.000%)
snort[69]:   IP6opts: 0          (0.000%)
snort[69]:   IP6disc: 0          (0.000%)
snort[69]:       IP4: 67259      (99.996%)
snort[69]:   IP4disc: 0          (0.000%)
snort[69]:     TCP 6: 0          (0.000%)
snort[69]:     UDP 6: 0          (0.000%)
snort[69]:     ICMP6: 0          (0.000%)
snort[69]:   ICMP-IP: 0          (0.000%)
snort[69]:       TCP: 67254      (99.988%)
snort[69]:       UDP: 5          (0.007%)
snort[69]:      ICMP: 0          (0.000%)
snort[69]:   TCPdisc: 0          (0.000%)
snort[69]:   UDPdisc: 0          (0.000%)
snort[69]:   ICMPdis: 0          (0.000%)
snort[69]:      FRAG: 0          (0.000%)
snort[69]:    FRAG 6: 0          (0.000%)
snort[69]:       ARP: 3          (0.004%)
snort[69]:     EAPOL: 0          (0.000%)
snort[69]:   ETHLOOP: 0          (0.000%)
snort[69]:       IPX: 0          (0.000%)
snort[69]:     OTHER: 0          (0.000%)
snort[69]:   DISCARD: 0          (0.000%)
snort[69]: InvChkSum: 0          (0.000%)
snort[69]:    S5 G 1: 0          (0.000%)
snort[69]:    S5 G 2: 0          (0.000%)
snort[69]:     Total: 67262
===============================================================================
snort[69]: Action Stats:
snort[69]: ALERTS: 0
snort[69]: LOGGED: 0
snort[69]: PASSED: 0
```

Deutlich zu erkennen ist, dass es sich tatsächlich nur um TCP-Datenverkehr handelt. Lediglich drei Pakete wurden für das ARP genutzt. Von 227761 empfangenen Paketen konnte Snort nur 67262 analysieren. Das entspricht

29,5% der Gesamtmenge. Die fünf UDP-Pakete resultieren aus dem genutzten Network File System.

5.5 Performancetest von Snort auf dem Desktop-PC

Das Testnetzwerk ist im Vergleich zum Performancetest in Punkt 5.3 wesentlich einfacher aufgebaut. Dies ist in Abbildung 5.2 deutlich zu erkennen. Das analysierende System ist rot gekennzeichnet. Die Trennung der Subnetze ist mit unterschiedlichen Farben (grün und rosa) aufgezeigt. Der Pfeil im Switch deutet an, welcher Port auf welchen gespiegelt wird.

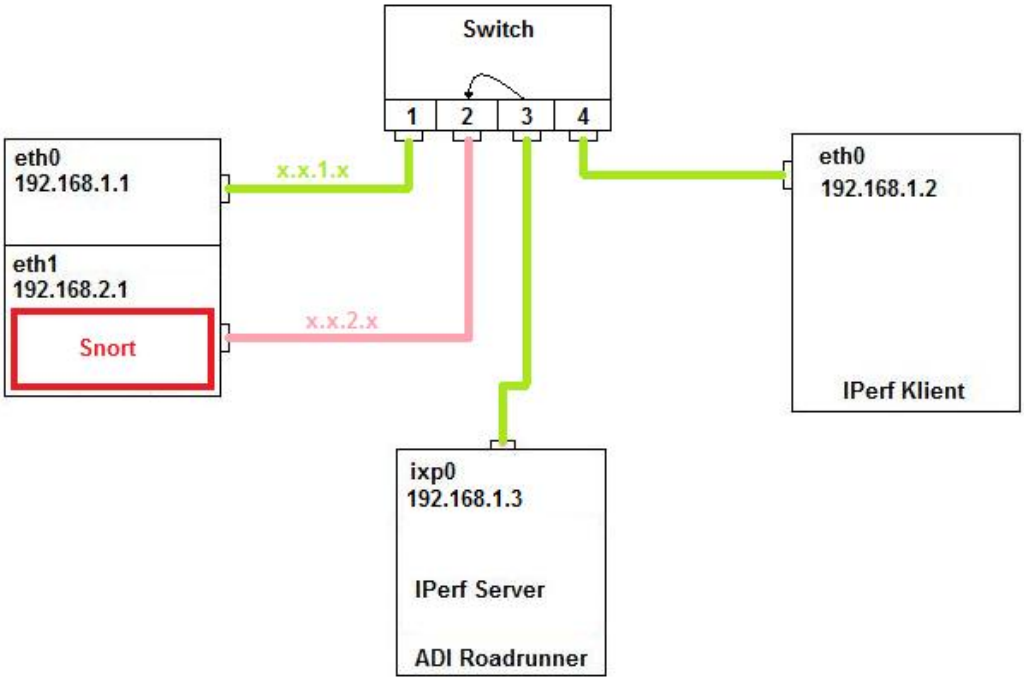

Abbildung 5.2: Testnetzwerk, Snort auf Desktop-PC

IPerf wird mit denselben Paket- und Puffergrößen gestartet, wie im Performancetest unter Punkt 5.3 bereits verwendet. Nach Beendigung der Übertragung und einer übertragenen Datenmenge von 220 MB gibt Snort folgendes Ergebnis aus:

```
===============================================================================
Packet Wire Totals:
   Received:       242586
   Analyzed:       242586 (100.000%)
    Dropped:            0 (0.000%)
Outstanding:            0 (0.000%)
===============================================================================
Breakdown by protocol (includes rebuilt packets):
      ETH: 242586     (100.000%)
  ETHdisc: 0          (0.000%)
     VLAN: 0          (0.000%)
     IPV6: 0          (0.000%)
  IP6 EXT: 0          (0.000%)
  IP6opts: 0          (0.000%)
  IP6disc: 0          (0.000%)
      IP4: 242586     (100.000%)
  IP4disc: 0          (0.000%)
    TCP 6: 0          (0.000%)
    UDP 6: 0          (0.000%)
    ICMP6: 0          (0.000%)
  ICMP-IP: 0          (0.000%)
      TCP: 242586     (100.000%)
      UDP: 0          (0.000%)
     ICMP: 0          (0.000%)
  TCPdisc: 0          (0.000%)
  UDPdisc: 0          (0.000%)
  ICMPdis: 0          (0.000%)
     FRAG: 0          (0.000%)
   FRAG 6: 0          (0.000%)
      ARP: 0          (0.000%)
    EAPOL: 0          (0.000%)
  ETHLOOP: 0          (0.000%)
      IPX: 0          (0.000%)
    OTHER: 0          (0.000%)
  DISCARD: 0          (0.000%)
InvChkSum: 0          (0.000%)
   S5 G 1: 0          (0.000%)
   S5 G 2: 0          (0.000%)
    Total: 242586
-------------------------------------------------------------------------------
Action Stats:
ALERTS: 0
LOGGED: 0
PASSED: 0
```

Von 242586 erhaltenen Paketen konnten 100% analysiert werden. Alle Pakete wurden dem TCP-Protokoll zugeordnet.

5.6 Vergleich der Ergebnisse

Die Ergebnisse fallen sehr unterschiedlich aus. Während Snort auf dem Desktop-PC alle Pakete prozessieren konnte, wurden auf der Entwicklungsplattform nur circa 30% der Pakete analysiert. Desweiteren scheinen insgesamt mehr Pakete am Desktop-PC angekommen zu sein, als an der Entwicklungsplattform. Es können mehrere Gründe für die unterschiedlichen Ergebnisse genannt werden.

- Die Rechenleistung der verwendeten Systeme
- Die Art der Umsetzung der Gleitkommaberechnung
- Der Aufbau der Testnetzwerke

Der Hauptgrund liegt in der technischen Ausstattung beider Systeme. Während der Desktop-PC mit einem Intel Pentium 4 Prozessor ausgestattet ist, der mit einer Taktung von 2,4 GHz läuft, steht der Entwicklungsplattform nur ein Intel XScale Prozessor mit 900MHz Taktfrequenz zur Verfügung. Neben den offensichtlichen Unterschieden beider Prozessoren, wie Taktfrequenz und Cache, spielt zusätzlich noch die Tatsache mit ein, dass der Intel Pentium 4 die Technik des Hyperthreading beherrscht. Hyperthreading ist die annäherend parallele Verarbeitung verschiedenster Aufgaben. Der Desktop-PC kann also alle Vorteile seines Prozessors ausnutzen. Der Netzwerk Prozessor des Entwicklungssystems hingegen hat andere Stärken, zum Beispiel die Möglichkeit der Programmierung der Microengines. Dieser Vorteil wurde nicht genutzt, da es für die reine Implementierung von Snort nicht notwendig war. So

musste der NPU mit wesentlich weniger verfügbaren Ressourcen exakt die gleiche Arbeit leisten, wie die CPU des Desktop-PCs.

Ein weiterer Grund liegt in der hardwareseitigen Realisierung der Gleitkommaberechnung des Intel Pentiums. Die Entwicklungsplattform hingegen greift auf eine softwarebasierte Emulation zurück. Dies bedeutet naturgemäß eine verringerte Leistung in derartigen Berechnungen.

Zuletzt unterscheiden sich die beiden Testnetzwerke maßgeblich. Im Testnetzwerk zur Perfomancemessung der Entwicklungsplattform befindet sich ein Hub als verlangsamendes Element. Dies könnte auch erklären, warum im Performancetest der Entwicklungsplattform insgesamt weniger Datenpakete angekommen sind, als im Test des Desktop-PCs.

5.7 Folgerung des Vergleichs

Wie in Punkt 5.5 bereits erwähnt, konnten die vollen Möglichkeiten des Netzwerk Prozessors nicht ausgeschöpft werden. Das IDS lief als einfache Software auf der Entwicklungsplattform. Um die wahre Leistung dieser Plattform nutzen zu können, muss Snort mit allen seinen Funktionen direkt in die Microengines eingefügt werden. Zur Realisierung dessen müssten die performancekritischen Teile des Quellkodes von Snort in Microcode oder MircoC umgeschrieben werden, da die Microengines des Netzwerkprozessors nur mit diesen Programmiersprachen arbeiten können. Zusätzlich sollten beim Umschreiben der Quellkodes alle Gleitkommaberechnungen aus dem Kode entfernt werden. Zuletzt müssten noch Microengines programmiert werden, um die beiden Gigabit-Ports der Entwicklungsplattform zu aktivieren, womit der Hub im Testnetzwerk überflüssig wäre. Mit den aufgezeigten Anpassungen würde der Netzwerkprozessor wesentlich bessere Ergebnisse erzielen.

Zur Realisierung dieser Aufgaben ist ein sehr fundiertes Wissen in den Programmiersprachen C, C++, MicroC sowie Microcode notwendig. Desweiteren muss der Befehlssatz erlernt werden, der von Intel eigens für die Microengines geschaffen wurde. Um die Abläufe der geschaffenen Programme korrekt zu erstellen, ist es weiterhin notwendig genauestens über die Funktionen, Speicheradressierung und das Zusammenspiel der Komponenten innerhalb des Netzwerkprozessors Bescheid zu wissen. Da zudem der Support für die Entwicklungsumgebung und dem Netzwerkprozessor eingestellt wurde, war es im Rahmen dieser Diplomarbeit nicht möglich die oben genannte Aufgabe zu realisieren. Es wurde sich darauf beschränkt eine theoretische Betrachtung des Netzwerkprozessors durchzuführen, sowie die Software zur Simulation der entwickelten Applikationen vorzustellen.

6 Internet Exchange Processor (IXP)

6.1 Grundlegendes

Intel bietet eine große Anzahl verschiedener Modelle der IXP Netzwerk Prozessoren. Bis jetzt gibt es zwei Generationen. Im Rahmen dieser Diplomarbeit wurde mit der zweiten Generation, namentlich IXP2xxx, gearbeitet. Die bekanntesten Vertreter sind der IXP2400 und der IXP2800. Hauptunterschiede zwischen den Vertretern der zweiten Generation sind Anzahl und Art der integrierten Funktionseinheiten, Grad der Parallelisierung, die Speichergröße und die interne Datenbandbreiten. Zusätzlich unterstützen einzelne Vertreter auch extra Features, wie zum Beispiel Kryptografie.

Die grundlegenden Features der IXP2xxx Familie sind:

- Ein eingebetteter RISC Prozessor

- Bis zu 16 programmierbare Microengines
- Mechanismen zur Synchronisation der Prozessoren
- Onboard Speicher
- Ein serielles Interface
- Verschiedene Anschlüsse für externe Speichermedien
- Verschiedene Anschlüsse für externe I/O Busse
- Ein Prozessor zur Streuwertberechnung und Kryptografie

6.2 Externe Anschlüsse

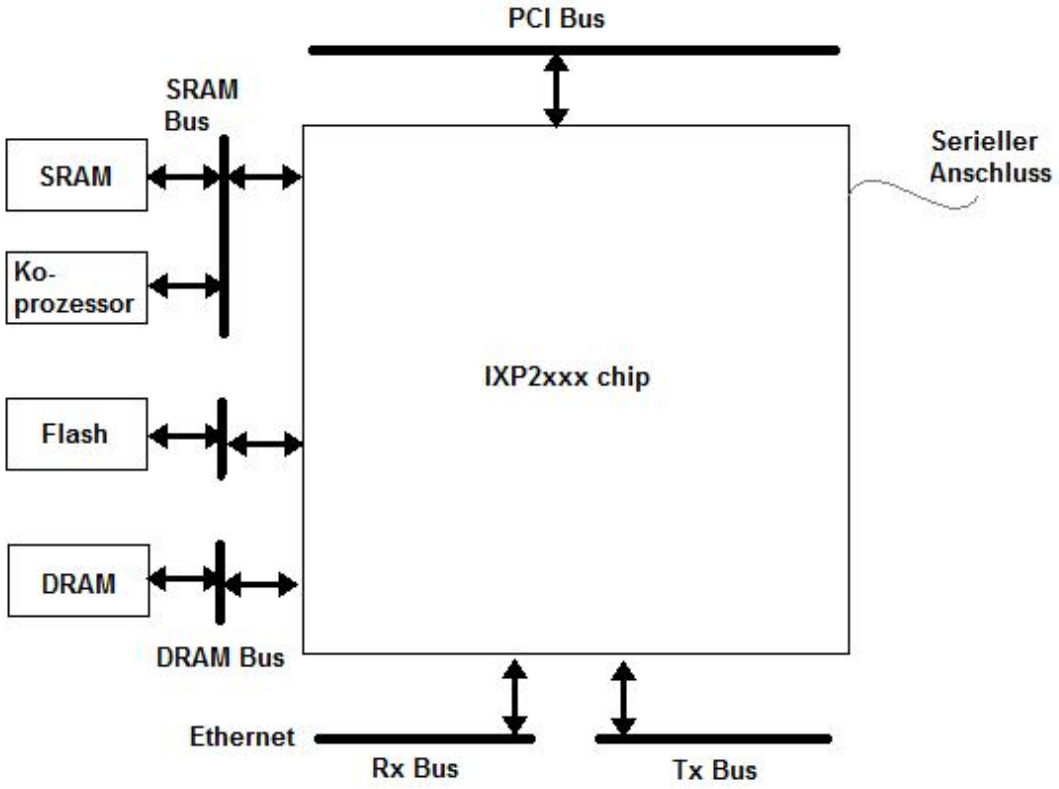

Abbildung 6.1: Externe Verbindungen

Wie in Abbildung 6.1 zu erkennen, verfügt der IXP2xxx Chip über mehrere externe Anschlüsse.

Serieller Anschluss

Mit Hilfe der UART-Hardware wird ein Interface bereitgestellt, das Verbindung zu einem konventionellen seriellen Anschluss schafft. Dieser Anschluss wird benötigt, um Zugriff auf Kontroll- und Managementfunktionen zu etablieren. Die

Geschwindigkeit reicht aus, um Befehle zu senden. Zum Übermitteln von Daten sollten allerdings andere Schnittstellen genutzt werden. Im Rahmen dieser Diplomarbeit wurde der serielle Anschluss genutzt, um auf das implementierte Linux zuzgreifen.

Ethernet Schnittstellen

Der IXP2xxx verfügt über einen 10/100 MBit Port, sowie über zwei 1 GBit Ports. Der 10/100 MBit Port wurde für verschiedene Aufgaben genutzt:

- Schaffen einer Netzwerkverbindung zwischen Entwicklungs-plattform und anderen Systemen zur Geschwindigkeitsermittlung
- Etablierung eines Network File Systems
- Übertragung der Ramdisk und des Kernelimages an die Entwicklungsplattform

SRAM- und DRAM Busse

Die Entwicklungsplattform stellt SRAM- und DRAM-Speicher zur Verfügung. Diese Speicher werden mit Hilfe der Busse an den Chip angeschlossen und werden unter Benutzung eindeutiger Adressierung angesprochen.

Flash Slowport Bus

Diese Verbindung erlaubt Zugriff auf ROM und FlashROM, welcher sich auf der Entwicklungsplattform befindet. Beispielsweise wird der Flashspeicher genutzt, um Ramdisk und Kernelimage zu speichern und von dort zu booten.

6.3 Prozessor-Hierarchie

Die IXP2xxx Familie nutzt eine vier-stufige Prozessorhierachie. Während sich drei Stufen direkt auf dem Chip befinden, werden eine von ihnen von externer Hardware bereitgestellt. Desweiteren sind nicht alle von ihnen programmierbar, bei Bedarf aber konfigurierbar. Folgend eine Auflistung der verwendeten Prozessoren:

XScale RISC Prozessor

Der eingebettete RISC Prozessor auf dem IXP2xxx ist ein XScale und ist programmierbar. Er basiert auf der ARM Architektur und wurde von Intel weiterentwickelt. Das genutzte Betriebssystem wird von dem XScale getragen. Desweiteren dient er als Schnittstelle zum Nutzen der Microengines und zum übertragen der Programme auf diese. Hauptsächlich werden von ihm alle Aufgaben übernommen, die auch ein herkömmlicher CPU in einem Desktop-PC hat.

Microengines

Microengines bilden die unterste Stufe des Datenaustausches zwischen I/O-Geräten. Mit ihrer Hilfe kann dieser Datenaustausch sehr schnell von statten

gehen. Des Weiteren sind die Mircoengines in der Lage, einfache Datenpakete, nach Wünschen des Programmierers, zu verarbeiten.

Koprozessoren

Ein IXP2xxx enthält verschiedene zusätzliche Koprozessoren, welche sich direkt im Chip befinden und nicht programmierbar sind. Sie übernehmen verschiedene Funktionen, wie zum Beispiel Streuwertberechnung, Timer mit Echtzeituhren und Kryptografie.

Physical Interfaces

Die Hardware ist weder eingebettet, noch programmierbar. Sie wird benötigt um Layer 1 (Bitübertragung) und Layer 2 (Sicherung) Funktionen zu implementieren.

6.4 Microengines

Da die Microengines und deren Programmierung einen wesentlichen und sehr wichtigen Bestandteil in der Entwicklung einer Applikation für einen Netzwerkprozessor einnehmen, wird im Folgenden ihre Funktion und Arbeitsweise näher erläutert.

Wie bereits erwähnt, bilden Microengines die unterste Stufe der Prozessorhierarchie. Die Verwendung und Programmierung kann demnach

hohe bis höchste Geschwindigkeit in der Paketverarbeitung schaffen. Wesentliche Punkte bezüglich ihres Aufbaus sind folgend aufgelistet:

- RISC Design
- Mehrere verschiedene Registertypen
- 1024 Bytes lokaler Speicher
- Eine Einheit zur Berechnung eines CRC-Wertes
- Ein ALU
- Zugriff auf verschiedene Koprozessoren

Während der IXP2400 acht und der IXP2800 sogar 16 Microengines besitzt, befinden sich im IXP2350 lediglich vier Stück. Zur Verbesserung der Geschwindigkeit sind die Microengines in der Lage, parallel zu arbeiten. Folgend einige Beispiele wofür Microengines genutzt werden:

- Prüfsummenberechnung -und überprüfung
- Klassifikation, Verarbeitung und Bearbeitung von Kopfdaten
- IPv4 und/oder IPv6 Paketweiterleitung unter Verwendung von Routingtabellen oder direkt
- Multi- und Demultiplexen

6.5 Applikation

6.5.1 Genereller Aufbau

Um funktionierende Software zu schaffen, ist es wichtig ihren Aufbau zu verstehen. Sie besteht in der Regel aus sogenannten “microblocks“ (engl.: Mikroblocks) und einer “core component“ (engl.: Kernkomponente).

Microblocks

Die Software für einen Netzwerkprozessor besteht aus einem oder mehreren Microblocks. Diese arbeiten entweder unabhängig voneinander, oder setzen einen anderen Microblock zum korrekten funktionieren voraus. Zur einfachen IPv4 Paketweiterleitung werden beispielsweise mindestens drei Microblocks benötigt, wobei ein Microblock die ankommenden Pakete direkt von dem Port in einen Speicher weiterleitet, der zweite die Pakete verarbeitet (zum Beispiel die Kopfdaten ändert, entsprechend dem neuen Ziel) und der letzte die Pakete auf einen Outputport gibt. In Abbildung 6.2 ist die einfache Funktion einer solchen Applikation dargestellt.

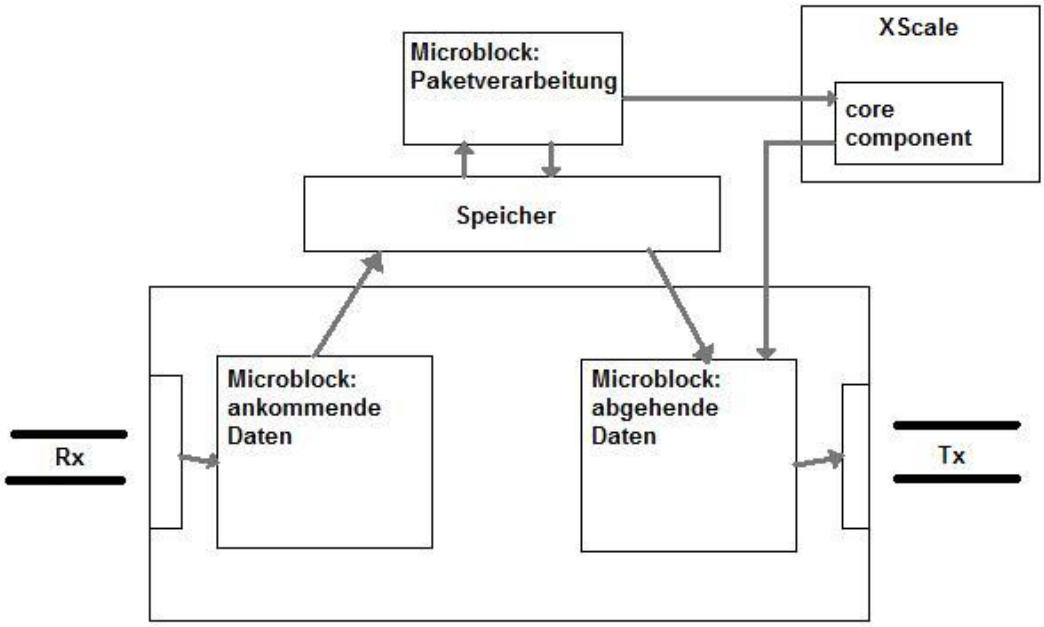

Abbildung 6.2: Kleine Applikation mit 3 Microblocks und Core Component

Durch hinzufügen weiterer Microblocks kann die Software beliebig komplex gemacht werden. Im Falle der drei Microblocks kann jeweils eine Microengine mit einem Microblock geladen werden. Dies bietet maximale Geschwindigkeit. Sollten mehr Microblocks als Microengines existieren, können die Microengines auch mehrere Aufgaben übernehmen.

Core components

Wie bereits erwähnt, sind die Microengines die beste Lösung für schnelle Paketverarbeitung. Der XScale wird nur für spezielle Ausnahmen genutzt. Erscheint also ein Paket, das nicht zu den Erwartungen eines Microblocks passt, wird es zum XScale und dessen Core component weitergeleitet. Ausnahmen sind beispielsweise Pakete vom ARP Protokoll. In diesem Fall weiß ein Microblock nicht, wie er mit dem Paket verfahren soll und überreicht es an

den XScale (siehe Abbildung 8). Der XScale ist nun in der Lage, direkt auf Linuxfunktionen zuzugreifen und das Paket zu verarbeiten. Desweiteren ist der XScale dafür zuständig, Microengines zu laden und zu starten.

6.5.2 Starten einer Applikation

Um die fertige Applikation zu starten, muss ein eigenes Programm kreiert werden, welches verschiedene Aufgaben übernimmt. Unter anderem sind dies folgende:

- Einfügen verschiedener Funktionen in den Linuxkernel mit Hilfe von zuvor erstellter Bibliotheken
- Implementierung des Programmkodes in die Microengines
- Laden und Starten der Microengines

Die Erschaffung des Programmes setzt gute Kenntnisse in der Programmiersprache C voraus. Desweiteren müssen die Manuals zu den einzelnen Funktionen genauestens gelesen werden. Im Rahmen dieser Diplomarbeit konnten folgende Erkenntnisse erzielt werden.

- Bevor irgendeine Microengine geladen werden kann, muss im Betriebssystem eine Schnittstelle zwischen Kernel und Microengines geschaffen werden. Dies geschieht in Form einer sogenannten Device (engl.: Gerät).
- Um den Programmkode in die Microengines zu laden wird eine Funktion namens “UcLo_<Befehl>“ verwendet

- Zum Starten und Laden der Microengines wird die Funktion “halMe_<Befehl>“ genutzt. Wichtig hierfür ist die Angabe des Wertes “meMask“ welcher verschiedene Funktionen der Microengine aktiviert bzw. deaktiviert.
- Nach dem erfolgreichen Starten und Laden befinden sich die Microengines in einem Zustand des “Horchens“. Das bedeutet, dass sie auf ein Signal warten um ihre Arbeit aufzunehmen.

Im Anhang A.3 dieser Arbeit befindet sich der Quellkode für ein solches Programm. In der Netzwerkprozessorterminologie wird solch ein Programm als “Loader“ bezeichnet.

7 Intel Exchange Architecture Software Development Kit 4.2

7.1 Vorwort

Das Software Development Kit von Intel dient als Unterstützung in der Arbeit mit einem Netzwerkprozessor aus dem Hause Intel. Dies beinhaltet Simulationssoftware, Entwicklungssoftware und Compiler. Der Programmierer kann mit Hilfe verschiedener, in die Software integrierte Tools seinen geschriebenen Kode überprüfen. Er hat dabei die Wahl, auf eine reine Simulation zurückzugreifen oder seine Software direkt auf dem Netzwerkprozessor zu testen.

Leider hat Intel den Support dieses Entwicklungskits eingestellt. Es ist nichtmehr direkt bei Intel erhältlich. Aus diesem Grund musste auf eine andere Version zurückgegriffen werden. Die Firma Netronome bietet eigene Plattformen mit Intel Netzwerkprozessoren an und stellt dazu ein Entwicklungskit bereit. Diese Software unterstützt zwar den IXP2350 Chip, allerdings werden keinerlei Testapplikationen bereitgestellt. Lediglich ein kleines Tutorial inklusive einer Applikation befindet sich im Softwarepaket. Diese Applikation wurde allerdings nicht für den hardwareseitigen Gebrauch geschaffen und kann nur zur Simulation verwendet werden. Alle anderen Beispielapplikationen wurden für Plattformen mit IXP2400 und IXP2800 Chips erstellt. Es wurde im Rahmen dieser Arbeit versucht, diese Applikationen zu portieren und für Plattformen mit einem IXP2350 Chip zu verwenden.

Es musste festgestellt werden, dass es nicht ohne weiteres möglich ist und es weit praktikabler wäre, eine Applikation komplett neu zu entwickeln. Die Gründe für die Inkompatibilität sind unter anderem:

- Unterschiedliche Anzahl von Microengines
- Verschiedenartige Schnittstellen zwischen Microcode und Microengines
- Während der IXP2400 bzw. IXP2800 jeweils ein Kommandofenster für Dateneingang (Ingress) und Datenausgang (Egress) besitzen, ist dies bei dem IXP2350 nicht der Fall.
- Teilweise wird im Programmkode ein bestimmter Speicherplatz direkt angesprochen. Die Adressierung unterschiedet sich allerdings zwischen den Produktversionen.

Aus diesen Gründen wird in dem folgenden Kapitel eine Übersicht über Funktion und Nutzen der Simulation gebracht. Dies geschieht unter Verwendung des Tutorials für den IXP2350.

7.2 Durchführen einer Simulation

Wie in Abbildung 7.1 zu sehen ist, bildet die sogenannte Workbench den Hauptrahmen der Entwicklungssoftware. In ihr vereint sind alle Tools zur Simulation, Entwicklung und Kompilation einer Applikation.

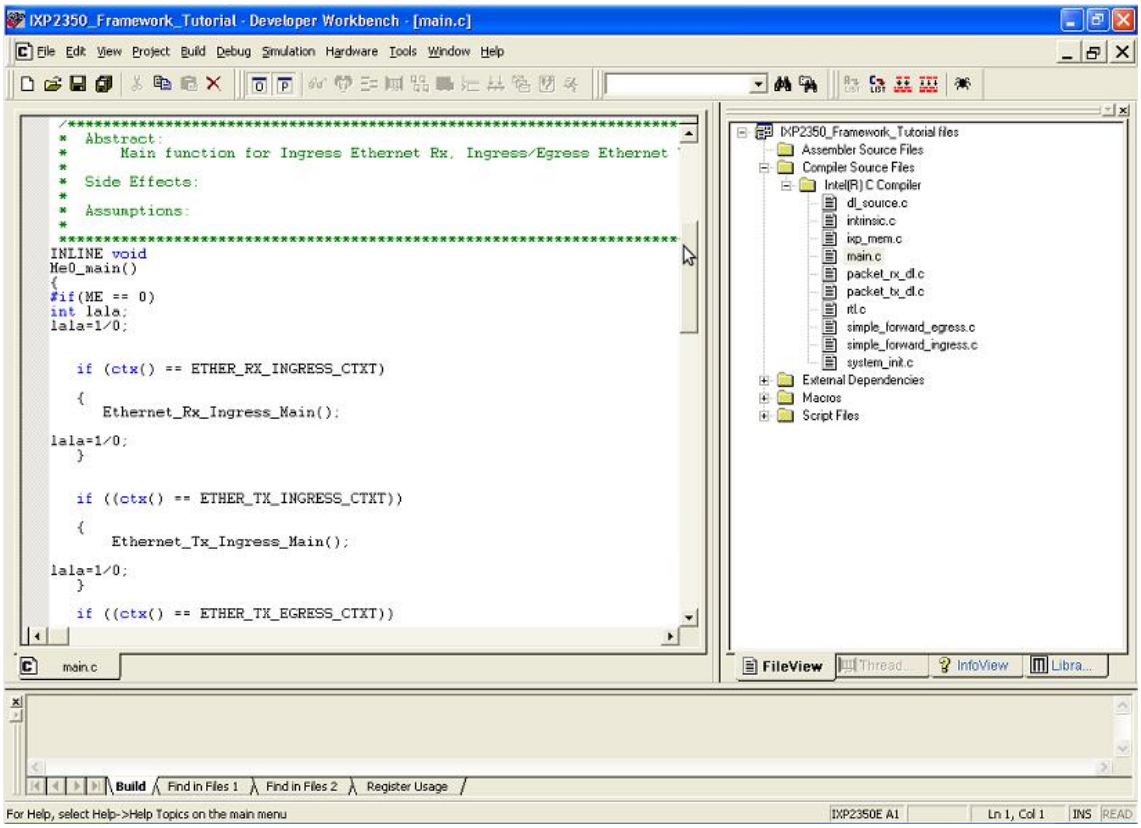

Abbildung 7.1: IXA SDK Workbench

Während in der rechten Bildhälfte die Verwaltung der Programmdateien stattfindet, wird die linke Seite zum Editieren des Programmkodes genutzt. Schlüsselbefehle und andere wichtige Eingaben werden dabei farblich hervorgehoben.

Bevor eine Simulation gestartet werden kann, muss zuerst der Programmkode kompiliert werden. Der Reiter hierfür befindet sich in der Aktionsleiste:

Build -> Rebuild

Sollte es sich um eine Applikation für verschiedene Chiparten handeln, muss zuvor der gewünschte Chip ausgewählt werden.

Project -> Chip Settings…

Sollten Fehler oder Warnungen während der Kompilation auftreten, erscheinen diese im unteren Teil der Workbench. Da es sich bei dem Beispielprogramm um ein fertiges und korrekt geschriebenes Programm handelt, sind keinerlei Fehler zu erwarten.

Nach erfolgreicher Kompilation kann der Debug-Modus wie folgt gestartet werden:

Debug -> Start Debugging

Nach einer kurzen Bearbeitungszeit befindet man sich im Debug-Modus und hat nun die Möglichkeit die Simulation zu starten.

Debug -> Run Control -> Go

Nun können verschiedene Tools genutzt werden, um die Applikation zu testen.

Zum Überprüfen der Speichernutzung kann das Schaltelement “Memory Watch“ in Form eines Widders ausgewählt werden. Wird dies getan, öffnet sich im unteren Teil des Programms ein Anzeigeelement. In Abbildung 7.2 ist dies zu erkennen.

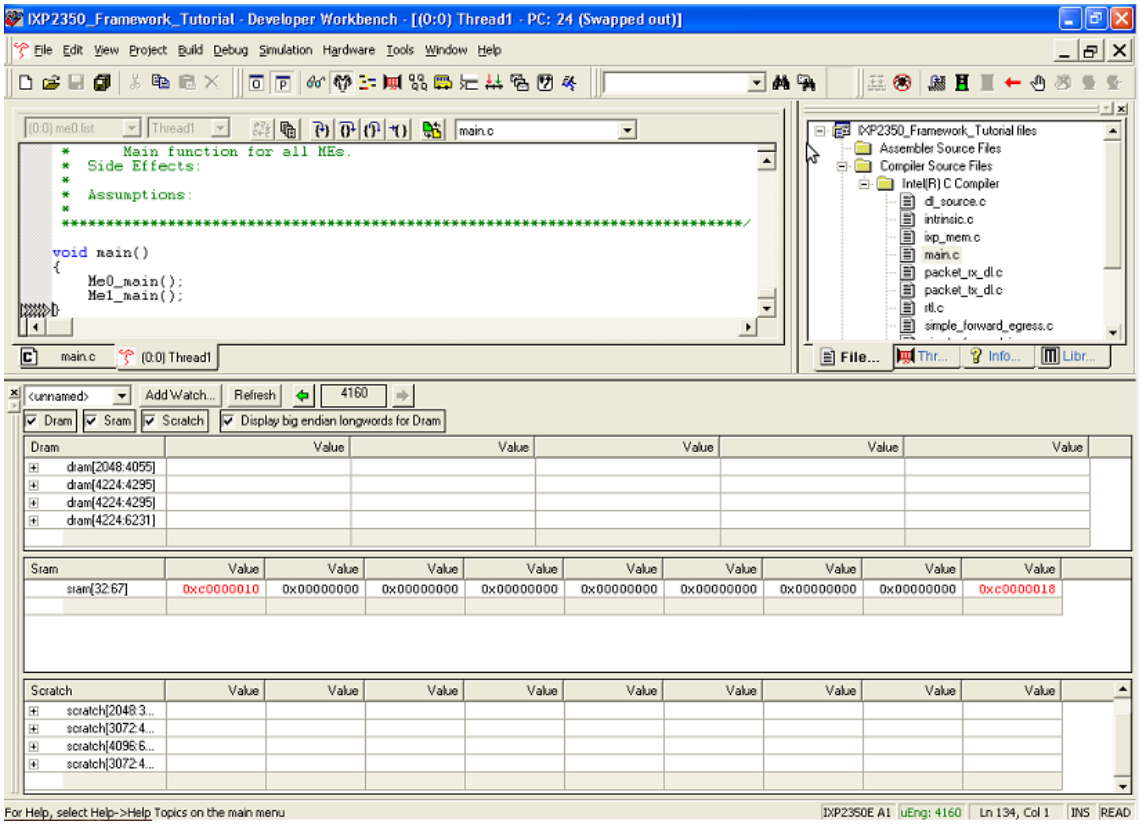

Abbildung 7.2: IXA SDK Workbench mit Memory Watch

In der Anzeige befinden sich Tabellen für den DRAM, SRAM und den Scratch Memory. In der ersten Spalte jeder Tabelle werden die einzelnen Bereiche des Speichers angezeigt. In den weiteren Spalten speichert das Programm nun die Werte, die sich in den jeweiligen Speicherbereichen befinden. Mit Hilfe dieses Tools kann der Entwickler überprüfen, ob sich die korrekten Werte in den jeweiligen Speicherbereichen befinden. Über den Button “Add Watch..“ kann der Programmierer außerdem spezielle Speicherbereiche anzeigen lassen.

Da es sich bei der Beispielapplikation um eine einfache IPv4 Weiterleitung handelt, sollte auch das Schaltelement “Packet Simulation Status“ in Form eines Busses genutzt werden. In Abbildung 7.3 ist zu erkennen, dass sich dabei auch ein Anzeigeelement in dem unteren Teil der Applikation öffnet.

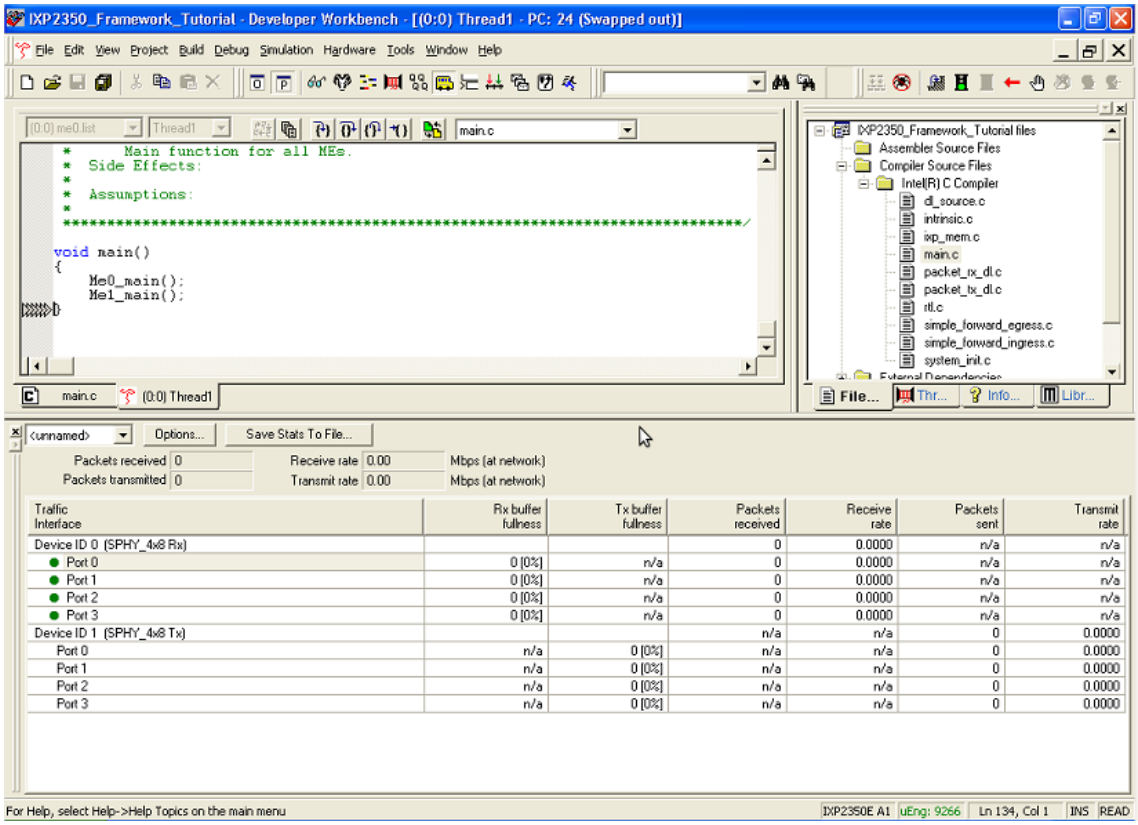

Abbildung 7.3: IXA SDK Workbench mit Packet Simulation Status

In diesem Anzeigeelement befindet sich wiederrum eine Tabelle, in welcher die verschiedenen Rx und Tx Ports gelistet sind. In den weiteren Spalten befinden sich Informationen zu den gesendeten Paketen. Da die Beispielapplikation lediglich zur Simulation erschaffen wurde, können keine Vergleichswerte zu einer hardwareseitigen Ausführung der Applikation erstellt werden. Somit kann nichts über die Echtheit der gewonnen Daten ausgesagt werden.

Es befinden sich eine Reihe weitere Tools in der Aktionsleiste. Beispielsweise das Element “Thread Status“ in Form eine Garnrolle. Hier kann die Anzahl der Microengines und ihr derzeitiger Status angezeigt werden. Außerdem existiert noch eine Kommandozeile mit deren Hilfe einzelne Befehle an den

Netzwerkprozessor gesendet werden können bzw. zur Anzeige der vom Programm genutzten Befehle.

7.3 Praktischer Nutzen der Entwicklungssoftware

Um eine Applikation für einen Netzwerkprozessor zu entwerfen, ist es nahezu unerlässlich eine Entwicklungssoftware zu verwenden. Wenngleich sie nicht zwingend erforderlich ist, schafft sie doch eine Umgebung, die den Entwurf von Software wesentlich vereinfacht. Außerdem beinhaltet die Entwicklungssoftware eine sehr ausführliche Dokumentation. Leider bezieht sich diese Dokumentation hauptsächlich auf die Netzwerkprozessoren der Produktline IXP24xx und IXP28xx, was die Entwicklung einer Applikation für den IXP23xx wesentlich schwieriger macht.

Im Rahmen dieser Diplomarbeit spielte die Entwicklungssoftware aus diesem Grund keine große Rolle. Sie diente lediglich zur Veranschaulichung bzw. zur einfacheren Kompilation des Quellkodes.

Hinderlich ist indes, dass die grafische Oberfläche der Entwicklungssoftware das Betriebssystem Microsoft Windows voraussetzt, während zur Kompilation der erforderlichen Bibliotheken eine Linux Distribution verwendet wird.

8 Zusammenfassung der Diplomarbeit

8.1 Aufgetretene Probleme

Während der Diplomarbeit traten eine Vielzahl von Problemen auf, die die Bearbeitung der Aufgaben zum Teil erheblich verzögerten.

- Ausgelaufener Support für Hardware
- Ausgelaufener Support für Software
- Abgabe der Zuständigkeit an andere Hersteller
- Bedarf an teurer kostenpflichtiger Software
  - Ausweichen auf freie Software nur bedingt möglich
- Zum Teil mangelhafte Dokumentationen

Größtes Hindernis hierbei war die Tatsache, dass der Support für die genutzten Hardware- und Softwaresysteme von allen Firmen eingestellt wurde. So verwies der Hersteller der Entwicklungsplattform (ADI Engineering) bei Fragen lediglich an den Chiphersteller, da er selbst keine weitere Unterstützung für die Entwicklungsplattform bieten konnte bzw. wollte. Intel, der Chiphersteller hingegen verwies zurück an den Hersteller der Entwicklungsplattform, da er den Support auch eingestellt hat. Betreffend Fragen zu der Software, sowohl der Entwicklungssoftware als auch der auf der Entwicklungsplattform genutzten Software, wurde jegliche Hilfe abgelehnt, da die Entwickler der Software an die Hardwareentwickler verwiesen und die Entwickler der Hardware an die Softwareentwickler. Desweiteren wurde zum Teil kostenpflichtige Software genutzt, die der Hardwareentwickler zur Bearbeitung voraussetzte, aber nicht bewusst mitlieferte. Dies betrifft vor allem Toolchains und Quellkode für den

Linuxkernel von MontaVista. Die Anfrage auf Unterstützung beantwortete MontaVista wiederrum damit, dass der Support für dieses Produkt ausgelaufen sei und Lizenzen dafür nur noch in Sonderfällen für einen absurd hohen Preis vergeben werden. Aus den genannten Gründen musste auf freie Software zurückgegriffen werden, die wiederum zusätzliche Probleme verursachte, da sie nicht speziell für diese Aufgabe vorgesehen war.

8.2 Das Ergebnis der Arbeit

Das Hauptziel, die Implementierung von Snort auf einen Netzwerkprozessor, konnte erfolgreich erfüllt werden. Das Programm wurde dahingehend kompiliert und installiert, dass es auf der Entwicklungsplattform genutzt werden kann. Allerdings musste festgestellt werden, dass die Lösung auf der Entwicklungsplattform keinerlei Vorteile gegenüber einem herkömmlichen Desktop-PC bietet. Vielmehr zeigte sich, dass ein Desktop-PC wesentlich bessere Ergebnisse erzielte. Der verwendete Desktop-PC ist zudem nicht auf dem aktuellen Stand der Technik. Würde man einen modernen Desktop-PC mit weitaus höherer Rechenleistung verwenden, wäre der Vorsprung des Desktop-PCs gegenüber der Entwicklungsplattform noch größer.

Die Aufgabe, durch verschiedene Testszenarien die Vor- und Nachteile des Netzwerkprozessors zu ermitteln, konnte nicht erfüllt werden. Dies liegt darin begründet, dass weder der Hersteller der Entwicklungsplattform, noch der Hersteller des Netzwerkprozessors, Beispielapplikationen mitlieferte. Mit ihrer Hilfe hätten zumindest rudimentäre Tests durchgeführt werden können. Ein Beispiel dafür wäre eine einfache IPv4 Weiterleitungsapplikation. In diesem Fall ist davon auszugehen, dass der Netzwerkprozessor im Vergleich zu einem Desktop-PC eine bessere Performance geliefert hätte. Um dennoch einen

Einblick in die Technik des Netzwerkprozessors und der Entwicklung von geeigneten Applikationen zu erhalten, wurden Kapitel erstellt, die bei der zukünftigen Arbeit mit dieser Hard- und Software als Hilfe dienen sollen. Sie beinhalten die wichtigsten bis dahin erhaltenen Erkenntnisse.

Die erzielten Leistungen der Diplomarbeit im Überblick sind:

- Erstellung verschiedener eigener Toolchains zum Cross-Kompilieren
- Herstellung eines lauffähigen Linuxkernel mit speziellen Optionen für den Roadrunner
- Etablierung eines Snortsensors und –servers
- Cross-Kompilation von Snort
- Leistungsvergleich von Snort auf unterschiedlichen Systemen
- Simulation einer Testapplikation für den IXP2350 mit Hilfe des Netronome SDK 4.2

8.3 Ausblick

Trotz der aufgetretenen Probleme im Rahmen dieser Arbeit konnte erkannt werden, dass Netzwerkprozessoren in verschiedenen Aufgaben die wesentliche bessere Alternative zu Desktop-PCs darstellen. Dies betrifft vor allem spezielle Netzwerkaufgaben, wie Routing oder Paketverarbeitung im Allgemeinen. Es ist allerdings fraglich, ob ein derart komplexes Programm wie Snort komplett in die Hardware integriert werden kann. Es ist wahrscheinlicher, dass diverse Teilaufgaben, vor allem bezüglich der Paketverarbeitung, direkt an die

Microengines übertragen werden müssen. Während Aufgaben, wie der Vergleich der Pakete mit Signaturen sowie die Übergabe der Ergebnisse an eine Datenbank, von dem XScale getragen werden sollten. Für diese Aufgabe muss der Programmierer über eine Expertise in mehreren Gebieten verfügen.

Literatur

[1] Douglas E. Comer: *Network Systems Design.* New Jersey: Pearson Education, 2006.

[2] Jack Koziol: *Intrusion Detection with Snort.* Indianapolis: Sams Publishing, 2003.

[3] Intel Corporation: *Intel IXP23XX Product Line Hardware Reference Manual,* 2005.

[4] Intel Corporation: *Software Framework Getting Started Guide,* 2005.

[5] Intel Corporation: *Intel IXP2350 – Software Framework Tutorial,* 2005.

[6] Intel Corporation: *Intel IXP2XXX – Architecture Tool User Guide,* 2005.

[7] Intel Corporation: *Intel XScale Core Support Libraries Reference Manual,* 2005.

[8] Intel Corporation: *Intel IXP2350 Network Processor Product Brief,* 2005.

[9] ADI Engineering, Inc.: *Roadrunner IXP23XX Reference Platform Users Manual,* 2004.

[10] @traktiv GmbH: *Vortrag über das IDS Snort,* Datum unbekannt.

[11] Prof. Dr. Jürgen Ehrensberger: *Foliensatz "Matériel de Réseau",* 2006.

[12] Internet: *Wikipedia – Die freie Enzyklopädie.* de.wikipedia.org/en.wikipedia.org, 2004.

[13] Internet: *OIF Optical Internetworking Forum.* http://www.oiforum.com/public/NPF_IA.html, 1998.

[14] Internet: *Ubuntu User.* http://ubuntuusers.de/, 2004.

[15] Internet: *Snort Homepage.* http://www.snort.org/, 1998.

[16] Internet: *HitchHiker Guide to the ENP-2611.* http://www.run.montefiore.ulg.ac.be/~martin/enp-faq.html, Datum unbekannt.

Eidesstattliche Erklärung

Ich versichere an Eides statt, dass ich die beiliegende Diplomarbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie alle wörtlich oder sinngemäß übernommenen Stellen in der Arbeit gekennzeichnet habe.
Ferner gestatte ich der Hochschule für Technik und Wirtschaft Dresden, die beiliegende Diplomarbeit unter Beachtung insbesondere urheber-, datenschutz- und wettbewerbsrechtlicher Vorschriften für Lehre und Forschung zu nutzen. Mir ist bekannt, dass für die Weitergabe oder Veröffentlichung der Arbeit die Zustimmung der HTW Dresden sowie der an der Aufgabenstellung und Durchführung der Arbeit unmittelbar beteiligten Partnereinrichtungen erforderlich ist.

.....
Datum/Unterschrift

Anlagenverzeichnis

A.1 Konfigurationsdatei von BASE (Auszug)

```
[..]
$DBlib_path = '/usr/local/apache/htdocs/php/adodb';


/* The type of underlying alert database
 *
 *  MySQL       : 'mysql'
 *  PostgresSQL : 'postgres'
 *  MS SQL Server : 'mssql'
 *  Oracle      : 'oci8'
 */
$DBtype = 'mysql';

/* Alert DB connection parameters
 *   - $alert_dbname   : MySQL database name of Snort alert DB
 *   - $alert_host     : host on which the DB is stored
 *   - $alert_port     : port on which to access the DB
 *   - $alert_user     : login to the database with this user
 *   - $alert_password : password of the DB user
 *
 *  This information can be gleaned from the Snort database
 *  output plugin configuration.
 */
$alert_dbname   = 'snortdb';
$alert_host     = '192.168.1.2';
$alert_port     = '';
$alert_user     = 'snort_dipl';
$alert_password = 'dipl09';

/* Archive DB connection parameters */
$archive_exists   = 1; # Set this to 1 if you have an archive DB
$archive_dbname   = 'snortarchivedb';
$archive_host     = '192.168.1.2';
$archive_port     = '';
$archive_user     = 'acid_dipl';
$archive_password = 'dipl09_acid';

/* Type of DB connection to use
 *   1  : use a persistant connection (pconnect)
 *   2  : use a normal connection (connect)
 */
$db_connect_method = 1;
[..]
```

A.2 Konfigurationsdatei von Snort (Auszug)

```
[..]
# var HOME_NET [10.1.1.0/24,192.168.1.0/24]
#
# MAKE SURE YOU DON'T PLACE ANY SPACES IN YOUR LIST!
#
# or you can specify the variable to be any IP address
# like this:
var HOME_NET any
# Set up the external network addresses as well.  A good start may
#be "any"
var EXTERNAL_NET any

[..]

# above to use Stream5.
#
# See README.stream5 for details on the configuration options.
#
# Example config (that emulates Stream4 with UDP support compiled
#in)
preprocessor stream5_global: max_tcp 8192, track_tcp yes, \
                               track_udp no
preprocessor stream5_tcp: policy first, use_static_footprint_sizes
# preprocessor stream5_udp: ignore_any_rules

[..]

# unicode.map should be wherever your snort.conf lives, or given
# a full path to where snort can find it.
preprocessor http_inspect: global \
    iis_unicode_map unicode.map 1252
preprocessor http_inspect_server: server default \
    profile all ports { 80 8080 8180 } oversize_dir_length 500

[..]

# no_alert_incomplete - don't alert when a single segment
#                       exceeds the current packet size
preprocessor rpc_decode: 111 32771

[..]

preprocessor bo

[..]

output database: log, mysql, user=snort_dipl password=dipl_09
dbname=snortdb host=192.168.1.2

[..]
```

A.3 Quellkode für Microengine Loader

```
#include <uclo.h>
#include <hal_mev2.h>
#include <halMev2Api.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "utils.h"

unsigned int globalAssignMes;

int main() {
  unsigned int meMask=0x0f, assignMes, cycles=200;
  int ret;
  static void *objHandle = NULL;
  unsigned char me;
  int ret4;
  int chkInactive;
  int waitNumSimCycles(unsigned char me, unsigned int cycles, int
chkInactive);

  UcLo_InitLib();
  UcLo_InitLibUeng(meMask);

  if ((ret = UcLo_LoadObjFile(&objHandle,"IXP2350_Framework_Tutorial.uof"))
!= UCLO_SUCCESS) {
    printf ("*** load failed *******\n");
    exit (1);
  }
  if (UcLo_WriteUimageAll(objHandle) == UCLO_NOOBJ) {
    printf ("n fehler\n");
    exit (1);  }

  if (UcLo_VerifyUengine(objHandle,0) != UCLO_SUCCESS) {
        printf ("ME 0 code not matched\n");
  }
  halMe_Init(meMask);
      assignMes = UcLo_GetAssignedMEs(objHandle);
      globalAssignMes = assignMes;

      for(me=0; me < 0x18; me++){
          if(!(assignMes & (1<< me))) continue;
          halMe_Start(me, meMask);
            printf ("ME %d\n",me);

      if ((ret4 = halMe_IsMeEnabled(me)) != HALME_ENABLED)
      printf ("ME not enabled!\n");
       else printf ("ME enabled!\n");}
            for(me=0; me < 0x08; me++){
          if(!(assignMes & (1<< me))) continue;
          waitNumSimCycles(me, cycles, chkInactive);
          halMe_Stop(me, meMask);}
  return(0);
}
```